Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Triennale in Informatica

SIMULAZIONE DEL VARIATIONAL QUANTUM EIGENSOLVER APPLICATO A PROBLEMI DI OTTIMIZZAZIONE

Relatore:
Chiar.ma Prof.ssa
ELISA ERCOLESSI

Presentata da:
PETRU MARCEL MARINCAS

Correlatore:
Chiar.mo Prof.
UGO DAL LAGO

IV Sessione
Anno Accademico 2023/2024

Sommario

Questa tesi esplora l'applicazione del Variational Quantum Eigensolver (VQE), un algoritmo euristico ibrido quantistico-classico, al traveling salesman problem (TSP) ed al knapsack problem (KP), con un'analisi comparativa delle prestazioni rispetto a metodi esatti ed euristici classici, discutendone l'utilità pratica odierna. L'algoritmo nella fase sperimentale è stato implementato e simulato utilizzando computer classici attraverso la SDK Qiskit. I risultati ottenuti mostrano che l'algoritmo VQE ha diversi aspetti che lo rendono problematico, sia legati ai tempi di convergenza ad una soluzione che legati alla qualità delle soluzioni che ottiene, ma rimane al contempo un algoritmo interessante da esplorare in maniera accademica. Questo lavoro fornisce quindi una panoramica delle potenzialità future e delle limitazioni attuali dell'uso del VQE per la soluzione dei problemi di ottimizzazione.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Nella nostra quotidianità, i computer ricoprono una vasta gamma di utilizzi sia ricreativi che di grande importanza: da applicazioni web come i social network alla simulazione di sistemi fisici, dalla grafica 3D all'uso di intelligenze artificiali che supportano decisioni critiche in ambito medico, industriale e finanziario. Nonostante questo, vi sono dei limiti intrinsechi che sono associati alla computazione delle macchine classiche, ovvero quelle che usano bit e porte logiche per l'elaborazione di dati.

Uno di questi limiti è legato alla **complessità computazionale** dei problemi che rientrano nella classe *NP-hard*, per i quali il tempo richiesto per trovare una soluzione esatta cresce esponenzialmente al crescere della dimensione del problema. Quando le dimensioni superano determinate soglie, questa loro caratteristica li rende impossibili da risolvere in tempi ragionevoli. Tuttavia, molti di questi problemi sono di grande interesse pratico, in quanto nella categoria *NP-hard* rientrano parecchi problemi di ottimizzazione, di simulazione di sistemi fisici e di crittografia.

Per andare a superare questi limiti non ci è sufficiente incrementare la potenza di calcolo delle macchine. Gli ostacoli imposti da limiti fisici, come le dimensioni, il calore e l'efficienza, si manifestano come barriere insormontabili e che ci bloccano da un miglioramento perpetuo della loro velocità[1]. Una possibile soluzione è quella di cambiare il paradigma con il quale affrontiamo questi problemi: uno dei modi che abbiamo trovato per farlo è attraverso l'uso di **algoritmi euristici**, che utilizzano strategie di ricerca intelligenti per trovare soluzioni approssimate in tempi ragionevoli.

Questi algoritmi non ci garantiscono pertanto di trovare la soluzione migliore che esiste, a differenza degli *algoritmi esatti*, ma ci danno in un tempo estremamente ragionevole una soluzione che si cerca di avvicinare il più possibile a quella ottimale. Questo è ideale per problemi in cui la soluzione ottimale è troppo costosa da trovare, ma dove ottenere una soluzione migliore aiuta a

diminuire i costi o aumentare l'efficienza di un sistema reale il cui modello si basa su un problema *NP-Hard*.

Prendiamo come esempio il problema del commesso viaggiatore: un commesso viaggiatore deve attraversare un insieme di città, visitandole tutte una sola volta e tornando infine alla città di partenza, cercando di minimizzare la distanza che percorre per attraversarle. Questo è un problema *NP-Hard* e sebbene esistano algoritmi esatti per risolverlo diventano impraticabili per un numero di città elevato. Tuttavia esso sta alla base di molti problemi reali, come la pianificazione di rotte per la logistica oppure la progettazione di circuiti elettronici. In questi casi, avere una soluzione approssimata in tempi ragionevoli è preferibile a non avere soluzione affatto.

Diversi metodi euristici esistono e lo stato dell'arte è in continua evoluzione, con approcci che spaziano dagli algoritmi genetici alle tecniche di ottimizzazione basate su swarm intelligence. Tuttavia, con l'avanzare della tecnologia, emerge una nuova frontiera nel campo dell'ottimizzazione: **il calcolo quantistico**. Questo paradigma sfrutta i principi della meccanica quantistica per elaborare informazioni in modi inaccessibili ai computer classici, offrendo potenzialmente soluzioni più efficienti per problemi complessi.

Uno degli **algoritmi quantistici** usati nel contesto delle ottimizzazioni euristiche è il **Variational Quantum Eigensolver** (**VQE**). Il VQE è un algoritmo ibrido che combina circuiti quantistici parametrizzati e metodi di ottimizzazione classica per trovare gli stati fondamentali di Hamiltoniane quantistiche. Sebbene il suo utilizzo principale sia nella chimica quantistica, la sua versatilità lo rende applicabile anche a problemi di ottimizzazione combinatoria. In questi casi, il problema viene riformulato come un'Hamiltoniana il cui stato fondamentale corrisponde alla soluzione ottimale del problema originale.

Nonostante le potenzialità promettenti, il calcolo quantistico è ancora in una fase relativamente embrionale, con sfide significative legate alla coerenza quantistica, agli errori dei gate e alla scalabilità dei circuiti. Tuttavia, i progressi nella tecnologia dei qubit e nello sviluppo di *Quantum Processing Units* (*QPU*) stanno lentamente colmando queste lacune, aprendo la strada ad applicazioni pratiche che si avvicinano sempre di più all'essere vantaggiose rispetto alle soluzioni classiche.

Questa tesi si propone di esplorare l'applicazione del VQE ai problemi del commesso viaggiatore (*Travelling Salesman Problem* o *TSP*) e dello zaino

(*Knapsack Problem* o *KP*), confrontando le prestazioni degli algoritmi euristici quantistici con quelli classici. Attraverso una serie di esperimenti e analisi comparative, si valuterà l'efficacia del VQE nel trovare soluzioni approssimate di alta qualità in tempi ragionevoli, nonché il suo potenziale vantaggio rispetto ai metodi tradizionali. Inoltre, si esamineranno le implicazioni pratiche dell'adozione di algoritmi quantistici nell'ottimizzazione di sistemi reali, considerando aspetti quali la scalabilità, la robustezza e l'efficienza computazionale. Attraverso il confronto tra approcci quantistici e classici, questa tesi intende delineare le prospettive future e le limitazioni di questi approcci, offrendo spunti per ulteriori ricerche e applicazioni pratiche in un'era di rapida evoluzione tecnologica.

Struttura della tesi

La tesi è divisa in tre parti principali:

1. **Teoria** - In questa parte introduttiva andiamo a presentare al lettore gli argomenti di cui tratterà la tesi, dapprima mostrando i concetti che stanno alla base dei computer quantistici, poi progredendo verso l'introduzione formale e rigorosa dei problemi del TSP e del KP, per poi arrivare agli algoritmi classici usati per risolverli. Infine, introdurremo il VQE, spiegandone il funzionamento e le sue caratteristiche principali.

2. **Implementazione e sperimentazione** - Proseguiremo descrivendo l'implementazione dell'algoritmo e le istanze dei problemi scelti, mostrando i vari passaggi che ho seguito per ottenere i miei risultati.

3. **Analisi e conclusioni** - Infine andremo a discutere i risultati ottenuti, a fare confronti tra le prestazioni degli algoritmi, a cercare di capire quali sono i punti di forza e di debolezza dell'approccio quantistico, e a trarre delle conclusioni su quali sono le prospettive future di questi algoritmi e di questa ricerca.

Queste parti sono suddivise ciascuna in uno o più capitoli che andranno a trattare in modo più dettagliato gli argomenti sopra citati. Ampio spazio sarà dedicato alla parte teorica, in quanto la computazione quantistica è un argomento che spazia dalla fisica alla matematica all'informatica, richiedendo conoscenze approfondite per essere compreso appieno.

La parte pratica e di analisi cercherà di non omettere dettagli implementativi, mantenendo il codice ove utile per mostrare il procedimento seguito. Il codice, insieme alla tesi, sarà accessibile su GitHub[1], in modo che questi esperimenti possano venire ripetuti ed ampliati da chiunque ne sia interessato.

[1]GitHub repository con il codice sorgente:
`https://github.com/PetruMr/VQE_Simulation_for_TSP-KP`

Capitolo 1

La computazione quantistica

Questo capitolo si occupa di fornire un'*introduzione alla computazione quantistica*, presentando i concetti più importanti dei computer quantistici, i postulati che sono alla base del loro funzionamento ed una breve panoramica sulle origini e lo stato attuale di questa tecnologia.

1.1 Introduzione ai computer quantistici

La nascita dei computer quantistici vede le sue origini tra la fine degli anni '70 e l'inizio degli anni '80. Diversi eventi in quel periodo, tra cui gli articoli di Paul Benioff[2][3], la conferenza del '81 tenuta all'MIT di Beinoff[4] e Richard Feynman[5] ed altri articoli interessanti[6], hanno collaborato a generare l'idea che poi è stata formalizzata nel 1985 da David Deutsch: il *computer quantistico universale*[7], l'equivalente quantistico di una macchina di Turing universale, il quale può simulare efficientemente - ovvero con un overhead polinomiale - qualsiasi altro tipo di macchina quantistica o classica.

Quando parliamo di computer quantistici, oggi ci riferiamo principalmente a dispositivi che operano su singoli atomi o particelle in una maniera controllata ed estremamente precisa, sfruttando i principi fondamentali della meccanica quantistica. Le soluzioni adottate per la realizzazione di queste macchine in realtà sono molteplici, e vanno dai superconduttori ai fotoni, passando per ioni intrappolati e materiali topologici, in quanto in questo ambito di continua ricerca e scoperte non esiste ancora un'unica tecnologia che si è affermata come la migliore. In generale, ci troviamo attualmente nell'**era NISQ** (*Noisy Intermediate-Scale Quantum*), in cui i computer quantistici disponibili sono di

piccole dimensioni e soggetti a rumore ed errori, ma che comunque possono essere usati per eseguire algoritmi quantistici e sperimentare nuove idee.

A differenza di un computer classico, il quale fa uso di *bit* e *porte logiche* per eseguire computazioni, in un computer quantistico abbiamo **qubit** e **quantum gate**. Se i bit possono essere o 0 o 1, i qubit possono anch'essi assumere gli stessi valori, rappresentati rispettivamente come $|0\rangle$ e $|1\rangle$, ma possono anche essere in un qualsiasi *stato intermedio* tra 0 e 1, ovvero possono essere in uno stato di **sovrapposizione**. I quantum gate sono il modo che abbiamo di eseguire operazioni controllate sui qubit, modificando quindi il loro **stato**. Quando poi un qubit che si trova in uno stato di sovrapposizione viene **misurato**, ovvero se ne osserva il valore, esso collassa in uno stato definito, ovvero diventa necessariamente o $|0\rangle$ o $|1\rangle$, ma in maniera **probabilistica**. Questo fenomeno è interamente quantistico e non ha corrispondenza in un computer classico.

Oltre a questo, esiste un'altra proprietà fondamentale che è puramente quantistica, ovvero quella dell'**entanglement**, la proprietà per cui due qubit possono essere correlati tra loro - cioè che la misura di uno determini i risultati dell'altro, anche a distanze elevate.

Questi fenomeni sono alla base degli algoritmi quantistici, come per esempio l'algoritmo di ricerca di Grover o l'algoritmo di fattorizzazione di Shor, ma anche dell'algoritmo che andremo a trattare dettagliatamente in questa tesi, ovvero il **Variational Quantum Eigensolver** (**VQE**).

Mostriamo quindi ognuno dei concetti citati in questa introduzione nel loro dettaglio, in modo da rendere note le basi che sono utili alla comprensione dei capitoli successivi. Ci concentreremo in particolare ad esporre **come funziona un computer quantistico**, senza entrare necessariamente da subito nei dettagli formali della meccanica quantistica che sta alle spalle di questi concetti. Solo successivamente metteremo in relazione questi argomenti, in modo da completare il quadro generale del funzionamento dei computer quantistici. Una descrizione dettagliata di questi concetti si trova in [8].

1.1.1 Qubit e misure

Se nei calcolatori classici abbiamo i *bit* come elemento fondamentale per la computazione e l'informazione classica, alla base dei computer quantistici vi

sono i **qubit** che servono ad uno scopo analogo.

Proprio come i bit sono descritti da uno **stato**, ovvero da una variabile logica che può essere o 0 o 1, anche i qubit possono essere descritti da stati. Abbiamo $|0\rangle$ e $|1\rangle$ che corrispondono all'analogo classico, ma oltre a questi due essi possono anche trovarsi in una qualsiasi loro *combinazione lineare*, chiamata *superposizione*:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

La notazione "$|\ \rangle$" è nota come *notazione di Dirac* ed è usata per rappresentare i vettori di uno spazio di Hilbert, lo spazio in cui vivono i qubit.

All'interno della formula (1.1), $|\psi\rangle$ è chiamato lo *stato* del qubit, α e β sono due numeri complessi che ci *esprimono* rispettivamente quanto il "*qubit è*" nello stato $|0\rangle$ e "*quanto è*" nello stato $|1\rangle$.

A differenza di un bit, che possiamo osservare quando vogliamo per capirne interamente le caratteristiche del suo stato, ovvero se in quel determinato istante esso ha come valore 0 oppure 1, noi non possiamo osservare direttamente un qubit, cioè non possiamo semplicemente misurarlo e capire i valori di α e β. Quando noi effettuiamo la **misura** sul qubit, otteniamo $|0\rangle$ con **probabilità** di $|\alpha|^2$ ed analogamente otteniamo $|1\rangle$ con **probabilità** di $|\beta|^2$. Le probabilità totali pertanto dovranno essere 1, quindi deve valere la **proprietà di normalizzazione** $|\alpha|^2 + |\beta|^2 = 1$. Dopo che la misura viene effettuata, il qubit **collassa** in uno dei due stati, ovvero diventa o $|0\rangle$ o $|1\rangle$: se noi andiamo ad eseguire un'altra misura subito dopo otterremo quindi lo stesso risultato di quella precedente, in quanto il qubit è collassato in uno stato definito.

Esempio 1.1.1

Uno stato possibile di un qubit, chiamato **stato di massima sovrapposizione** e noto come $|+\rangle$, è:

$$|\psi\rangle = |+\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \tag{1.2}$$

Questo qubit, quando misurato, ha probabilità del 50% di essere misurato come $|0\rangle$ e del 50% di essere misurato come $|1\rangle$ - poiché le coefficienti delle componenti $|0\rangle$ e $|1\rangle$ hanno modulo quadrato pari a $|\alpha|^2 = |\beta|^2 = \left|\frac{1}{\sqrt{2}}\right|^2 = 0.5$.

Dopo la misura, se per esempio supponiamo sia stato misurato come $|0\rangle$, il qubit diventerà $|\psi\rangle = |0\rangle$ e quindi tutte le misurazioni successive daranno sempre $|0\rangle$. Notiamo che la misura ha lo stesso comportamento per $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, ma essi hanno una differenza di **fase**, la quale li rende a tutti gli effetti due stati distinti e non equivalenti - in particolare in contesti di interferenza quantistica.

Da notare che $|0\rangle$ e $|1\rangle$ sono *stati speciali*, essi sono *stati della base computazionale* $\mathcal{B} = \{|0\rangle, |1\rangle\}$ e formano una base ortonormale di uno spazio vettoriale complesso di dimensione 2. Matematicamente li indichiamo come vettori ed essi sono:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \quad \Longrightarrow \quad |\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \tag{1.3}$$

Come possiamo notare da (1.3), essi sono vettori colonna di due elementi, e sono tra loro ortogonali, ovvero il loro prodotto scalare è nullo. Questa non è l'unica possibile base: possiamo scegliere una base diversa, ad esempio $\mathcal{B}' = \{|+\rangle, |-\rangle\}$, dove:

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \quad |-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \tag{1.4}$$

Noi possiamo eseguire le **misure** in una qualsiasi base computazionale, e la scelta della base influenza i risultati che otteniamo: misurare $|+\rangle$ con la base $\mathcal{B}$ ci darà $|0\rangle$ con probabilità 0.5 e $|1\rangle$ con probabilità 0.5, ma misurare $|+\rangle$ con la base $\mathcal{B}'$ ci darà $|+\rangle$ con probabilità 1 e $|-\rangle$ con probabilità 0.

Visto che $|\alpha|^2 + |\beta|^2 = 1$, possiamo anche scrivere lo stato di un qubit utilizzando le **coordinate polari** di α e β, ovvero $\theta \in [0, \pi]$ e $\phi \in [0, 2\pi]$:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{1.5}$$

Possiamo usare questo per rappresentare il qubit usando la **sfera di Bloch**. La sfera di Bloch è una sfera unitaria, ovvero di raggio 1, ed ogni suo punto sulla superficie rappresenta i possibili stati che può assumere un qubit.

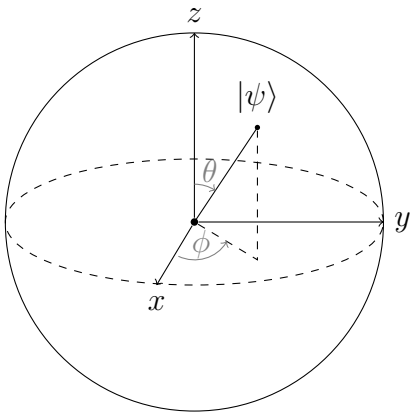

Figura 1.1: Sfera di Bloch con raffigurato lo stato $|\psi\rangle$

Questo ci è utile per visualizzare un qubit, anche se è limitata al caso di un singolo qubit in quanto non vi è un modo semplice per rappresentare un sistema a più qubit usando sfere di Bloch.

Da un punto di vista fisico, i due possibili stati $|0\rangle$ e $|1\rangle$ rappresentano due livelli energetici del sistema che si usa per implementare il qubit: nel caso di un fotone, possiamo usare la sua polarizzazione $|V\rangle$, verticale, e $|H\rangle$, orizzontale, come base computazionale, per un atomo possiamo considerare i livelli di energia $|G\rangle$, il suo stato fondamentale, ed $|E\rangle$, il suo stato eccitato, come base computazionale, e così via. Non andremo a discutere troppo su come questo venga implementato da un punto di vista pratico, ma rimane interessante e soprattutto importante per riuscire a concretizzare i concetti - tutti molto reali, seppur sorprendenti - che noi trattiamo del funzionamento di questi dispositivi.

1.1.2 Sistemi a più qubit

Un sistema di 2 bit può assumere 4 possibili stati: 00, 01, 10, 11. In modo analogo, un sistema di 2 qubit ha 4 stati della base computazionale che sono $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$. Un generico stato di un qubit $|\psi\rangle$ è una combinazione lineare di questi 4 stati:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \tag{1.6}$$

Anche qui abbiamo le stesse proprietà di prima: α_n sono numeri complessi che rappresentano le possibilità di un risultato nella misura e che devono sod-

disfare la condizione di normalizzazione, ovvero $|\alpha_{00}|^2+|\alpha_{01}|^2+|\alpha_{10}|^2+|\alpha_{11}|^2 = 1$.

Possiamo anche scegliere di misurare **un solo qubit**, per esempio immaginiamo di misurare il primo qubit ed ottenere il risultato $|1\rangle$: il nuovo stato del sistema sarà quindi:

$$|\psi'\rangle = \frac{\alpha_{10}|10\rangle + \alpha_{11}|11\rangle}{\sqrt{|\alpha_{10}|^2 + |\alpha_{11}|^2}} \tag{1.7}$$

Quindi esso si potrà trovare in uno dei due stati dove il primo qubit è $|1\rangle$, con una nuova probabilità che è stata normalizzata rispetto alla probabilità iniziale, mantenendo la condizione di normalizzazione.

Se si vuole rappresentare usando dei vettori questi stati, essi sono il **prodotto tensoriale** dei singoli vettori che rappresentano i singoli qubit:

$$|x_1\rangle \otimes |x_2\rangle = |x_1x_2\rangle \rightarrow |00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{1.8}$$

e formano una base ortonormale dello spazio vettoriale (complesso e di dimensione 4) di tutti gli stati di 2 qubit.

In generale possiamo considerare uno stato di n qubit: questo sistema sarà descritto da una configurazione di $|x_1x_2...x_n\rangle$ possibili stati, dove $\forall k\ x_k \in \{|0\rangle, |1\rangle\}$, ed ogni stato avrà un suo coeffiente complesso. Pertanto tale sistema è descritto da 2^n numeri complessi, come descritto dall'equazione (1.9).

$$|\psi\rangle = \sum_{x\in\{0,1\}^2} \alpha_x |x\rangle \quad \text{con} \quad \sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1 \tag{1.9}$$

Questo rende la simulazione di un sistema quantistico su un computer classico molto difficile, in quanto la quantità di memoria necessaria cresce esponenzialmente e ci rende impossibile simulare sistemi di grandi dimensioni: è qui che entra in gioco un punto fondamentale di forza dei **computer quantistici**, che possono effettivamente **simulare questi sistemi in modo effeciente**, con la necessità di n qubit piuttosto che 2^n numeri complessi.

1.1.3 Stati entangled

Un concetto fondamentale della computazione quantistica è quello di **entanglement**, una proprietà per cui lo stato complessivo di due o più sistemi quantistici risulta non fattorizzabile in un prodotto tensoriale degli stati dei singoli sistemi. Un esempio di stato entangled è il seguente:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{1.10}$$

Questo stato è chiamato *stato di Bell*, in particolare è conosciuto come $|\Phi^+\rangle$. Se misuriamo il primo qubit lungo la base computazionale $\{|0\rangle, |1\rangle\}$, il risultato ottenuto sarà lo stesso che otterremo poi misurando il secondo nella stessa base computazionale: si dice quindi che i due risultati siano *perfettamente correlati*. Questo implica che, conoscendo il risultato della prima misura, possiamo prevedere con certezza il risultato della seconda. Il fenomeno descritto non ha analogo classico e sta alla base di molte applicazion della computazione quantistica. L'entanglement dà luogo a correlazioni non locali che sfidano l'intuizione classica*disuguaglianze di Bell*[9].

Gli altri stati di Bell sono i seguenti:

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle \tag{1.11}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle \tag{1.12}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle \tag{1.13}$$

Possono poi anche esistere stati entangled a più qubit, come ad esempio lo stato di *GHZ*:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle \tag{1.14}$$

1.1.4 Quantum Gate

I qubit vengono manipolati all'interno di un **circuiti quantistici**, i quali sono composti da una serie di **quantum gate** che permettono di eseguire operazioni sui qubit. Queste operazioni sono *reversibili*, ovvero possono essere

eseguite sia in avanti che all'indietro, e sono la controparte quantistica delle operazioni logiche dei circuiti classici.

Da un punto di vista matematico, le operazioni sui qubit sono rappresentate da **matrici unitarie** - in modo da mantenere la normalizzazione dello stato - che agiscono sullo stato del qubit. Ad esempio, la porta di Hadamard, che è una delle porte più comuni, è rappresentata dalla seguente matrice:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.15}$$

Questa matrice agisce sul qubit come segue:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle\,, \quad H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle \tag{1.16}$$

Notiamo che essa ci trasforma il qubit in uno stato di massima sovrapposizione (Vedere esempio (1.1.1)).

Altri quantum gate fondamentali sono le porte di Pauli X, Y e Z le quali sono descritte dalle seguenti matrici:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.17}$$

Notiamo che per esempio la porta X è una porta che effettua una negazione logica sul qubit, ovvero trasforma $|0\rangle$ in $|1\rangle$ e viceversa. Esse in generale eseguono una rotazione sulla sfera di Bloch di π attorno all'asse corrispondente all'operazione che effettuano.

Esistono poi quantum gate che agiscono su più di un qubit, come ad esempio la porta di CNOT (anche conosciuta come *controlled Pauli X Gate*), che è una porta a due qubit che agisce come segue:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \Longrightarrow \quad \text{CNOT}\,|x\rangle\,|y\rangle = |x\rangle\,|x \oplus y\rangle \tag{1.18}$$

dove $\oplus$ indica l'operazione di XOR tra i due qubit. Essa quindi applica la negazione logica (la Pauli X Gate) al secondo qubit se e solo se il primo qubit è $|1\rangle$.

Infine, una categoria di gate molto utili, e che ci serviranno successivamente per capire come è costrutito il circuito quantistico dell'algoritmo **VQE** sono i **quantum gate di rotazione**, che sono porte quantistiche che permettono di ruotare lo stato del qubit attorno ad un asse specifico. Queste porte quantistiche sono rappresentate da matrici parametrizzate da un angolo θ, e sono $R_x(\theta)$, $R_y(\theta)$ e $R_z(\theta)$, che eseguono una rotazione rispettivamente attorno agli assi x, y e z di θ della sfera di Bloch. Combinando queste rotazioni opportunamente possiamo ottenere qualsiasi rotazione per il qubit.

Esse sono definite nel seguente modo:

$$R_x(\theta) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{1.19}$$

$$R_y(\theta) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{1.20}$$

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \tag{1.21}$$

Ci limiteremo a considerare quantum gates ad uno o due qubit. Questo deriva dal fatto si può costruire un set composto da questi gate che è universale[8], ossia si può creare qualsiasi altra quantum gate a partire da esso, e, inoltre, la maggiorparte dell'hardware quantistico supporta unicamente questo tipo di gate[10].

1.1.5 Circuiti quantistici

Un circuito quantistico è un modello per la computazione quantistica, simile ai circuiti classici, che rappresenta l'evoluzione di un sistema quantistico attraverso l'**inizializzazione di qubit** a determinati valori, un insieme finito di **quantum gate** ed eventuali **operatori di misura**.

I circuiti sono scritti in modo che l'asse orizzontale sia il *tempo*, iniziando a sinistra ed andando avanti verso destra. Le linee orizzontali sono i **qubit**, le linee orizzontali doppie sono i **bit classici** (usati per le misure), ed i gate sono

rappresentati come **blocchi** che agiscono sui qubit, a volte con delle linee che collegano i qubit per indicare che il gate agisce su più qubit.

A differenza dei circuiti classici, i circuiti quantistici non permettono le operazioni di **FANOUT** o **FANIN**, ovvero rispettivamente la separazione di un cavo in più cavi, non permessa per il *no cloning theorem*, oppure l'unione di più cavi in uno solo, non permessa in quanto è un operazione *non reversibile*, e sono **aciclici**, ossia non prevedono la connessione di una parte del circuito ad un altra in una maniera tale che si crei un *feedback loop*.

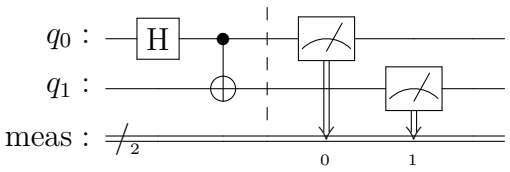

Figura 1.2: Circuito quantistico di esempio

Come esempio, consideriamo il circuito di di Fig. (1.2). Vediamo che il circuito ha 3 "linee", due di queste si trovano in corrispondeza dei qubit q_0 e q_1, mentre la terza (una linea doppia) è la linea di misura, ovvero una linea "classica" dove i risultati delle misure verranno inseriti. Vediamo che quest'ultima è, all'inizio, sbarrata, con affianco il numero 2: ciò indica che in realtà si tratta di due linee di misura, una per ciascun qubit, ma che per essere più sintetici nel disegno del circuito vengono unite in una sola.

All'interno del circuito abbiamo poi due **quantum gates** che notiamo essere rappresentati in modo diverso:

- **Hadamard gate**: È rappresentato da una H all'interno di un blocco. Questo è un gate che agisce su un singolo qubit, in questo caso q_0.
- **CNOT gate**: È rappresentato da un $\oplus$, il quale è collegato ad un altro qubit dove vi è un punto. Esso agisce su due qubit, in questo caso in quanto il qubit q_0 è il **controllo** e il qubit q_1 è il **target**: se q_0 è $|1\rangle$, allora il secondo qubit diventerà $X(q_1)$ (ovvero si applica la porta di Pauli X a q_1, quindi si invertono α e β), in caso contrario rimarrà uguale.

Infine, dopo una linea tratteggiata, abbiamo due **misure** (rappresentate dai blocchi con all'interno una lancetta inclinata), le quali indicano che i qubit q_0 e q_1 vengono misurati. Genericamente, assumiamo la misura venga effettuata nella base computazionale $\{|0\rangle, |1\rangle\}$. Ottenuta la misura, essa viene trasferita ai due bit classici che fanno parte dell'ultima linea doppia, pertanto *classica*, che si trova in basso.

Vediamo quindi brevemente la rappresentazione grafica delle porte descritte in precedenza. Le porte quantistiche di Pauli X, Y e Z in un circuito quantistico vengono mostrate in Fig. (1.3).

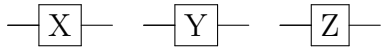

Figura 1.3: Porte di Pauli X, Y e Z

Le quantum gates di rotazione $R_x(\theta)$, $R_y(\theta)$ e $R_z(\theta)$ vengono invece rappresentate come illustrato in Fig. (1.4).

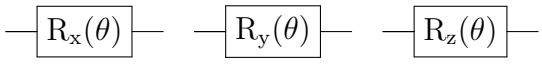

Figura 1.4: Porte R_x, R_y e R_z

Esempio 1.1.2

Nel circuito della Fig. (1.2) ciò che viene fatto è creare lo stato $|\psi\rangle$ seguente:

$$|\psi\rangle = CNOT\left(H|q_0\rangle \otimes |q_1\rangle\right) \tag{1.22}$$

Nel caso q_0 e q_1 fossero rispettivamente inizializzati come $|0\rangle$ e $|0\rangle$, allora lo stato finale, prima delle operazioni di misura, sarà:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{1.23}$$

Si tratta quindi di uno degli stati di Bell visto nell'equazione (1.10), in particolare lo stato $|\Phi^+\rangle$. Per mostrare questo tipo di inizializzazione possiamo rappresentare il circuito quantistico nel seguente modo:

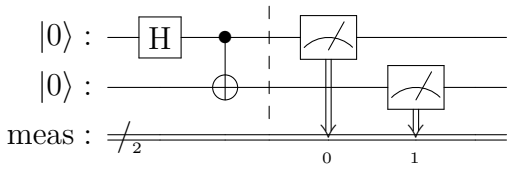

Figura 1.5: Circuito quantistico per la creazione dello stato $|\Phi^+\rangle$

Diverse inizializzazioni ci darebbero i diversi stati di Bell: ad esempio, inizializzando q_0 come $|1\rangle$ e q_1 come $|0\rangle$ otterremmo lo stato $|\Phi^-\rangle$, inizializzando q_0 come $|0\rangle$ e q_1 come $|1\rangle$ otterremmo lo stato $|\Psi^+\rangle$, ed infine inizializzando q_0 come $|1\rangle$ e q_1 come $|1\rangle$ otterremmo lo stato $|\Psi^-\rangle$.

1.2 Introduzione alla meccanica quantistica

Per formalizzare quanto introdotto nel capitolo sulla computazione quantistica - qubit, sovrapposizione, entanglement, quantum gate e quantum circuit - esponiamo i postulati della meccanica quantistica e chiariamo come sono legati a queste tecnologie.

1.2.1 Postulati

Un sistema fisico è in generale descritto attraverso tre sue parti fondamentali: gli stati, gli osservabili e come questo sistema si evolve nel tempo. Il *framework* della meccanica quantistica si basa su tre postulati che ci descrivono proprio queste caratteristiche, e la misura, di un sistema quantistico. Diversi autori organizzano questi aspetti in maniera diversa, ma per i nostri scopi questa formulazione è sufficiente.

Stati quantistici

> **Postulato I** *Ad ogni sistema fisico isolato è associato uno spazio di Hilbert $\mathcal{H}$. Uno stato del sistema fisico è descritto da una direzione in $\mathcal{H}$ (ovvero un vettore con una costante moltiplicativa arbitraria).*

Uno spazio di *Hilbert* $\mathcal{H}$ è uno spazio vettoriale complesso con un prodotto interno. Per rappresentarene uno stato viene usato per convenzione un vettore $|\psi\rangle$ normalizzato, ovvero per cui $\langle\psi|\psi\rangle = 1$, dove $\langle\psi|\phi\rangle$ indica il prodotto scalare tra i vettori $|\psi\rangle$ e $|\phi\rangle$. Insieme al primo postulato viene anche delineato come si descrive un sistema composto:

> **Postulato I.a: Stati Composti** *Lo spazio di Hilbert di un sistema composto è il prodotto tensorale degli spazi di Hilbert associati ai singoli sistemi che lo compongono.*

Il primo postulato pertanto completa la nostra descrizione di un **qubit**: esso è un sistema quantistico che vive in uno spazio di Hilbert bidimensionale. Un sistema a più qubit è descritto da uno spazio di Hilbert che è il prodotto tensiorale degli spazi di Hilbert dei singoli qubit e vive in uno spazio di Hilbert di dimensione 2^n.

Evoluzione temporale

> **Postulato II** *L'evoluzione temporale di uno stato $|\psi(t)\rangle$ di un sistema isolato è descritto dall'equazione di Schrödinger:*
>
> $$i\hbar\frac{d\,|\psi(t)\rangle}{dt} = H(t)\,|\psi(t)\rangle \tag{1.24}$$
>
> *dove $\hbar$ è la costante di Plank divisa 2π ed $H(t)$ è un operatore Hermitiano ed chiamata l'Hamiltoniana del sistema*

La funzione d'onda (o lo stato) evolve **unitariamente** se il sistema è isolato. Se l'Hamiltoniana H non dipende esplicitamente dal tempo, la soluzione di questa equazione è:

$$|\psi(t)\rangle = e^{-iHt/\hbar}\,|\psi(0)\rangle = U(t)\,|\psi(0)\rangle \tag{1.25}$$

Da ciò si nota che questo corrisponde con le **quantum gates**[11], i quali sono operatori unitari che agiscono sullo stato del qubit. Essi sono implementati fisicamente dall'evoluzione dovuta all'Hamiltoniana che governa il sistema fisico per un certo intervallo di tempo.

Abbiamo inoltre un'altra equazione utile che possiamo ottenere da questo postulato, e che poi ci servirà nel contesto del **VQE**, ovvero l'*equazione di Schrödinger indipendente dal tempo*:

$$H\left|\psi\right\rangle = E\left|\psi\right\rangle \tag{1.26}$$

Quando non si è interessati al moto temporale, ma piuttosto ai **livelli energetici** di un sistema, si considerano gli **autostati** di H. Gli autovalori E corrispondono con le possibili energie misurabili. Questa relazione è essenziale per molti algoritmi quantistici: la ricerca dello stato fondamentale - cioè l'autostato con energia minima - è alla base del **VQE**.

Osservabili

Gli osservabili sono descritti da operatori Hermitiani su $\mathcal{H}$. In quanto questi operatori sono Hermitiani, i loro autovalori sono sempre reali, coerentemente con il fatto che i risultati di misura devono essere numeri reali. Se lo spettro dell'osservabile è discreto, allora i possibili risultati della misura sono *quantizzati*.

> **Postulato III** *La misura di un osservabile è descritta da una collezione $\{M_m\}$ di operatori di misura. Questi sono gli operatori che agiscono sullo spazio di Hilbert del sistema che sta venendo misurato. L'indice m si riferisce al possibile risultato della misura. Se lo stato del sistema è $\left|\psi\right\rangle$, la probabilità di ottenere il risultato m è data da:*
>
> $$p(m) = \left\langle\psi\right| M_m^\dagger M_m \left|\psi\right\rangle \tag{1.27}$$
>
> *Dopo la misura, lo stato del sistema è:*
>
> $$\frac{M_m\left|\psi\right\rangle}{\sqrt{\left\langle\psi\right| M_m^\dagger M_m \left|\psi\right\rangle}} \tag{1.28}$$

Gli operatori di misura soddisfano la relazione di completezza:

$$\sum_m M_m^\dagger M_m = I \tag{1.29}$$

Con questo postulato possiamo capire meglio da dove deriva l'equazione (1.7) ed in generale il concetto di misura introdotto nei circuiti quantistici. Non si tratta dell'unico modo di descrivere una misura quantistica, ma è il più comune e il più semplice.

Esempio 1.2.1

Un esempio semplice ma molto importante per capire meglio questo postulato è *la misura di un qubit* nella base computazionale $\{|0\rangle, |1\rangle\}$. Si tratta di una misura con due possibili risultati definiti dai due operatori di misura $M_0 = |0\rangle\langle 0|$ e $M_1 = |1\rangle\langle 1|$. Supponiamo lo stato che venga misurato è $|\psi\rangle = a|0\rangle + b|1\rangle$. Allora la probabilità di ottenere il risultato 0 è:

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = |a|^2$$

In modo analogo, la possibilità di ottenere 1 è $p(1) = |b|^2$. Lo stato dopo la misura diventa di 0 è:

$$\frac{M_0|\psi\rangle}{\sqrt{\langle\psi| M_0^\dagger M_0 |\psi\rangle}} = \frac{a|0\rangle}{|a|} = |0\rangle$$

Sempre in modo analogo, anche il risultato della misura di 1 è $|1\rangle$.

1.2.2 Valore atteso

Un'altro concetto importante da trattare all'interno di questo capitolo è il **valore atteso** (o *di aspettazione*) di un osservabile. Nel postulato della misura abbiamo visto come gli osservabili siano rappresentati da operatori Hermitiani, i cui autovalori corrispondono ai possibili risultati di una misura fisica. Spesso però siamo anche interessati a sapere il valore medio che può assumere un osservabile.

Dato uno stato quantistico $|\psi\rangle$ appartenente ad uno spazio di Hilbert $\mathcal{H}$ e un osservabile quantistico O rappresentato da un operatore Hermitiano, definiamo

il valore medio (o valore di aspettazione) di tale osservabile nello stato $|\psi\rangle$ come:

$$\langle O\rangle_\psi = \langle\psi|O|\psi\rangle \tag{1.30}$$

L'interpretazione fisica di questa quantità è molto importante: rappresenta il risultato medio che si ottiene effettuando ripetutamente la misura dell'osservabile O sullo stato $|\psi\rangle$, in condizioni identiche. Matematicamente, supponendo che l'operatore O abbia uno spettro discreto e una decomposizione spettrale del tipo:

$$O = \sum_i o_i \, |o_i\rangle \, \langle o_i| \tag{1.31}$$

dove $|o_i\rangle$ sono gli autostati (autovettori) e o_i i relativi autovalori (possibili risultati di misura), il valore medio è dato da:

$$\langle O\rangle = \langle\psi| \, O \, |\psi\rangle = \sum_i o_i |\, \langle o_i|\psi\rangle \,|^2 \tag{1.32}$$

dove $|o_i\rangle$ è l'autostato corrispondente all'autovalore o_i.

Esempio 1.2.2

Se consideriamo un qubit nello stato $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ e l'osservabile Z (la matrice di Pauli Z). Il valore atteso di Z nello stato $|\psi\rangle$ è:

$$\langle Z\rangle = \langle\psi|Z|\psi\rangle = |\alpha|^2 - |\beta|^2 \tag{1.33}$$

1.3 Panoramica delle tecnologie attuali

In maniera graduale ma costante la computazione quantistica ha compiuto notevoli progressi, evolvendo dalla teoria alla realizzazione di dispositivi concreti. Sebbene vi siano ancora sfide significative prima che i computer quantistici diventino scalabili e facilmente accessibili, le tecnologie odierne hanno già dimostrato risultati promettenti, alimentando nuove opportunità di sperimentazione e ricerca. Recenti risultati, come quelli raggiunti da Google con

tecniche avanzate di correzione d'errore quantistico[12] e da IBM con i propri processori a più di mille qubit[13], ed ancora da Microsoft con l'introduzione di Majorana 1[14], dimostrano che ci troviamo in una fase importante dello sviluppo di questo ambito.

1.3.1 NISQ Era

L'attuale generazione di computer quantistici appartiene alla cosiddetta **NISQ era** (*Noisy Intermediate-Scale Quantum*), termine introdotto da John Preskill nel 2018[15]. Questi dispositivi hanno un numero limitato di qubit e sono soggetti a errori dovuti a rumore e decorenza. Il rumore può modificare lo stato del sistema in modo imprevedibile, mentre la decoerenza avviene nell'interazione del sistema con l'ambiente circostante, rimuovendone le caratteristiche quantistiche. Seppur le limitazioni siano tante, i computer quantistici NISQ sono già in grado di eseguire algoritmi interessanti ed hanno dimostrato, anche se ancora dibattuta come pietra migliare, l'esecuzione di algoritmi impossibili in tempi umani per computer classici[16].

Proprio grazie e per questi dispositivi sono stati sviluppati algoritmi come il **VQE** (*Variational Quantum Eigensolver*) ed il **QAOA** (*Quantum Approximate Optimization Algorithm*), che sfruttano le capacità limitate dei computer quantistici attuali, ovvero cercano di usare al meglio il numero limitato di qubit e di gate, e li combinano con tecniche classiche per risolvere problemi di ottimizzazione e simulazione quantistica.

Uscire dalla NISQ era vorrebbe dire avere computer quantistici con decine di migliaia di qubit e una correzione degli errori sufficiente a garantire la fedeltà dei risultati. Questi dispositivi potrebbero, per esempio, venire utilizzati per implementare *l'algoritmo di Shor*[8] per numeri molto grandi e rendere obsoleta la crittografia RSA, modificando completamente e permanentemente il mondo della sicurezza informatica.

1.3.2 Piattaforme attuali

Uno dei fornitori più noti per l'utilizzo di computer quantistici è IBM, che offre un servizio cloud per l'accesso a computer quantistici reali. Il servizio, chiamato *IBM Quantum*, permette di programmare e simulare circuiti

quantistici e di eseguire algoritmi scritti su macchine reali, che sono gestite direttamente da IBM. Diverse aziende e istituzioni stanno sviluppando computer quantistici e tecnologie quantistiche. Google, Microsoft, Rigetti Computing, IonQ e D-Wave sono alcuni dei nomi più noti in questo settore.

I **limiti** che esistono per i circuiti quantistici dipende molto dalla specifica implementazione che si considera. Il numero di qubit logici che si è riusciti ad implementare, attraverso l'error correction, è di poche decine, per ora il record raggiunto è di 28[17], ma nelle macchine di IBM, per esempio, abbiamo a disposizione fino a 12 qubit logici[18]. Riguardo al numero di *porte quantistiche* si va invece a parlare di *probabilità* di ottenere un risultato corretto: l'uso di una porta quantistica oppure il passare del tempo durante l'esecuzione di un circuito aumentano la probabilità di errore. Se per esempio consideriamo un circuito con 100 operazioni, che non sono necessariamente i gate inseriti nel circuito ma anche operazioni intermedie usate dall'hardware, allora l'accuratezza che possiamo ottenere è sotto al 65%[19].

Capitolo 2

Problemi di ottimizzazione

In questo capitolo vengono presentate le definizioni formali dei due problemi di ottimizzazione che trattiamo: il **Problema del Commesso Viaggiatore** (*Traveling Salesman Problem* o *TSP*) ed il **Problema dello Zaino** (*Knapsack Problem* o *KP*).

Il capitolo introduce prima le caratteristiche dei problemi di ottimizzazione in generale[20], poi andando nello specifico per ognuno dei problemi. Descriveremo i metodi per ottenere le soluzioni sia attraverso algoritmi esatti che algoritmi euristici, spiegando perché sono necessari[21]. Infine, mostriamo i casi d'uso reali di questi algoritmi, per rendere più tangibile il loro impatto ed il motivo per cui li abbiamo scelti.

2.1 Formulazione dei problemi di ottimizzazione

Un problema di ottimizzazione consiste nel **trovare la migliore soluzione** tra un insieme di tutte le possibili soluzioni. Si traduce nella ricerca di un vettore $\mathbf{x} \in \mathbb{R}^n$ che ottimizza (cioè massimizza o minimizza) una funzione obiettivo $f : \mathcal{R}^n \to \mathbb{R}$, soggetta ad un insieme di **vincoli**. Il problema nella sua forma più generale si esprime come:

$$\begin{aligned} &\text{Minimizza } f(\mathbf{x}) \\ &\quad \text{tale che } \mathbf{x} \in \mathcal{D}, \\ &\qquad g_i(\mathbf{x}) \leq 0 \quad \forall i \in I, \end{aligned} \tag{2.1}$$

dove abbiamo:

- **Variabili decisionali**: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$: rappresentano i parametri da determinare e costituiscono il punto nello spazio decisionale. Ognuna di queste variabili possono essere di due tipi: continue (reali) o intere.

  Nei problemi che andiamo a considerare noi esse saranno intere, ovvero appartenenti a $\mathbb{Z}$. Questo tipo di problemi sono detti **problemi di ottimizzazione intera** e ne fanno parte le versioni che noi prenderemo in considerazione del TSP e del KP.

- **Funzione obiettivo**: $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ definisce il criterio da ottimizzare, assegnando ad ogni soluzione $x \in \mathbb{R}^n$ un valore. Da notare che noi indichiamo che è essa è da "minimizzare", ma se il problema fosse di massimizzazione, sarebbe sufficiente moltiplicare per -1 la funzione obiettivo.

- **Vincoli**: $g_i(\mathbf{x}) \leq 0$ per ogni $i \in I$: stabiliscono le condizioni che le soluzioni devono necessariamente soddisfare, restringendo l'insieme delle soluzioni ammissibili a $\mathcal{D} \subseteq \mathbb{R}^n$.

  Da notare che se vogliamo esprimere un vincolo di uguaglianza $h_i(\mathbf{x}) = 0$, possiamo sempre riscriverlo come due vincoli di disuguaglianza $h_i(\mathbf{x}) \leq 0$ e $-h_i(\mathbf{x}) \leq 0$.

2.2 Il Problema del Commesso Viaggiatore

Il **Problema del Commesso Viaggiatore** (*Traveling Salesman Problem* o *TSP*) consiste nel trovare il percorso di costo minimo che permetta a un commesso viaggiatore di visitare un insieme di città esattamente una volta ciascuna, tornando infine al punto di partenza, dove ogni coppia di città è collegata da un arco con un costo associato.

Modello matematico

Si consideri un grafo completo $G = (V, E)$, dove:

- $V = \{1, 2, \ldots, n\}$ è l'insieme delle n città
- $E = \{(i, j) \mid i, j \in V,\, i \neq j\}$ è l'insieme delle strade che collegano le città

- ad ogni arco (i, j) è associato un costo (o distanza) $d_{ij} \geq 0$. Nel nostro caso useremo istanze del TSP che sono simmetriche, pertanto $d_{ij} = d_{ji}$ per ogni coppia di città i, j.

Detta x_{ij} la generica variabile binaria tale che:

$$x_{ij} = \begin{cases} 1 & \text{se l'arco } (i,j) \text{ è attraversato,} \\ 0 & \text{altrimenti,} \end{cases} \tag{2.2}$$

Allora si vuole cercare di minimizzare la seguente funzione obiettivo:

$$\min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} d_{ij}\, x_{ij} \tag{2.3}$$

Sottoposta ai seguenti vincoli:

$$(1) \quad \sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall i \in V, \tag{2.4}$$

$$(2) \quad \sum_{i=1, i\neq j}^{n} x_{ij} = 1 \quad \forall j \in V, \tag{2.5}$$

dove il vincolo (2.4) e (2.5) assicurano che ogni città sia visitata esattamente una volta, ovvero che da ognuna esca ed entri un solo arco. Infine, dobbiamo anche aggiungere il vincolo di eliminazione dei sottocircuiti. Questo vincolo ha due formulazioni ben note, che sono la MTZ (*Miller-Tucker-Zemlin*) e la DFJ (*Dantzig-Fulkerson-Johnson*). Nel nostro caso specifico però questi vincoli non verranno implementati in quanto aumentano eccessivamente la complessità della formulazione del problema[22] - rendendolo impossibile da simulare. Da un punto di vista pratico questo coincide con un *metodo lazy* (pigro), dove abbiamo rilassato i vincoli del problema iniziale e cerchiamo di ottenere una soluzione accettabile, ed il ciò viene risolto in situazioni reali - ovvero dove i risultati di questi problemi devono venire utilizzati a determinati scopi pratici - aggiungendo vincoli appropriati alla soluzione solo quando la soluzione trovata non è accettabile.

2.3 Il Problema dello Zaino

Il **Problema dello Zaino** (*Knapsack Problem* o *KP*) è un problema di ottimizzazione che riguarda la selezione da un insieme di oggetti, ciascuno caratterizzato da un **valore** e da un **peso**, di un loro sottoinsieme al fine di massimizzare il loro valore totale senza superare una capacità massima prestabilita dello zaino.

Modello matematico (0-1 Knapsack Problem)

Si consideri un insieme di n oggetti, in cui ciascun oggetto i (con $i = 1, 2, \ldots, n$) è caratterizzato da un valore $v_i > 0$ ed un peso $w_i > 0$. Si dispone inoltre di uno zaino con capacità massima W.

L'obiettivo è selezionare un sottoinsieme di oggetti, rappresentato tramite le variabili decisionali:

$$x_i = \begin{cases} 1 & \text{se l'oggetto } i \text{ viene selezionato,} \\ 0 & \text{altrimenti,} \end{cases} \tag{2.6}$$

in modo da massimizzare il valore totale:

$$\max \sum_{i=1}^{n} v_i \, x_i, \tag{2.7}$$

soggetto al vincolo di capacità:

$$\sum_{i=1}^{n} w_i \, x_i \leq W, \tag{2.8}$$

con $x_i \in \{0, 1\}$ per ogni $i = 1, 2, \ldots, n$.

Questa formulazione è tipica del **0-1 Knapsack Problem**, in cui ogni oggetto può essere preso al massimo una volta. Esistono poi varianti in cui gli oggetti possono essere frazionabili o in cui si può selezionare lo stesso oggetto più volte. Ognuna ha caratteristiche e complessità computazionali diverse, ma il nostro interesse si focalizza sulla versione 0-1, che è la più comune e la più studiata.

2.4 Soluzioni dei problemi di ottimizzazione

I problemi **TSP** e **KP** sono facilmente dimostrabili come appartenenti alla classi di complessità **NP-hard** (per il TSP simmetrico si parla di NP-hard[23], per il KP 0-1 di NP-completo[24]. Ciò implica che, in generale, non si conoscono algoritmi che funzionano in tempo polinomiale e che garantiscano di trovare la soluzione migliore. Di conseguenza, a fianco di metodologie **esatte** - che esplorano lo spazio delle soluzioni in maniera completa, con complessità esponenziale o *pseudopolinomiale* nel caso peggiore - sono largamente utilizzate tecniche **euristiche** e **metaeuristiche** - che non garantiscono l'ottimalità, ma offrono soluzioni di buona qualità in tempi computazionalmente più contenuti.

Nelle tabelle riportate nel seguito, la complessità si riferisce a quella nel **caso peggiore** e si esprime in notazione *Big-O*. Le variabili usate per indicare la complessità sono n - il numero di città nel TSP oppure il numero di oggetti nel KP - e W - la capacità dello zaino.

2.4.1 Algoritmi Esatti

Il più intuitivo degli algoritmi che trovano la soluzione esatta di questi problemi è quello che esplora *tutte le possibili soluzioni* e restituisce quella di costo minimo - ovvero utilizza un approccio *brute force*. Ma la complessità computazionale è almeno esponenziale, in particolare abbiamo $O(n!)$ per il TSP e $O(2^n)$ per il KP.

Per mitigare questa esplosione combinatoria, sono stati sviluppati approcci più efficienti: la **programmazione dinamica** ed il **branch and bound**. Il primo parte dalla soluzione di sottoproblemi e memorizza i risultati intermedi, riducendo la complessità computazionale a $O(n^2 2^n)$ per il TSP e $O(nW)$ - dove n è il numero di oggetti e W è la capacità dello zaino - per il KP. Da notare che W indica il numero di *bit* con cui viene rappresentata la dimensione dello zaino, pertanto la dimensione scala in modo esponenziale al variare di W - se per esempio avessimo $W = 4$ bit allora abbiamo una dimensione dello zaino di 2^4.

Il **branch and bound** è un algoritmo di tipo divide-et-impera che costruisce un albero di ricerca in cui vengono esplorati solo i rami promettenti, mentre gli altri vengono potati in base a limiti inferiori sulla soluzione ottima. La sua

complessità computazionale rimane comunque alta nel caso peggiore, ma nei casi favorevoli può ridurre drasticamente lo spazio di ricerca.

Infine, è importante anche citare **Concorde TSP Solver**, considerato essere il milgiore risolutore esatto per TSP, attraverso l'uso del quale è stata risolta la più grande istanza di TSP (85'900 città)[25].

Algoritmo	**Idea Principale**	**Complessità**	**Punti di Forza**	**Limiti**
Branch and Bound	Albero di ricerca con potatura dei rami non promettenti	$\mathcal{O}(n!)$	Riduzione drastica dello spazio di ricerca nei casi favorevoli	Può essere comunque inefficiente se deve esplorare tutte le opzioni
Held-Karp (Dinamico)	Programmazione dinamica su sottoinsiemi di città	$\mathcal{O}(n^2 2^n)$	Struttura ricorsiva chiara, facilita versioni ottimizzate	Complessità elevata, memoria cresce esponenzialmente

Tabella 2.1: Confronto tra algoritmi esatti per il TSP

Algoritmo	**Idea Principale**	**Complessità**	**Punti di Forza**	**Limiti**
Branch and Bound	Albero di ricerca con potatura dei rami non promettenti	$\mathcal{O}(2^n)$	Riduzione dello spazio di ricerca, efficace su istanze piccole	Può essere comunque inefficiente nel caso peggiore
Programmazione Dinamica	Risoluzione tramite sottoproblemi e memorizzazione dei risultati intermedi	$\mathcal{O}(nW)$	Ottimale, sfrutta struttura del problema	Scalabilità limitata per grande W

Tabella 2.2: Confronto tra algoritmi esatti per il Knapsack Problem

Nella Fig. (2.1) possiamo vedere come i tempi di esecuzione degli algoritmi esatti per il KP crescano in modo esponenziale al crescere della dimensione del problema. Questi grafici sono stati generati facendo una media di 8 diverse istanze generate casualmente del KP, ciascuna con oggetti dal valore compreso tra 1 e 10 e pesi compresi tra 1 e 5, per ogni dimensione presa in considerazione, facendo una media dei tempi ottenuti. Abbiamo interrotto l'esecuzione

quando il tempo medio superava 1 secondo, il che ci ha permesso di osservare l'*esplosione* del tempo richiesto nella computazione per i metodi brute force e per la programmazione dinamica, anche se per il breach and bound lo stacco non è così netto.

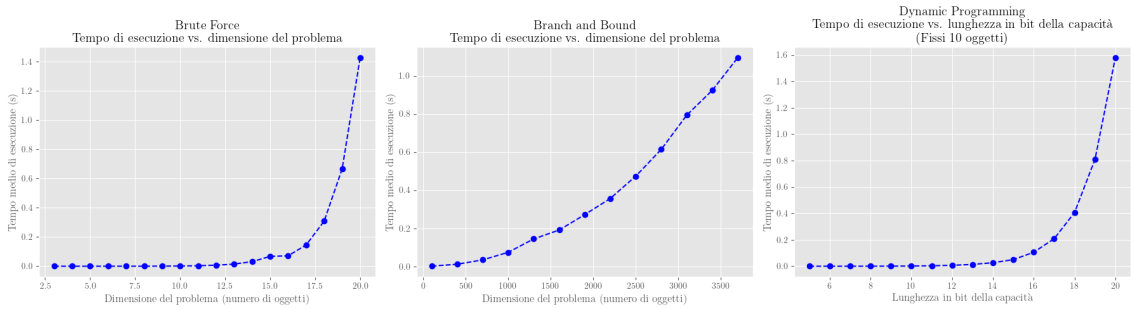

Figura 2.1: Tempi di esecuzione degli algoritmi esatti per il KP su istanze di varie dimensioni.

2.4.2 Algoritmi Euristici

Gli *algoritmi euristici* che trattiamo in generale si possono raggruppare in due macro-categorie: **costruttivi** e **metaeuristici**. I primi costruiscono una soluzione aggiungendo elementi uno alla volta, utilizzando regole oppure un approccio *greedy*, mentre i secondi partono da una soluzione iniziale e cercano di migliorarla iterativamente.

In particolare vorrei evidenziare gli algoritmi euristici stipulati come i migliori per il TSP e il KP, che sono rispettivamente la **Lin-Kernighan Heuristic** ed il **FPTAS**. Questi algoritmi sono importanti perché riescono a raggiungere entrambi gli obiettivi che si hanno con algoritmi che trovano soluzioni approssimate: sono veloci e le soluzioni che ottengono si distanzano poco da quelle ottime (nel caso del FPTAS, la differenza è controllata da un parametro ϵ, mentre per il Lin-Kernighan Heuristic le soluzioni si discostano dal 1 al 2% rispetto alla soluzione ottimale[26]).

Bisogna far notare che *FPTAS* in realtà non è definibile come un algoritmo euristico, ma come un algoritmo approssimato, in quanto garantisce di trovare una soluzione ottima entro un certo margine di errore. Tuttavia, per mantenenre una certa coerenza abbiamo preferito inserirlo in questa categoria.

Algoritmo	Idea Principale	Complessità	Punti di Forza	Limiti
Nearest Neighbor	Costruzione del tour scegliendo il nodo più vicino	$\mathcal{O}(n^2)$	Semplice ed efficiente	Spesso lontano dall'ottimo
Simulated Annealing	Ricerca locale con accettazione probabilistica di soluzioni peggiori	Variabile	Buone soluzioni approssimate con tempo sufficiente	Richiede tuning dei parametri
Ant Colony Optimization	Simulazione del comportamento delle formiche per costruire soluzioni iterative	$\mathcal{O}(n^2)$ per iterazione	Funziona bene su istanze grandi	Lento, necessita molte iterazioni
Lin-Kernighan Heuristic[27]	Euristica di ricerca locale basata su scambi di archi	$\mathcal{O}(n^2)$	Ottime soluzioni su istanze grandie	Complessità implementativa
Stem and Cycle Heuristics	Costruzione di un albero di supporto e conversione in ciclo	Variabile	Migliori risultati rispetto a LKH	Qualora più lenta nell'esecuzione

Tabella 2.3: Confronto tra algoritmi euristici per il TSP

Algoritmo	Idea Principale	Complessità	Punti di Forza	Limiti
Greedy (Valore/Peso)	Selezione degli oggetti con miglior rapporto valore/peso	$\mathcal{O}(n \log n)$	Rapido e semplice	Non sempre ottimale
Greedy (Valore massimo)	Selezione degli oggetti con il valore maggiore per prima	$\mathcal{O}(n \log n)$	Rapido e semplice	Non sempre ottimale
Simulated Annealing	Ricerca locale con accettazione probabilistica di peggioramenti	Variabile	Soluzioni vicine all'ottimo	Richiede tuning dei parametri
FPTAS	Approssimazione tramite scaling e programmazione dinamica	$\mathcal{O}(nW/\epsilon)$	Approccio quasi ottimale	Precisione dipende da ϵ, richiede più memoria

Tabella 2.4: Confronto tra algoritmi euristici per il Knapsack Problem

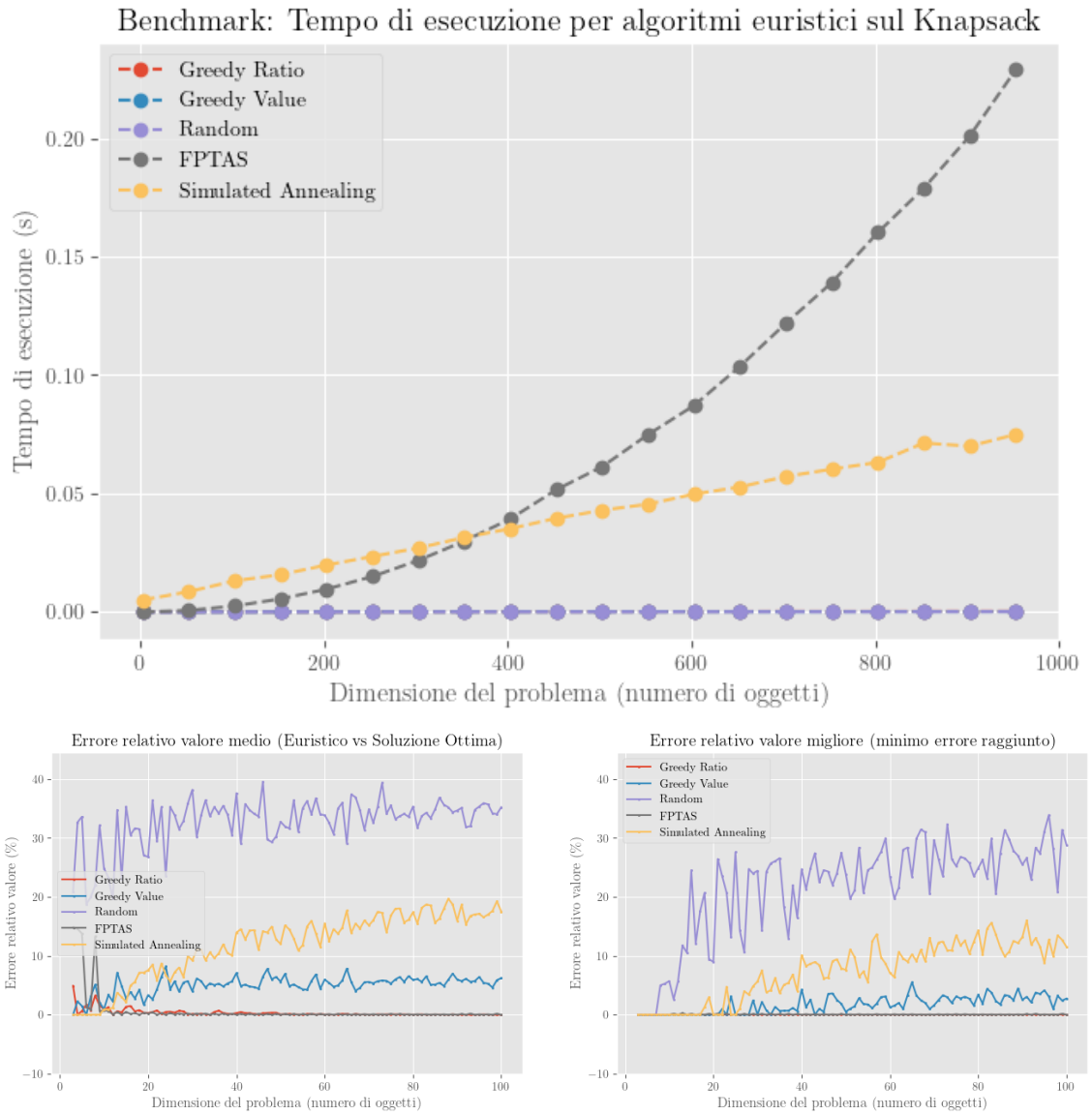

Figura 2.2: Soluzioni ottenute dagli algoritmi euristici per il KP su istanze di varie dimensioni. Nel grafico dei tempi di esecuzione, tutti i metodi fuorché FPTAS e Simulated Annealing si trovano "*sotto*" la linea orizzontale del metodo `Random`.

Nei grafici presenti nella Fig. (2.2) possiamo vedere come le soluzioni ottenute dagli algoritmi euristici per il KP si discostino da quella ottima, ma che comunque siano accettabili. Questi grafici sono stati generati usando la stessa metodologia utilizzata per i metodi esatti, facendo una media di 8 diverse istanze generate casualmente del KP, ciascuna con oggetti dal valore compreso tra 1 e 10 e pesi compresi tra 1 e 5, per ogni dimensione presa in considerazione, facendo una media dei valori ottenuti. Nei grafici sono presenti sia il tempo di esecuzione degli algoritmi al crescere della dimensione del problema, sia l'errore relativo rispetto alla soluzione ottima, calcolata con un metodo

esatto utilizzando la formula (2.9). Nel grafico in basso a destra è presente l'errore relativo rispetto alla soluzione ottima migliore trovato nelle 8 istanze, in modo da capire se, almeno a volte, l'algoritmo riesce davvero a trovare la soluzione ottima. Abbiamo anche incluso le soluzioni ottenute attraverso una *scelta casuale degli oggetti*, dove gli oggetti vengono mischiati e vengono selezionati uno ad uno fintanto che il loro peso è inferiore alla capacità dello zaino.

Analizzando i grafici, i tempi delle esecuzioni sono compatibili con quanto previsto: estremamente rapidi per gli algoritmi greedy, più lenti per il simulated annealing ed FPTAS, con quest'ultimo che cresce in maniera più rapida degli altri metodi, visto il fatto che alla base è composto da un algoritmo esatto eseguito con una precisione ridotta. L'errore relativo rispetto alla soluzione ottima è molto variabile, soprattutto per istanze piccole, ma al crescere della dimensione delle istanze si stabilizza a valori prossimi allo 0% per FPTAS e greedy (valore/peso), mentre per il greedy (valore massimo) si stabilizza attorno al 5%. Le soluzioni casuali rimangono sempre attorno al $30-35\%$ di errore relativo, mentre il simulated annealing ha un errore relativo che cresce con l'aumentare della dimensione del problema, arrivando oltre al 15% per istanze di dimensione $n = 100$.

$$\text{Errore relativo} = \frac{|\text{Soluzione euristica} - \text{Soluzione ottima}|}{|\text{Soluzione ottima}|} \tag{2.9}$$

2.5 Casi d'uso reali

Questi problemi, come anche gli altri problemi di ottimizzazione, sono modelli ideali di situazioni che si possono riscontrare in diversi ambiti.

Trovare *soluzioni migliori* per questi problemi in *tempi ridotti* ci permette di ottimizzare costi e tempi nei contesti reali in cui vengono utilizzate. Inoltre è utile notare che non è realistico trovare una soluzione ottimale o quasi ottimale nel caso essa debba venire calcolata in tempi reali, come non è ragionevole utilizzare una soluzione particolarmente grossolana per problemi critici, dove una soluzione precisa risulta importante. C'è quindi necessità di accertarsi su quali siano i compromessi che si possono fare di caso in caso.

Il Problema del Commesso Viaggiatore (TSP) è ampiamente utilizzato nella logistica, ottimizzando i percorsi di consegna per ridurre costi e consumo di

carburante[28], oppure nella produzione di *microchip*[29]. Il Problema dello Zaino (KP) può essere usato per trovare il modo con meno spreco per il taglio di materiali grezzi[30], oppure ha applicazioni nella finanza, aiutando a costruire portafogli di investimento ottimali[30].

Quando questi problemi vengono considerati in situazioni reali, le formulazioni possono variare leggermente da quelle *standard*. Non avviene sempre, difatti ci sono molti casi d'uso anche per le formulazioni pure del TSP e del KP. Tuttavia volevo spezzare una lancia in ogni caso sul trattarli nella maniera meno specializzata, come nel nostro caso, in quanto questo ci permette di fare delle considerazioni più generali e visionare gli algoritmi in una maniera meno contorta. In quanto la struttura che si trova alla base delle soluzioni di questi problemi o loro varianti rimane piuttosto simile, la nostra ricerca non risulta ridondande a rispondere ai quesiti riguardo all'utilità pratica di questi algoritmi.

Capitolo 3

Variational Quantum Eigensolver

Il *Variational Quantum Eigensolver* (**VQE**) è un algoritmo ibrido, in quanto combina componenti quantistiche e classiche, la cui funzione è quella di trovare lo stato fondamentale di un sistema fisico. Dato un circuito quantistico parametrizzato ed una configurazione iniziale, il computer quantistico calcola i valori di aspettazione del sistema rispetto ad un osservabile, di solito l'*Hamiltoniana* del sistema, ed un ottimizzatore classico si occupa di migliorare i parametri del circuito, minimizzando il valore misurato.

In questo capitolo viene fornita una descrizione dettagliata dell'algoritmo VQE e delle sue componenti, con particolare attenzione alla sua applicazione ai problemi di ottimizzazione combinatoria e come questi vengono tradotti in un problema di minimizzazione del valore di aspettazione di un'Hamiltoniana.

3.1 Origini e contesto

L'algortimo VQE è stato proposto per la prima volta tra il 2013 ed il 2014 da Alberto Peruzzo et al.[31]. Questo algoritmo è stato sviluppato per superare le limitazioni degli algoritmi quantistici precedenti, come il **Quantum Phase Estimation** (*QPE*), che richiedono circuiti quantistici profondi e pertanto sono difficili da implementare su hardware rumoroso e con tempi di coerenza limitati.

Il principale caso d'uso per il VQE è la chimica quantistica, dove viene utilizzato per calcolare l'energia di legame delle molecole[32]. In generale però esso è un algoritmo altamente versatile, poiché una moltitudine di proble-

mi, tra cui anche i problemi di ottimizzazione, possono venire espressi come un'Hamiltoniana.

QAOA

Per dare una visione più completa del panorama degli algoritmi quantistici attuali, è importante includere una piccola sezione sul **Quantum Approximate Optimization Algorithm** (*QAOA*). Si tratta sempre di un **Variational Quantum Algorithm** (*VQA*), come il VQE, ed è stato proposto per la prima volta da Farhi et al. nel 2014[33] per risolvere problemi di ottimizzazione combinatoria. L'idea alla base del QAOA è quella di utilizzare un'ansatz particolare, che si basa sul concetto di *adiabatic quantum computing*, per migliorare le prestazioni nell'approssimare la soluzione ottima di un problema di ottimizzazione.

La nostra scelta di utilizzare il VQE a discapito del QAOA consiste principalmente in un interesse accademico, in quanto vogliamo specificatamente vedere le performance di questo algoritmo e come esso si comporta in problemi di ottimizzazione, nonché per il fatto che la letteratura esistente per il QAOA è più consolidata.

3.2 Definizione

L'algoritmo è costituito da diverse componenti che interagiscono tra loro per ottenere il risultato ottimale. La Fig. (3.1) mostra il funzionamento generale dell'algoritmo.

Esso funziona seguendo questi passaggi ("*Prep.*" indica i passaggi che vengono fatti unicamente all'inizio per *preparare* l'esecuzione dell'algoritmo):

Prep. **Scelta dell'*ansatz***: Viene creata una funzione d'onda parametrizzata che dipende da un insieme di parametri variabili $\vec{\theta} = (\theta_1, ..., \theta_n)$, che prende il nome di *ansatz*. Questo viene implementato attraverso un circuito quantistico il quale, a partire da $|\psi(0)\rangle$, genera lo stato $|\psi(\vec{\theta})\rangle$ attraverso una serie di gate quantistici che noi sommariamente connoteremo come $U(\vec{\theta})$.

Prep. **Scelta dello stato iniziale**: Viene selezionato uno stato iniziale, di solito $|0...0\rangle$ oppure $|+...+\rangle$[34], ma altre scelte sono possibili.

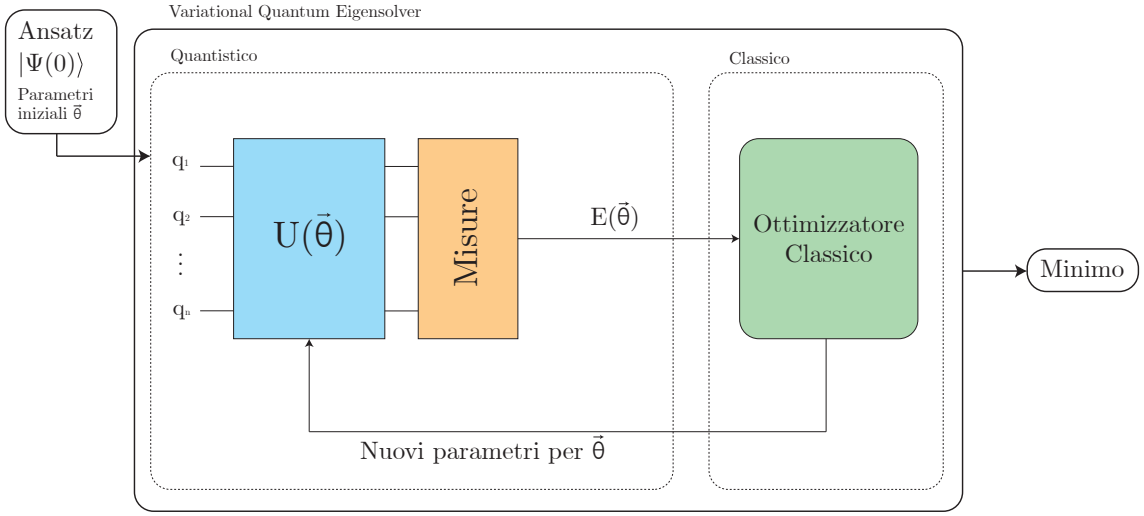

Figura 3.1: Funzionamento del VQE

Prep. **Definizione Hamiltoniana**: Descriviamo il sistema del quale vogliamo trovare lo stato fondamentale attraverso un'Hamiltoniana, ovvero:

$$\hat{H} = \sum_{i=1}^{n} \alpha_i P_i \tag{3.1}$$

dove P_i sono *stringhe di Pauli* - ovvero moltiplicazioni tensoriali di operatori di Pauli - che descrivono il sistema fisico, e α_i sono coefficienti reali. Questa Hamiltoniana può essere rappresentata come una matrice $2^n \times 2^n$, dove n è il numero di qubit del sistema.

1. **Preparazione del sistema quantistico**: Eseguiamo il nostro circuito quantistico utilizzando i parametri $\vec{\theta}$, in modo da trasformare $|\psi(0)\rangle$ a $|\psi(\vec{\theta})\rangle$.

2. **Misura del valore di aspettazione dell'Hamiltoniana**: Si esegue la misura per ottenere il valore $\langle\hat{H}\rangle = \langle\psi(\vec{\theta})|H|\psi(\vec{\theta})\rangle = E(\vec{\theta})$, quindi si ottiene l'energia del sistema fisico per i parametri $\vec{\theta}$.

3. **Ottimizzazione classica**: Un ottimizzatore (*COBYLA*, *SPSA*, ...) eseguito su un computer classico si occuperà di prendere questi risultati e scegliere come modificare i vari parametri $\vec{\theta}$ in modo da minimizzare l'energia del sistema.

4. **Convergenza**: Si ripetono i passaggi 1-3 fino a quando non si raggiunge una certa condizione di convergenza, ad esempio quando la differenza tra due valori di energia successivi è inferiore ad una certa soglia, oppure quando si raggiunge un numero massimo di iterazioni.

A sua volta, ognuno di questi passi è costituito da un numero elevato di considerazioni, con ampia letteratura scientifica che tratta approfonditamente ciascuno di questi ambiti. Nelle sezioni seguenti ci siamo limitati pertanto a descriverli in una maniera non esaustiva, in quanto il nostro obiettivo finale rimane quello di mettere in pratica questi algoritmi e vederne i risultati, anche se nella nostra formulazione usiamo alcune accortezze subottimali rispetto allo stato dell'arte.

3.2.1 Motivazioni del VQE e Principio Variazionale

Per una data Hamiltoniana $\hat{H}$ ed un vettore di stato $|\psi\rangle$ che può assumere qualsiasi valore, abbiamo che $\min_{|\psi\rangle} \langle\psi|\hat{H}|\psi\rangle$ sarà l'energia dello stato fondamentale e $\text{argmin}_{|\psi\rangle} \langle\psi|\hat{H}|\psi\rangle$ sarà lo stato fondamentale[35]. Tuttavia il numero di stati che $|\psi\rangle$ può assumere sono 2^n, pertanto non è possibile esplorarli tutti per sistemi grandi.

Alla base del VQE vi è il **principio variazionale** della meccanica quantistica, il quale afferma che data un'Hamiltoniana $\hat{H}$ e un qualsiasi stato quantistico $|\psi\rangle$ allora l'energia misurata rispetto a tale stato soddisfa l'equazione:

$$E_0 \leq \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \tag{3.2}$$

dove E_0 è l'energia dello stato fondamentale della Hamiltoniana $\hat{H}$. Pertanto qualsiasi valore di aspettazione dell'energia calcolato sarà sempre superiore rispetto all'energia fondamentale.

Dato queste informazioni formali possiamo meglio capire perché il VQE è promettente: con abbastanza gradi di libertà, il nostro stato $|\psi(\vec{\theta})\rangle$ può davvero farci trovare lo stato fondamentale di $\hat{H}$, rimanendo inoltre certi che non possiamo mai scendere sotto il valore di E_0.

3.2.2 Scelta dell'ansatz

Nell'algoritmo VQE, l'**ansatz** è il circuito quantistico parametrizzato che viene utilizzato per approssimare lo stato fondamentale dell'Hamiltoniana del problema considerato. La scelta dell'ansatz è una *componente critica* del VQE, poiché influisce direttamente sulla capacità dell'algoritmo di esplorare lo spazio degli stati e sulla sua efficienza computazionale[36].

Esistono tre categorie principali nelle quali possiamo dividere gli ansatz:

- **Problem-Oriented ansatz**: progettati specificatamente per il problema in esame, utilizzano informazioni note sul sistema fisico per costruire circuiti che tendono ad essere più efficienti in termini di convergenza, in quanto lo spazio delle soluzioni è più ristretto o viene predisposto in modo intelligente. Esempi noti di questo approccio includono l'**UCC** (Unitary Coupled Cluster) per problemi di chimica quantistica[37] o sue varianti[38] oppure l'ansatz del **QAOA**, che fa uso del concetto di *adiabatic quantum computing* ed è particolarmente adatto per problemi di ottimizzazione combinatoria[39][40].

- **Hardware-Efficient ansatz**: progettati per essere facilmente implementabili su dispositivi quantistici attuali, minimizzando il numero di operazioni e tenendo conto delle limitazioni dell'hardware. Solitamente questi ansatz consistono in strati ripetuti di porte parametrizzate - come le porte quantistiche di rotazione di Pauli R_x, R_y e R_z - e operazioni di entanglement - con l'uso di CNOT. Possono, e a volte vengono, essere utilizzate solo uno o due *layer* di porte[41].

- **Adaptive ansatz**: questi ansatz vengono costruiti iterativamente, in base al feedback che viene dato dall'ottimizzatore oppure attraverso *machine learning*[42][43]

Esempio 3.2.1

Nella Fig. (3.2) viene mostrato un semplice ansatz per 3 qubit. In mezzo ai nostri gate di rotazione R_x ed R_z, abbiamo anche delle porte *CNOT*, che ci permettono di creare entanglement tra i qubit. Questo ansatz è un esempio di un'ansatz *hardware-efficient*, in quanto è composto da pochi gate e può essere facilmente implementato su un computer quantistico attuale. In questo esempio, abbiamo 11 parametri $\theta[n]$ da ottimizzare. Sebbene il numero di parametri sia eccessivo per questo semplice circuito, ci permette comunque di visualizzare come è formato un'ansatz.

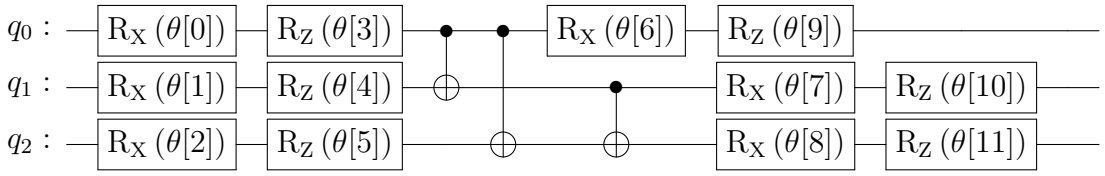

Figura 3.2: Esempio di ansatz per 3 qubit

Problemi di ansatz inopportuni

Scegliere ansatz inopportuni ci porta ad avere diversi problemi nella nostra esecuzione dell'algoritmo. Potremmo avere ansatz che non sono **sufficientemente espressivi** - e che quindi non possono, neanche dopo la minimizzazione, rappresentare lo stato fondamentale del sistema, oppure **eccessivamente grandi** - di cui quindi non è possibile una simulazione accurata, in quanto più suscettibili a rumore e decoerenza[44] [36]. Avere troppi parametri può inoltre portare a **minimi locali** che non sono minimi globali, e quindi a soluzioni subottimali dovute ad un *landscape* delle soluzioni troppo accidentato. Inoltre vi è il problema opposto, conosciuto come il problema del **barren plateaus**, dove regioni del *landscape* delle soluzioni hanno gradienti molto bassi, rendendo l'ottimizzazione molto difficile[36].

3.2.3 Mappare i problemi di ottimizzazione

Per risolvere un problema di ottimizzazione usando il VQE, esso deve venire **mappato** in un'Hamiltoniana. Il nostro scopo è far sì che la soluzione migliore del nostro problema di ottimizzazione corrisponda allo stato fondamentale del sistema. Abbiamo in precedenza (nel capitolo 2) definito le formulazioni dei problemi del TSP e del KP, descrivendoli usando delle **variabili decisionali** da determinare, una **funzione obiettivo** da minimizzare e dei **vincoli** che le soluzioni devono rispettare.

Nel nostro caso trasformeremo i modelli precedentemente descritti in formulazioni **QUBO** (*Quadratic Unconstrained Binary Optimization*), le quali sono facilmente traducibili poi nel modello **Ising**, che può essere direttamente implementato come Hamiltoniana. Questo è un processo piuttosto comune per risolvere problemi di ottimizzazione con computer quantistici, in quanto

l'hardware è predisposto alle operazioni di misura previste e la rappresentazione come Hamiltoniana è molto naturale. Da notare che questo non è l'unico modo per mappare problemi di ottimizzazione in un'Hamiltoniana ed esistono più formulazioni QUBO per lo stesso problema[45] [46].

Knapsack Problem

La formulazione del KP è:

$$\text{massimizza} \quad \sum_{i=1}^{n} v_i x_i \tag{3.3}$$

$$\text{soggetto a} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{3.4}$$

dove abbiamo n oggetti, ciascuno con un valore v_i ed un peso w_i, ed x_i sono le nostre variabili decisionali che valgono 1 se l'oggetto è selezionato e 0 altrimenti. Il KP può viene mappato come segue:

1. **Trasformiamo in QUBO**: La formulazione QUBO in generale è della forma $\min_x x^t Q x + \mu^T x$. Si tratta, come dal nome suggerisce, di minimizzare una funzione di costo quadratica senza vincoli. Per rimuovere i vincoli, essi verranno aggiunti alla funzione obiettivo con dei coefficienti di penalizzazione[47]:

$$\min_x f(x) = -\sum_{i=1}^{n} v_i x_i + \lambda \left(\sum_{i=1}^{n} w_i x_i - W + s \right)^2 \tag{3.5}$$

Qui abbiamo trasformato il nostro problema di massimizzazione in uno di minimizzazione - in quanto noi cerchiamo lo stato con minore energia, e abbiamo aggiunto un termine di penalizzazione per il vincolo del peso, con λ un parametro di penalizzazione e s un parametro di slack - il cui scopo è quella di gestire in maniera più controllata queste penalizzazioni. Il significato di λ è facilmente intuibile, mentre s è un parametro che ci permette di controllare la "distanza" dal vincolo. Se $s = 0$ allora il vincolo è rispettato esattamente, mentre se $s > 0$ allora il vincolo è rispettato con una certa tolleranza. Introduciamo questo termine solo per i vincogli di disuguaglianza, mentre per i vincoli di uguaglianza non è necessario. Esistono anche altri metodi per trasformare i vincoli che non fanno uso di slack variable, come ad esempio l'unbalanced penalization[48].

Otteniamo quindi una riformulazione, o meglio un'approssimazione, del KP, che possiamo ora risolvere con il VQE.

2. **Transformazione in Ising**: possiamo ora trasformare il nostro problema QUBO nella formulazione Ising, la quale è equivalente, dove $x_i \in \{0, 1\}$ diventano variabili di spin $s_i \in \{-1, 1\}$ attraverso $x_i = \frac{1-s_i}{2}$, e quindi la nostra Hamiltoniana sarà:

$$\hat{H} = \sum_{i=1}^{n} v_i \left(\frac{1 - s_i}{2}\right) + \lambda \left(\sum_{i=1}^{n} w_i \left(\frac{1 - s_i}{2}\right) - W + s\right)^2 \tag{3.6}$$

Il risultato finale, sia per il TSP che per il KP, sarà una Hamiltoniana che rappresenta il nostro problema e che ha la forma:

$$\hat{H} = \sum_{i=1}^{n} h_i s_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} J_{ij} s_i s_j \tag{3.7}$$

dove h_i e J_{ij} sono coefficienti reali.

3.2.4 Misura dell'Hamiltoniana

Nella computazione quantistica, l'Hamiltoniana di Ising è scritta in termini di operatori di Pauli Z, dove le variabili classiche diventano operatori di spin Z_i:

$$s_i \to Z_i, \quad s_i s_j \to Z_i Z_j \tag{3.8}$$

Una tipica Hamiltoniana di Ising per n qubit diventa:

$$\hat{H} = \sum_{i=1}^{n} h_i Z_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} J_{ij} Z_i Z_j \tag{3.9}$$

Questo viene fatto in quanto $Z|0\rangle = +1|0\rangle$ e $Z|1\rangle = -1|1\rangle$, quindi gli autovalori rappresentano in modo effettivo i due spin state del modello Ising.

Ora che abbiamo un'Hamiltoniana dobbiamo ottenerne il valore di aspettazione. Da notare che se vi fossero somme di operatori di Pauli diversi, allora dovremmo fare delle misure in basi diverse - tendenzialmente facendo delle rotazioni prima della misura con Z. Nel nostro caso però, in quanto abbiamo solo operatori di Pauli Z, possiamo misurare direttamente in questa base.

Ogni misura - chiamata **shot** - fornisce una sequenza di bit, che rappresenta lo stato del sistema dopo la misura. In quanto le misure sono **probabilistiche**, con poche misure non riusciamo a determinare in modo accurato il valore di aspettazione del nostro sistema. Si andrà quindi a collezionare più misure e calcoliamo il valore di aspettazione come la loro media. Di solito il numero di *shot* è di 1024 o meno, a seconda dall'hardware che abbiamo, il quale si cerca spesso di usare il meno possibile perché ha costi elevati.

Poi computiamo i valori $\langle Z_i \rangle$ e $\langle Z_i Z_j \rangle$ sullo stato variazionale $|\psi(\vec{\theta})\rangle$, e quindi otteniamo il valore di aspettazione dell'Hamiltoniana dopo una serie di misure come mostrato nell'equazione (3.10):

$$E(\vec{\theta}) = \sum_{i=1}^{n} h_i \langle Z_i \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} J_{ij} \langle Z_i Z_j \rangle \tag{3.10}$$

Esempio 3.2.2

Facciamo un esempio di misura di un'Hamiltoniana. Supponiamo di avere un sistema di 3 qubit e di avere l'Hamiltoniana:

$$\hat{H} = 2Z_1 + 0.5Z_2 + 0.3Z_3 + 5Z_1Z_2 + 3.5Z_1Z_3 \tag{3.11}$$

Supponiamo di eseguire uno *shot* ed ottenre la stringa di bit 101. Allora possiamo calcolare i valori di aspettazione per questo risultato come:

$$\langle Z_1 \rangle = -1 \tag{3.12}$$
$$\langle Z_2 \rangle = +1 \tag{3.13}$$
$$\langle Z_3 \rangle = -1 \tag{3.14}$$
$$\langle Z_1 Z_2 \rangle = -1 \tag{3.15}$$
$$\langle Z_1 Z_3 \rangle = +1 \tag{3.16}$$

Il nostro valore di aspettazione calcolato per questa misura sarà quindi:

$$E = 2 \cdot -1 + 0.5 \cdot 1 + 0.3 \cdot -1 + 5 \cdot -1 + 3.5 \cdot 1 = -3.3 \tag{3.17}$$

Ripetendo questa misura otteniamo magari una stringa di bit diversa, come per esempio 010, che ci dà quindi il valore di aspettazione $E = 2 \cdot 1 + 0.5 \cdot -1 + 0.3 \cdot 1 + 5 \cdot -1 + 3.5 \cdot -1 = -6.7$. Ogni stringa di bit che è possibile misurare

avrà una **probabilità associata** di venire misurata, che viene determinata sperimentalmente. Occorre quindi ripetere queste misure per un numero sufficiente di volte, in modo da far diminuire l'errore delle probabilità associate ed ottenere un valore di aspettazione più accurato.

Una volta raccolte abbastanza misure, possiamo calcolare il valore di aspettazione dell'energia usando l'equazione (3.18).

$$E(\vec{\theta}) = \sum_{x \in \{0,1\}^2} P(\tilde{x}) E(\tilde{x}) \tag{3.18}$$

dove $P(x)$ è la probabilità di misurare la stringa di bit $x \in \{0,1\}^n$ e $E(x)$ è il valore di aspettazione dell'energia per la stringa di bit x.

Notiamo che propio per come è costruita questa Hamiltoniana, che quindi usa solo operatori di Pauli Z, ci è sufficiente una singola stringa di bit per ottenere tutte le informazioni necessarie per calcolare tutti i valori di aspettazione dell'Hamiltoniana utilizzata. Non abbiamo bisogno quindi di eseguire misure in basi diverse - quindi fare rotazioni dei qubit - per calcolare i vari valori di aspettazione, ma ci basta fare *sampling* - quindi eseguire una misura del circuito - e, a partire da questo, possiamo calcolare tutti i valori di aspettazione.

Se vi fossero operatori di Pauli diversi, cosa che può accadere quando si usano un'Hamiltoniana che non derivano dalla formulazione Ising, allora una stringa ottenuta da una misura non sarebbe sufficiente a calcolare tutti i valori di aspettazione dell'Hamiltoniana allo stesso momento, pertanto dovremmo fare delle misure in basi diverse per completare il calcolo.

Supponiamo di avere l'Hamiloniana dell'equazione (3.19).

$$\hat{H} = 2X_1Z_2 + 0.5X_1X_2 \tag{3.19}$$

Per misurare X_i dobbiamo applicare una rotazione di Hadamard H prima della misura del qubit i-esimo, in quanto gli autostati di X sono $|+\rangle$ e $|-\rangle$ con rispettivamente gli autovalori $+1$ e -1, e quindi trasformiamo un eventuale $|+\rangle$ in $|0\rangle$ ed un $|-\rangle$ in un $|1\rangle$ e pertanto otteniamo, quando vengono misurati nella base computazionale, gli autovalori di X ma misurati con la tipica base computazionale $\{|0\rangle, |1\rangle\}$ (questo procedimento dell'aggiunta di H viene

tipicamente fatto per limitazioni dell'hardware, il quale spesso può eseguire misure in un'unica base).

Pertanto, per ottenere il valore di aspettazione di X_1Z_2 dobbiamo fare una misura in una base diversa rispetto a quella di X_1X_2. Dovremmo quindi fare delle misure **con** H applicata al qubit 1 e **senza** H applicata al qubit 2 per ottenere il valore di aspettazione di X_1Z_2 e delle misure con H applicata ad **entrambi** i qubit per ottenere il valore di aspettazione di X_1X_2.

Si può quindi notare la comodità di avere Hamiltoniane, come quelle che derivano della formulazione di Ising, che siano composte solo da operatori di Pauli Z, in quanto diminuiscono la complessità delle misure da eseguire per ottenere il valore di aspettazione dell'energia.

Metodi Diversi dalla Misura per Minimizzare l'Energia

La misura di un circuito parametrizzato da ottimizzare non è l'unico modo di trovare gli autovalori minimi di un'Hamiltoniana - e quindi lo stato fondamentale del sistema. Questi potrebbero anche essere ottenuti **classicamente** oppure utilizzando altri algoritmi quantistici come il **Quantum Phase Estimation** (QPE)[8]. Tuttavia, il metodo classico risulta di complessità esponenziale, in quanto la dimensione della Hamiltoniana è di $2^n \times 2^n$ ed un algoritmo per ottenerne l'autovettore minimo è di complessità almeno polinomiale[49], mentre il QPE richiede una profondità di circuito troppo elevata, non rendendolo attualmente utilizzabile nella pratica[50]. Questo ci porta quindi a preferire l'utilizzo del VQE, in quanto risulta essere un metodo più efficiente e scalabile.

3.2.5 Ottimizzazione dei Parametri

Diversi ottimizatori possono essere utilizzati per minimizzare l'energia del sistema[51]. Si dividono principalmente in due categorie: **gradient-based** e **gradient-free**, in base al loro utilizzo o meno del gradiente nel processo di ottimizzazione. In seguito, nelle tabelle (3.2) e (3.1) ne elenchiamo diversi di quelli più utilizzati e le loro caratteristiche principali.

Infine abbiamo anche la possibilità di utilizzare ottimizzatori basati su discesa del gradiente adattivi, ovvero che modificano il tasso di apprendimento

Nome	Funzionamento	Vantaggi	Svantaggi
COBYLA	Costruisce approssimazioni lineari locali, riducendo progressivamente la regione di fiducia	Non necessita di molte valutazioni	Può rimanere bloccato in minimi locali
Nelder-Mead	Costruisce un simplesso che si adatta dinamicamente alla funzione per scendere verso il minimo	Efficace per problemi semplici	Può avere problemi in spazi con più dimensioni
POWELL	Senza vincoli, ottimizza sequenzialmente lungo direzioni aggiornate a ogni iterazione	Buona convergenza	Può essere computazionalmente costoso

Tabella 3.1: Ottimizzatori gradient-free

Nome	Funzionamento	Vantaggi	Svantaggi
SPSA	Stima il gradiente in modo stocastico con poche valutazioni della funzione	Robusto in presenza di rumore	Convergenza meno rapida e tuning di iperparametri
L-BFGS-B	Approssima informazioni di secondo ordine con memoria limitata, usando i gradienti per affinare la ricerca entro i vincoli	Convergenza rapida	Richiede calcolo esplicito dei gradiente

Tabella 3.2: Ottimizzatori gradient-based

per ogni parametro in modo dinamico, basandosi su gradienti passati, come ADAM e ASMGRAD - descritti nella tabella (3.3), spesso usati in problemi di machine learning.

Nome	Funzionamento	Vantaggi	Svantaggi
ADAM	Utilizza momenti adattivi per aggiornare i pesi	Utile per problemi di grandi dimensioni	Sensibile alla scelta degli iperparametri
ASMGRAD	Modifica ADAM per evitare la decrescita del tasso di apprendimento	Migliora la generalizzazione rispetto ad ADAM	Più lento rispetto ad ADAM in alcuni casi

Tabella 3.3: Ottimizzatori basati su discesa del gradiente adattivi

Capitolo 4

Simulazione e risultati

In questo capitolo presentiamo la messa in pratica di quanto descritto nella parte teorica. Mostriamo come abbiamo implementato il **VQE** per risolvere istanze del *Travelling Salesman Problem* (TSP) e del *Knapsack Problem* (KP). Illustreremo poi i risultati ottenuti delle simulazioni e li discuteremo.

Il capitolo è strutturato introducendo prima l'ambiente di sviluppo utilizzato, **Qiskit** ed i suoi moduli ed un semplice esempio per mostrare un istanza tangibile di uso del VQE. Successivamente descriviamo le implementazioni del TSP e del KP, con le considerazioni fatte ed i risultati ottenuti.

4.1 Ambiente di sviluppo e Qiskit

4.1.1 Configurazione dell'ambiente di sviluppo

Per il lavoro di questa tesi tutto il codice è stato scritto in `Python 3.9.12`, utilizzando spesso `Jupyter Notebook` per favorire la leggibilità, sulla mia macchina locale, la quale dispone di un processore Intel Core i5-8400 e 24GB di RAM. Per i circuiti quantistici e la gestione dei problemi di ottimizzazione è stato utilizzato `Qiskit 1.3.2`, `Qiskit Aer 0.15.1` e `Qiskit Optimization 0.6.1`. Per le implementazioni degli algoritmi quantistici è stato usato `Qiskit Algorithms 0.3.1`. Oltre a questo, `matplotlib` è stato utilizzato per la generazione dei grafici, `numpy` per la gestione dei dati e `time` per la misurazione dei tempi di esecuzione.

4.1.2 Qiskit

Qiskit è un *software development kit* (*SDK*) open source originariamente sviluppato da IBM per la programmazione di computer quantistici[52]. Mette a disposizione strumenti per creare e manipolare circuiti quantistici e per eseguirli su simulatori classici o su computer quantistici reali, oltre a librerie per la risoluzione di problemi di ottimizzazione e l'implementazione di algoritmi quantistici[53].

Esempio 4.1.1

Vediamo un primo esempio dove creiamo il circuito quantistico che avevamo visto nella Fig. (1.2) nel Cap. 1. Per farlo utilizziamo la libreria di Qiskit come mostrato nel blocco di codice (4.1).

```
import qiskit

# Creazione di un circuito quantistico
qc = qiskit.QuantumCircuit(2) # Creiamo un circuito a due qubit
qc.h(0)           # Applichiamo una porta Hadamard al qubit 0
qc.cx(0, 1)       # Applichiamo una porta CNOT tra il qubit 0 e 1

qc.measure_all()  # Misuriamo tutti i qubit e ne otteniamo il
   risultato in bit classici, quindi otteniamo stringhe di bit che
   possono essere 00, 01, 10 o 11
```

Codice 4.1: Creazione di un circuito quantistico

Possiamo eseguire la **misura** di questo circuito usando un simulatore classico. Nel nostro caso, faremo uso di **Qiskit Aer**, una libreria che si basa su Qiskit e mette a disposizione simualtori ad alta performance[54]. Il codice (4.2) esegue la simulazione del circuito creato nel codice (4.1).

```
import qiskit_aer

# Configurazione simulare standard
backend = qiskit_aer.AerSimulator()

# Esecuzione del circuito e ottenimento dei risultati
```

```
job = backend.run(qc, shots=1024)
counts = job.result().get_counts()
```

Codice 4.2: Misura dei qubit

Il risultato che otteniamo è il numero di volte che abbiamo ottenuto ciascuna stringa di bit, quindi il risultato sarà un dizionario con le stringhe di bit come chiavi e il numero di occorrenze come valori. Nella Fig. (4.1) rappresentiamo graficamente i risultati ottenuti.

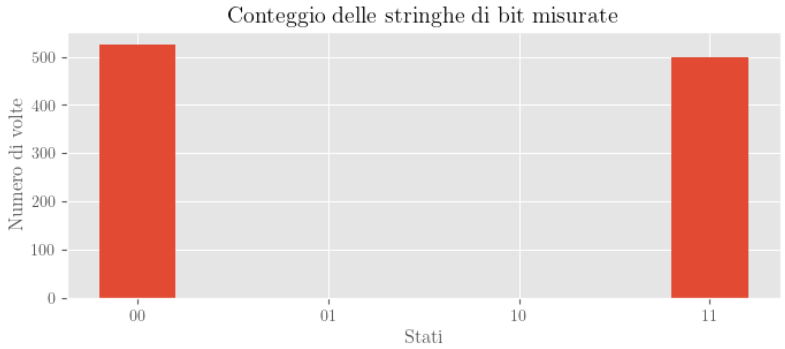

Figura 4.1: Risultati della simulazione del circuito quantistico

Come previsto, otteniamo in circa metà dei shot - quindi circa 500 volte - la stringa di bit 00 e nell'altra metà - sempre circa 500 volte - otteniamo 11, con invece probabilità del 0% di ottenere le stringhe 01 e 10, in quanto il circuito crea lo stato di Bell $|\Phi^+\rangle$. I risultati sono approssimati in quanto si tratta di simulazioni probabilistiche.

Se vogliamo ottenere i valori di aspettazione degli operatori di Pauli Z, possiamo invece fare uso di `Estimator`, sempre parte di Qiskit Aer, come mostrato nel codice (4.3).

```
from qiskit.quantum_info import SparsePauliOp
from qiskit_aer import Estimator

# Definizione degli operatori di Pauli
pauli_ops = [
    SparsePauliOp(["IZ"]), # Misura qubit 1 nella base Z
    SparsePauliOp(["ZI"]), # Misura qubit 2 nella base Z
    SparsePauliOp(["ZZ"]) # Misura qubit 1 e 2 nella base Z
]
```

```
# Utilizziamo Estimator per calcolare i valori di aspettazione
estimator = Estimator()
job = estimator.run([qc] * len(pauli_ops), pauli_ops, shots=1024)
results = job.result()
```

Codice 4.3: Calcolo dei valori di aspettazione

I risultati di questi calcoli si possono vedere nella Fig. (4.2), che come previsto mostrano valori di aspettazione di circa 0 per Z_1 e Z_2 e di circa 1 per Z_1Z_2. I valori non sono esatti in quanto si tratta sempre di simulazioni probabilistiche.

Figura 4.2: Valori di aspettazione degli operatori di Pauli

Qiskit Optimization

Qiskit Optimization è un *framework open-source* che offre strumenti per la gestione dei problemi di ottimizzazione, basati su `docplex`[55]. In particolare, ci permette di modellarli e ne automatizza la conversione a diverse rappresentazioni, come per esempio la formulazione QUBO, nonché implementa già al suo interno diversi problemi di ottimizzazione, tra cui il TSP ed il KP[56].

Qiskit Algorithms

Qiskit Algorithms è una libreria di algoritmi quantistici per Qiskit. Essa mette a disposizione diversi algoritmi, tra cui il VQE, il QAOA e l'Algoritmo

di Grover. Questi algoritmi sono implementati in modo da semplificarne l'uso e sono predisposti per essere utilizzati insieme a Qiskit Optimization[57].

La libreria include al suo interno alcuni algoritmi, come ad esempio `NumPyMinimumEigensolver`, che ricevono gli stessi input delle controparti quantistiche ma risolvono i problemi in maniera classica. Questo è utile nel breve termine, in quanto i problemi sono ancora piccoli abbastanza da essere risolti in maniera classica in tempi ragionevoli, per avere un punto di riferimento e confrontare i risultati ottenuti con il VQE.

In particolare per simulare il VQE noi faremo uso del `SamplingVQE`, un'implementazione del VQE ottimizzata per l'utilizzo di Hamiltoniane diagonali, come l'Hamiltoniana di Ising che noi usiamo. Esso usa la primitiva `Sampler` a differenza del VQE standard, che usa la primitiva `Estimator`, e piuttosto che ottenere i valori di aspettazione degli operatori di Pauli, campiona direttamente le stringhe di bit, che nel caso di un Hamiltoniana diagonale è sufficiente per calcolare il valore di aspettazione dell'energia attraverso una somma pesata delle stringhe di bit ottenute e delle loro probabilità di occorrenza.

4.2 Esempio di VQE

Mostriamo un esempio di VQE, dove il nostro circuito parametrizzato ha 2 variabili, θ_1 e θ_2, e l'Hamiltoniana è descritta dall'equazione (4.1).

$$\hat{H} = 3Z_1Z_2 + 5Z_1 + 1Z_2 \tag{4.1}$$

Iniziamo impostando l'Hamiltoniana e creando l'ansatz, visibile nella Fig. (4.3), come mostrato nel codice (4.4).

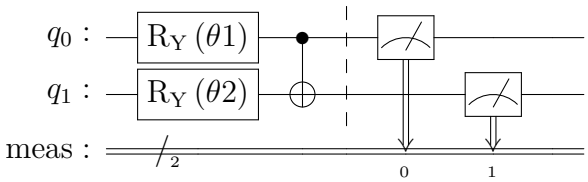

Figura 4.3: Ansatz del nostro esempio di VQE a 2 parametri e 2 qubit

```
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter
from qiskit.quantum_info import SparsePauliOp

# Definizione dell'Hamiltoniana
hamiltoniana = SparsePauliOp(["ZZ", "ZI", "IZ"], coeffs=[3.0, 5.0,
    1.0])

# Crea l'ansatz variazionale con due parametri
theta1 = Parameter("θ1")
theta2 = Parameter("θ2")
qc = QuantumCircuit(2)
qc.ry(theta1, 0)
qc.ry(theta2, 1)
qc.measure_all()
```

Codice 4.4: Creazione di un'ansatz per il VQE

Ora creiamo una funzione che determina il valore di aspettazione dell'energia dati i parametri θ_1 e θ_2, come mostrato nel codice (4.5).

```
# @parametri - e' un array di due elementi
def aspettazione_energia(parametri):
    estimator = Estimator()

    # Vincolo i parametri al circuito
    circuito_vincolato =
        circuito_quantistico.assign_parameters({theta1: parametri[0],
        theta2: parametri[1]})

    # Utilizzo l'Estimator per ottenere i valori di aspettazione
    result = estimator.run([circuito_vincolato], [hamiltoniana],
        shots=50).result()
    energia = result.values[0].real
    return energia
```

Codice 4.5: Calcolo del valore di aspettazione dell'energia

Ora possiamo utilizzare un ottimizzatore, in questo esempio abbiamo scelto `SPSA` in quanto permette l'uso di un `callback` attraverso il quale possiamo monitorare l'andamento dell'ottimizzazione, per trovare i parametri θ_1 e θ_2 che minimizzano l'energia, come mostrato nel codice (4.6).

```
from qiskit_algorithms.optimizers import SPSA

# Inizializzazione casuale dei parametri
icp = np.random.rand(2) * np.pi

# Inizializzazione dell'ottimizzatore
#   > Senza specificarlo, il numero massimo di iterazioni e' 100
ottimizzatore = SPSA()

# Eseguo l'ottimizzazione
risultato_ott = ottimizzatore.minimize(aspettazione_energia, icp)
```

Codice 4.6: Ottimizzazione dei parametri

Prima di mostrare i risultati, utilizziamo un calcolo classico per trovare il valore di aspettazione dell'energia minimo per confrontare i risultati che otterremo con il VQE. Per farlo abbiamo costrutito l'Hamiltoniana in forma matriciale e abbiamo calcolato il suo autovalore minimo, come mostrato nel codice (4.7). L'autovalore minimo dell'Hamiltoniana ottenuto, pertanto il valore di aspettazione dell'energia minimo, è -7.

```
import numpy as np

# Definiamo le matrici Z e I
Z = np.array([[1, 0], [0, -1]])
I = np.eye(2)

# Costruiamo l'Hamiltoniana H = 3*ZZ + 5*ZI + IZ
H = 3 * np.kron(Z, Z) + 5 * np.kron(Z, I) + np.kron(I, Z)

# Calcoliamo gli autovalori e gli autovettori
energie, stati = np.linalg.eigh(H)
```

Codice 4.7: Calcolo classico del valore di aspettazione dell'energia

Abbiamo quindi eseguito 10 diverse istanze del VQE, ciascuna con dei punti iniziali casuali, e abbiamo ottenuto i risultati mostrati nella Fig. (4.4). Nell'asse delle ascisse abbiamo il valore dell'energia trovato e nelle ordinate abbiamo il numero di istanze che hanno raggiunto quello specifico valore di energia, con un margine di ± 0.5. Notiamo in particolare che 8 delle 10 istanze hanno raggiunto il valore di aspettazione -7, mentre le altre due hanno raggiunto -3.

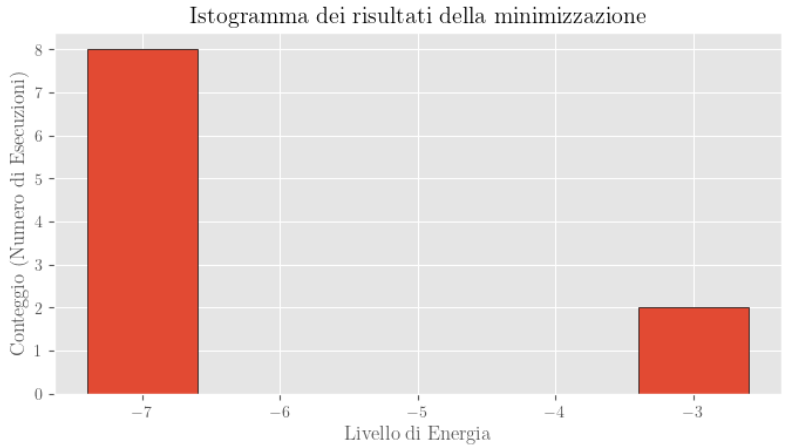

Figura 4.4: Risultati dell'ottimizzazione del VQE.

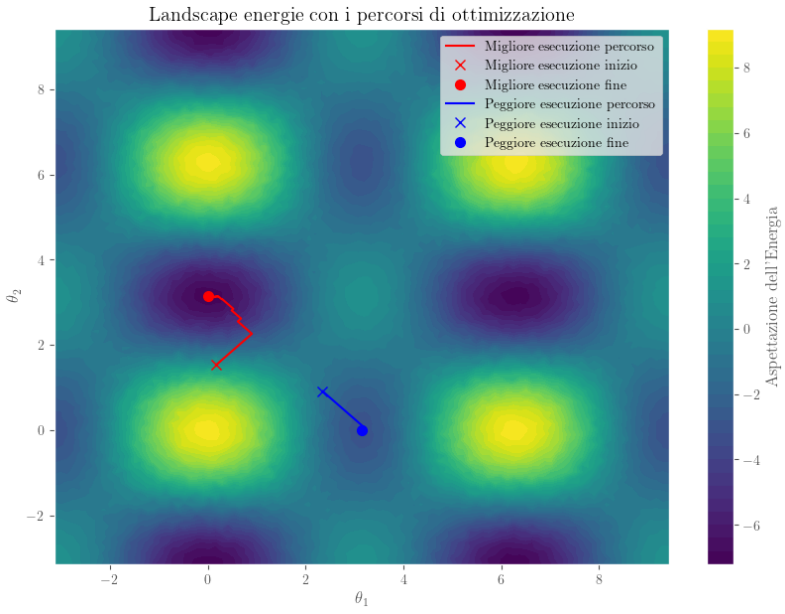

Figura 4.5: Landscape dell'energia al variare dei parametri.

Per approfondire ancor meglio cosa succeda, abbiamo anche generato il *landscape* delle energie al variare dei parametri θ_1 e θ_2, visible nella Fig. (4.5). All'interno di questo abbiamo inserito i percorsi che l'ottimizzatore ha seguito

per trovare il minimo, mostrando come a volte possa rimanere bloccato in minimi locali.

Andiamo anche ad eseguire questi stessi passaggi modificando però l'ansatz utilizzato, rimuovendo la porta *CNOT* finale, come visibile nella Fig. (4.6). Abbiamo ottenuto le misurazioni di aspettazione dell'energia mostrate nella Fig. (4.7) ed il nostro nuovo *landscape* è visibile nella Fig. (4.8). Quello che ci è immediato notare è che questo nuovo ansatz è molto migliore a raggiungere il minimo, e questo lo si può giustificare osservando il *landscape*, dove si nota che c'è un assenza di minimi locali. Non vi è comunque una convergenza totale, in quanto l'ottimizzatore può finire il numero massimo di iterazioni senza raggiungere il minimo.

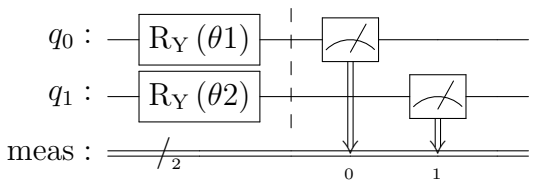

Figura 4.6: Ansatz del nostro esempio di VQE senza la porta CNOT

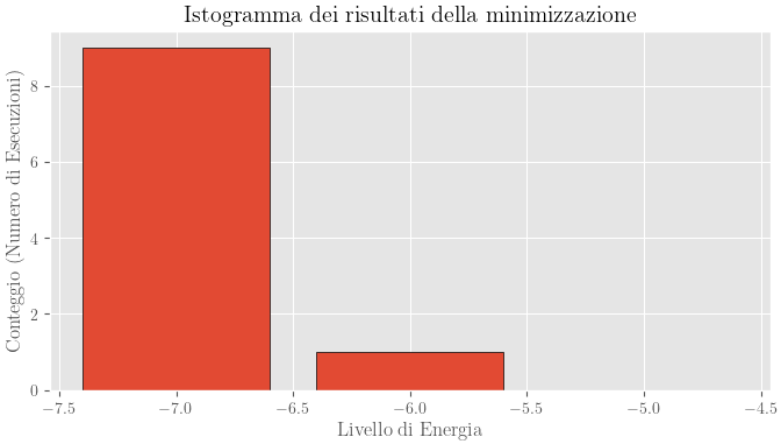

Figura 4.7: Risultati dell'ottimizzazione del VQE senza CNOT.

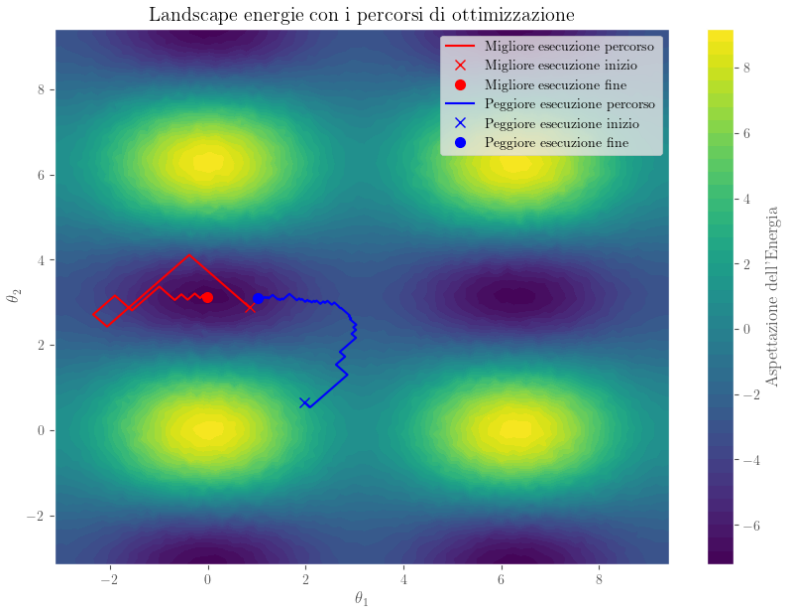

Figura 4.8: Landscape dell'energia al variare dei parametri senza CNOT.

Infine, mostriamo un ultimo caso, quello dove sostituiamo la porta CNOT con una porta CZ con control q_0 e con target q_1, ovvero una porta di controllo con due qubit che applica la porta Z al qubit target solo se il qubit di controllo è nello stato $|1\rangle$, e due porte Hadamard finali, come visibile nella Fig. (4.9). I risultati ottenuti sono mostrati nella Fig. (4.10) ed il *landscape* è visibile nella Fig. (4.11).

Con l'ansatz così costruito ci imbattiamo in un problema: l'ottimizzatore non riesce a trovare il minimo reale, in quanto, da come possiamo vedere dal *landscape*, esso non è raggiungibile con nessun valore che i parametri potrebbero assumere perché è proprio assente il valore dell'energia fondamentale -7, avendo invece come minimo globale del landscape il valore -5. L'ansatz genera quindi un landscape che non è sufficientmente espressivo da poter contenere il vero minimo dell'Hamiltoniana.

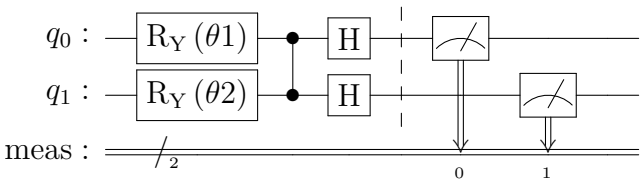

Figura 4.9: Ansatz del nostro esempio di VQE con CZ e Hadamard

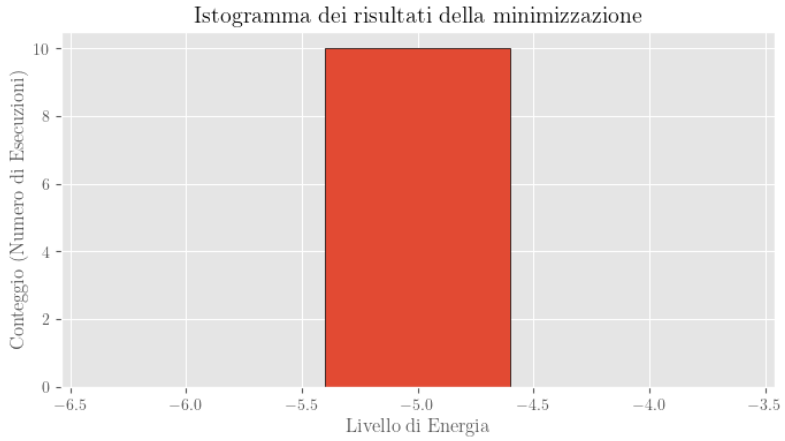

Figura 4.10: Risultati dell'ottimizzazione del VQE con CZ e Hadamard.

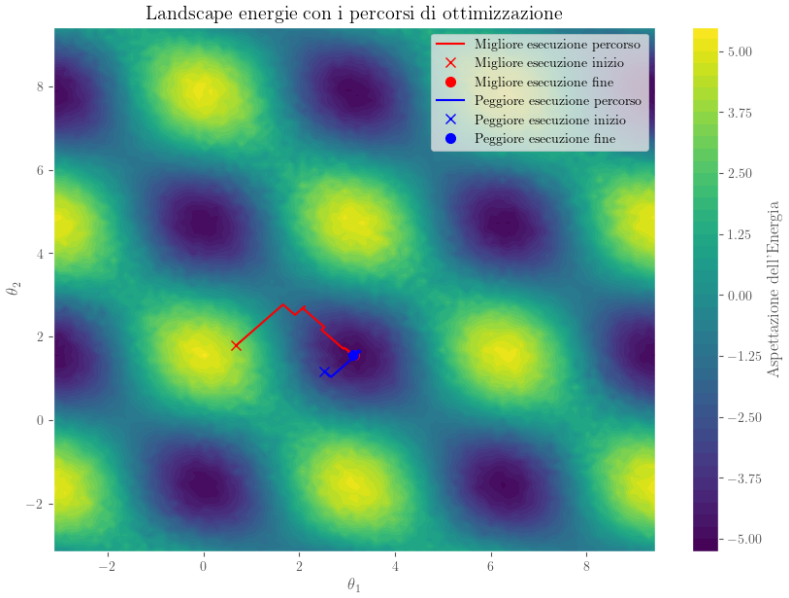

Figura 4.11: Landscape dell'energia al variare dei parametri con CZ e Hadamard.

4.2.1 Analisi dei risultati

Riassumendo le informazioni emerse da questi esempi, abbiamo osservato quanto siano fondamentali l'ansatz e la scelta dei parametri iniziali per garantire la convergenza dell'ottimizzatore. In particolare, i tre problemi riscontrati sono stati: la **lentezza nella convergenza**, la **convergenza a minimi locali** e l'**assenza di un minimo globale** che coincida all'autovalore minimo dell'Hamiltoniana.

Per causa della semplicità di queste istanze, non le utilizzeremo come riferimento per la creazione degli ansatz o dei parametri iniziali per la risoluzione

dei problemi di TSP e KP, in quanto difficilmente qualcosa osservato in questi esempi si potrà generalizzare a problemi più complessi. Tuttavia la loro analisi ci permette di comprendere meglio quelle che potrebbero essere le difficoltà che incontreremo nella risoluzione dei problemi reali.

4.3 Simulazioni

Questa sezione dedicata alle simulazioni effettuate con il VQE per risolvere problemi di ottimizzazione reali è strutturata in due parti, una per il **Travelling Salesman Problem** (TSP) e una per il **Knapsack Problem** (KP).

Il flusso di lavoro per entrambi i problemi è stato il seguente:

1. **Creazione del modello**: abbiamo creato il modello del problema utilizzando le funzionalità di Qiskit Optimization, in particolare abbiamo utilizzato la classe `QuadraticProgram` per modellare il problema in forma QUBO.

2. **Creazione dell'ansatz**: abbiamo creato più ansatz per il VQE, alcuni più complessi ed altri meno, in modo da poter confrontare i risultati ottenuti e valutare la qualità della soluzione.

3. **Ottimizzazione**: abbiamo utilizzato più ottimizzatori per eseguire le nostre prove, in modo da poterli confrontare e valutare quale funzionasse meglio. Abbiamo provato **SPSA**, **COBYLA** ed **L-BFGS-B**.

Abbiamo eseguito prove facendo variare sempre la dimensione del problema da più piccolo a più grande e, oltre a questo, confrontando un aspetto ulteriore - ovvero ansatz oppure ottimizzatori - in modo da poter valutare come questi influenzassero i risultati ottenuti. Trovati dei buoni candidati, abbiamo cercato di fare ulteriori prove per cercare di ottimizzare i risultati ottenuti.

4.3.1 Knapsack Problem

Le prime prove che abbiamo fatto sono state sul **KP**, in quanto è un problema più semplice dei due. Questo ci permette di eseguire più tentativi in tempi ridotti e farci un'idea delle scelte più opportune da fare per risolvere il problema.

Impostazione generale della simulazione

Abbiamo generato il modello utilizzando `qiskit_optimization` utilizzando le funzioni così come descritto nel codice (4.8).

```
from qiskit_optimization.applications.knapsack import Knapsack
from qiskit_optimization.converters import QuadraticProgramToQubo

# Creazione del modello: costruzione del problema KP
# @values - array dei valori degli oggetti
# @weights - array dei pesi degli oggetti
# @capacity - capacita' dello zaino
knapsack_app = Knapsack(values, weights, capacity)
qp = knapsack_app.to_quadratic_program()

# Conversione in QUBO e in problema Ising
qubo_converter = QuadraticProgramToQubo()
qubo = qubo_converter.convert(qp)
qubit_op, offset = qubo.to_ising()
```

Codice 4.8: Creazione del modello per il KP

Il VQE è stato implementato come descritto nel codice (4.9), dove l'ansatz è un circuito generato separatamente e passato come parametro alla funzione.

```
from qiskit_algorithms.optimizers import SPSA, COBYLA, L_BFGS_B
from qiskit_algorithms import SamplingVQE
from qiskit_aer.primitives import Sampler

# Scelta dell'ottimizzatore (SPSA come esempio)
optimizer = SPSA(maxiter=100)

# Creazione del VQE classico-quantistico
vqe = SamplingVQE(
    sampler=Sampler(),
    optimizer=optimizer,
    ansatz=ansatz     # Ansatz generato separatamente
)

# Esecuzione della simulazione VQE con controllo del tempo
```

```
start_time = time.time()
result = vqe.compute_minimum_eigenvalue(qubit_op)
elapsed_time = time.time() - start_time

# Interpretazione della soluzione
binary_solution = knapsack_app.sample_most_likely(result.eigenstate)
decision_vars = binary_solution[:len(values)] # Rimuoviamo i bit
    delle slack variables
chosen_items = [i for i, bit in enumerate(decision_vars) if bit == 1]

# Calcolo del valore e del peso totale degli oggetti selezionati
total_value = sum(values[i] for i in chosen_items)
total_weight = sum(weights[i] for i in chosen_items)
```

Codice 4.9: Esecuzione del VQE

Abbiamo inoltre utilizzato `NumPyMinimumEigensolver` per calcolare il valore di aspettazione dell'energia minimo in modo classico, come mostrato nel codice (4.10).

```
from qiskit.algorithms import NumPyMinimumEigensolver

# Calcolo classico del valore di aspettazione dell'energia minimo
np_solver = NumPyMinimumEigensolver()
np_result = np_solver.compute_minimum_eigenvalue(qubit_op)
energy_np = np_result.eigenvalue.real # Valore di aspettazione
    dell'energia minimo
value_np = - (np_result.eigenvalue.real + offset) # Valore totale
    degli oggetti selezionati calcolato in modo classico
```

Codice 4.10: Calcolo classico del KP

Così facendo abbiamo tutti gli strumenti necessari per eseguire i nostri confronti, che abbiamo deciso di incentrare sui seguenti parametiri:

- **Tempo di esecuzione**: misurato in secondi, per valutare quanto tempo impiega il VQE per trovare la soluzione.
- **Errore relativo dell'energia minima**: calcolato come:

$$E_R^{En} = \frac{E_{\text{VQE}} - E_{\text{classico}}}{E_{\text{classico}}} \tag{4.2}$$

dove E_{VQE} è il valore di aspettazione dell'energia minimo ottenuto con il VQE e E_{classico} è il valore di aspettazione dell'energia minimo calcolato in modo classico, utilizzando `NumPyMinimumEigensolver`. Questo ci permette di valutare quanto il VQE si discosti dal valore esatto. Questo valore è stato espresso come percentuale.

- **Errore relativo del valore massimo**: calcolato come:

$$E_R^V = \frac{V_{\text{VQE}} - V_{\text{classico}}}{V_{\text{classico}}} \tag{4.3}$$

dove V_{VQE} è il valore totale degli oggetti selezionati ottenuto con il VQE e V_{classico} è il valore totale degli oggetti selezionati calcolato in modo classico. Con questo valore possiamo capire la bontà reale delle soluzioni ottenute. Da notare che nel caso una soluzione violasse i vincoli, il valore totale degli oggetti selezionati è stato impostato come 0. Anche questo valore è stato espresso come percentuale.

Simulazione al variare di ansatz e ottimizzatori

La realizzazione di una prima simulazione è avvenuta seguendo il seguente schema:

- Le istanze generate sono dai 3 agli 8 oggetti, con valori e pesi generati casualmente attraverso un **seed**, in modo da poter riprodurre i risultati, e con una capacità dello zaino pari al 50% del peso totale degli oggetti. I valori variavano tra 1 e 10 ed i pesi tra 1 e 5.

- Abbiamo generato 5 diversi ansatz attraverso `TwoLocal`, il quale ci permette di creare un circuito parametrizzato con un numero ed un tipo variabile di porte quantistiche[58].

  Abbiamo chiamato queste ansatz $A1, ..., A5$ ed esse variano da 2 a 4 *reps*, ovvero ripetizioni dell'insieme di porte parametrizzate. Fanno uso di porte `RY`, `RZ`, `RY`, `CX` e `CZ` ed utilizzano diverse strategie di *entanglement*, ovvero `full`, `linear` o `circular`. La loro costruzione è mostrata nel codice (4.11).

- Più ottimizzatori sono stati utilizzati, in modo da vedere quanto cambiasse ai risultati, per un totale di 3 diversi ottimizzatori, `SPSA`, `COBYLA` ed `L-BFGS-B`.

- Abbiamo realizzato 8 simulazioni per ogni combinazione di ansatz, ottimizzatore e dimensione del problema, dove ognuna aveva parametri iniziali generati casualmente, in modo che i risultati ci indichino meglio il comportamento dell'algoritmo.

  Da notare che con l'ottimizzatore `L-BFGS-B` ci siamo limitati a 3 simulazioni per ogni combinazione. Questo è dovuto al fatto che esso è molto più lento degli altri due ottimizzatori e replicare gli stessi test sarebbe stato troppo dispendioso in termini di tempo.

- Per gli errori relativi calcolati, ne mostriamo la media ottenuta dalle 8 simulazioni ed il valore minimo trovato tra queste 8.

```
from qiskit.circuit.library import TwoLocal

# Creazione di ansatz per il KP
A1 = TwoLocal(num_qubits, ['rx'], 'cx', reps=2, entanglement='linear')
A2 = TwoLocal(num_qubits, ['ry'], 'cx', reps=2, entanglement='full')
A3 = TwoLocal(num_qubits, ['rx', 'rz'], 'cx', reps=4,
    entanglement='circular')
A4 = TwoLocal(num_qubits, ['ry', 'rz'], 'cx', reps=3,
    entanglement='full')
A5 = TwoLocal(num_qubits, ['rx', 'ry'], 'cz', reps=4,
    entanglement='circular')
```

Codice 4.11: Creazione di ansatz per il KP

Abbiamo quindi eseguito le simulazioni ed ottenuto i grafici mostrati nella Fig. (4.12) per **SPSA**, nella Fig. (4.13) per **COBYLA** e nella Fig. (4.14) per **L-BFGS-B**.

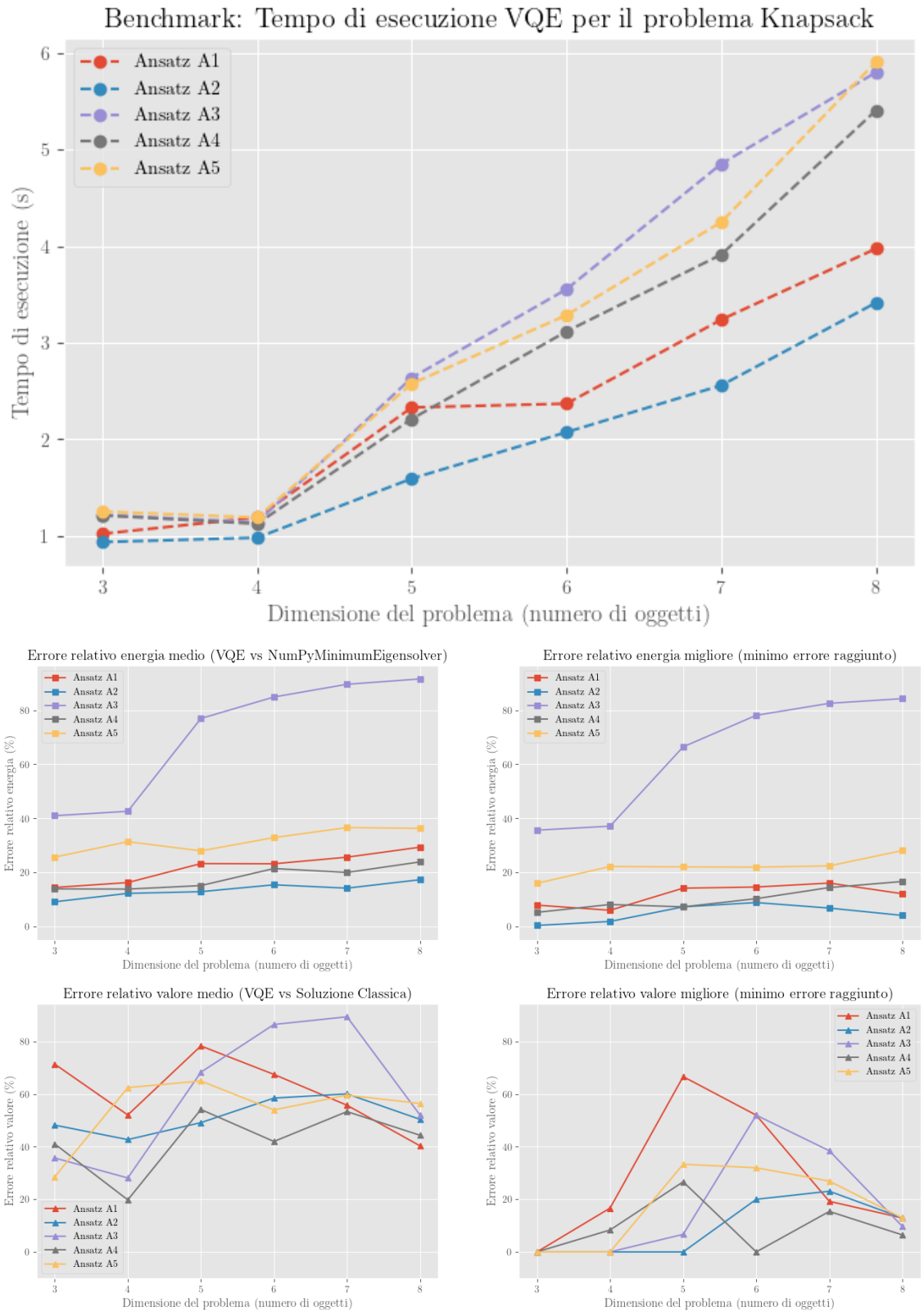

Figura 4.12: Risultati delle simulazioni con SPSA.

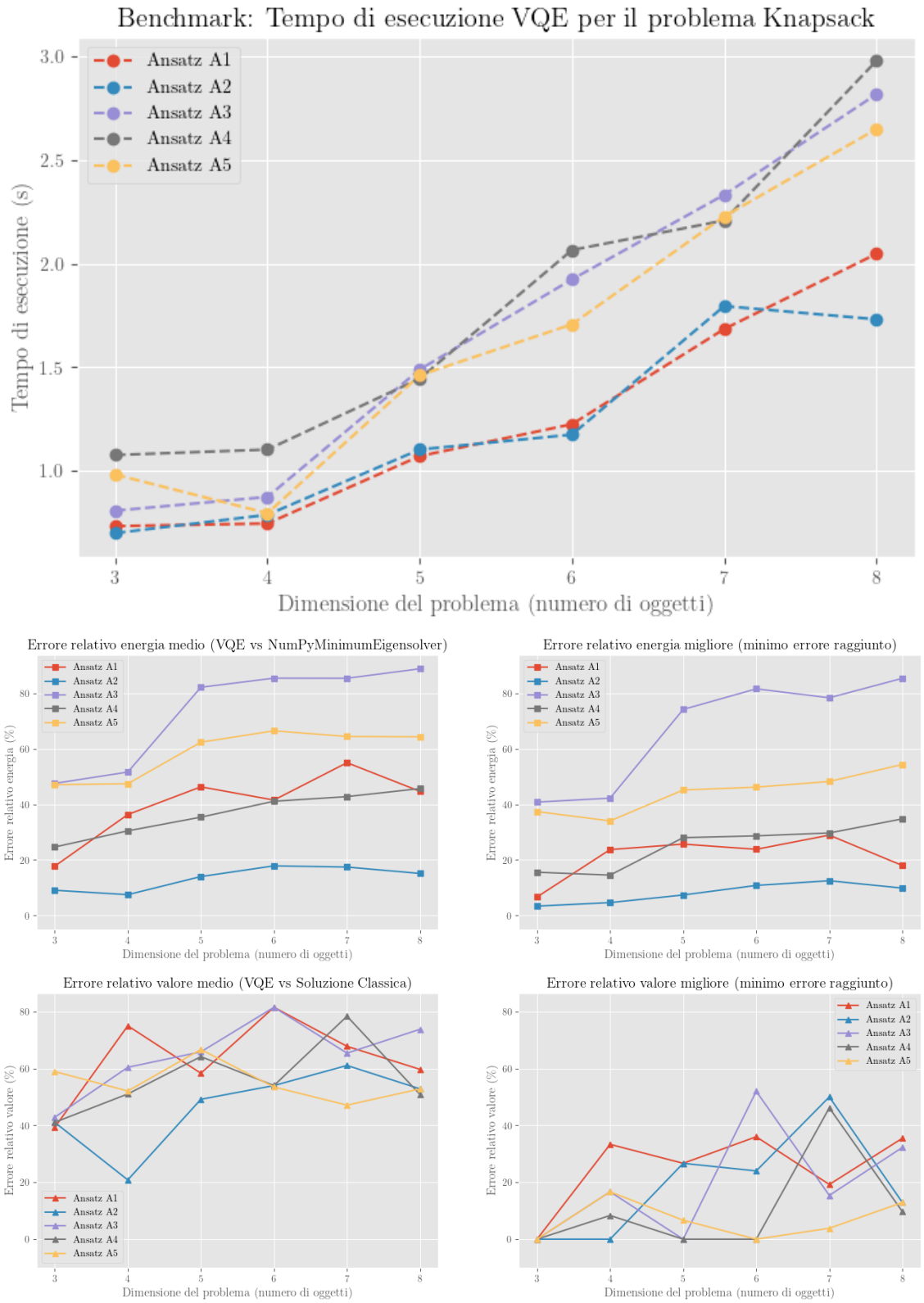

Figura 4.13: Risultati delle simulazioni con COBYLA.

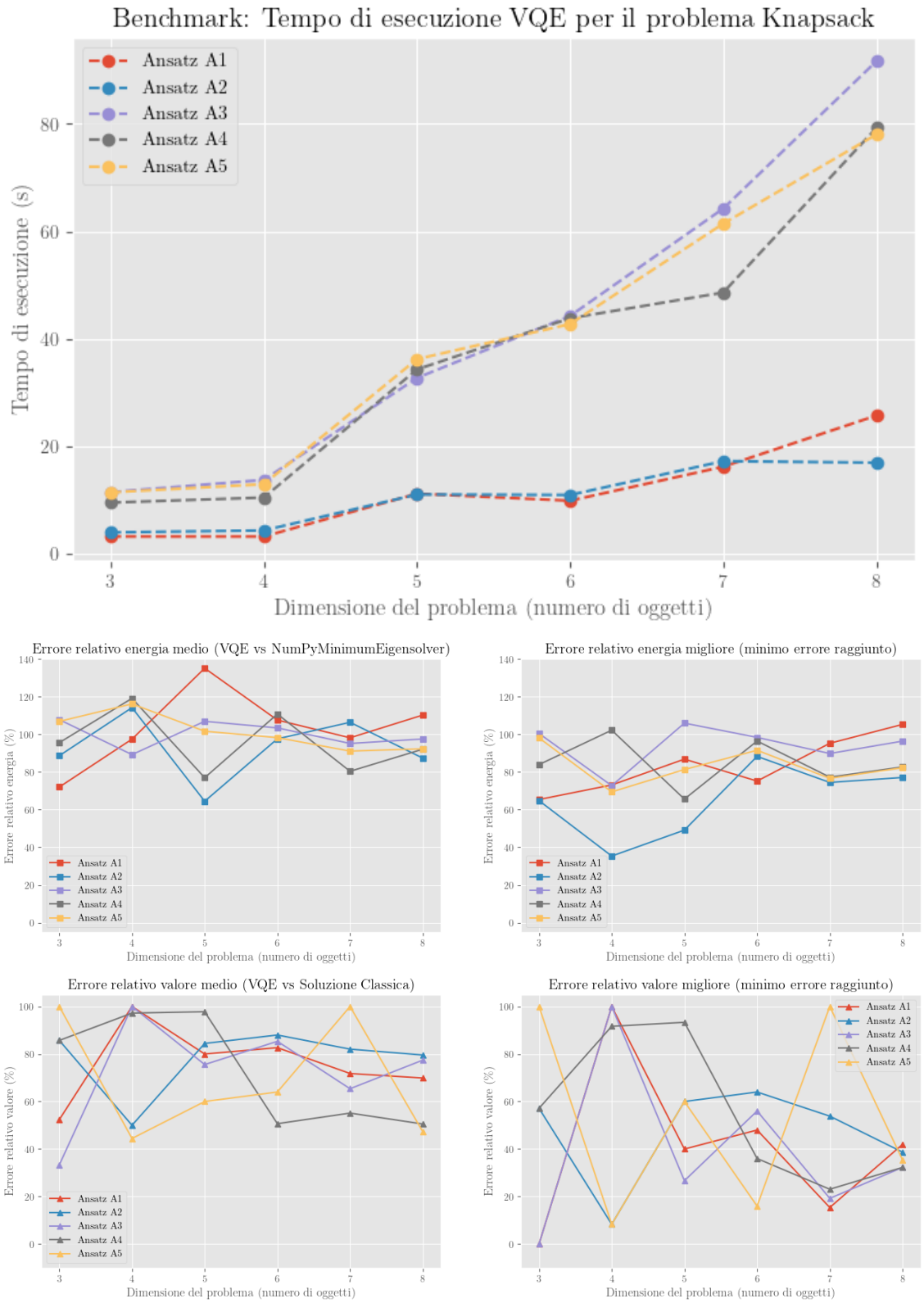

Figura 4.14: Risultati delle simulazioni con L-BFGS-B.

Analisi dei risultati del KP e nuove simulazioni

I risultati ottenuti si allineano in parte a quanto ci aspettavamo ed in parte ci hanno sorpreso. I punti principali che possiamo evidenziare sono:

- **Tempo di esecuzione**: il tempo di esecuzione cresce rapidamente ed in modo proporzionale alla dimensione del problema ed alla dimensione del circuito, entrambe proprietà che ne aumentano il numero di parametri. L-BFGS-B è l'ottimizzatore più lento, seguito da SPSA e COBYLA, con velocità simili.

- **Errore relativo dell'energia minima**: l'errore relativo dipende innanzitutto dall'ottimizzatore, anche in questo caso L-BFGS-B è il peggiore, mentre SPSA e COBYLA sono più comparabili, con COBYLA che ha ottenuto un ottimizzazione migliore per $A2$ nelle varie dimensioni del problema, ma con SPSA che ha avuto più costanza nell'avere risultati buoni su un maggior numero di ansatz.

- **Errore relativo del valore massimo**: questi risultati ci evidenziano un problema fondamentale di questo approccio: seppur noi ci avviciniamo abbastanza al valore esatto dell'energia minima, il valore totale degli oggetti selezionati difficilmente si avvicina a quello ottimale, con una differenza che notiamo essere, mediamente tra il 40% ed l'80%.

  Osserviamo che nelle istanze migliori i risultati che troviamo sono a volte il valore ottimale, anche se non sempre - e qui le percentuali grandi di distacco sono motivate da un numero piccolo di oggetti selezionabili, dove anche un solo errore può portare ad una differenza molto grande. Questo ci evidenzia quanto sia importante il valore iniziale dei parametri per l'ottimizzatore, in quanto hanno un impatto notevole sulla convergenza.

- **Diverse performance degli ansatz**: gli ansatz si sono comportati in maniera analoga con tutti gli ottimizzatori, pertanto, seppur esistano differenze da caso a caso che è utile notare, possiamo comunque analizzarne il quadro generale e farci un'idea di cosa funziona e cosa non funziona: $A2$, $A4$ ed $A1$ hanno avuto le performance migliori, in quest'ordine, mentre $A5$ ed $A3$ sono risultati essere i peggiori, soprattutto $A3$. Questo ci fa intuire che un numero ridotto di `reps`, che vada da 2 a 3, sia migliore. Inoltre, l'entanglement `full` sembra essere migliore rispetto a `circular` o a `linear`.

Con queste nuove conoscenze, considerando SPSA come ottimizzatore migliore per eseguire le simulazioni vista la sua consistenza, cerchiamo di esplorare alcuni ansatz simili a $A2$, $A4$ ed $A1$ per cercare di ottenere risultati migliori. Abbiamo quindi realizzato un nuovo set di simulazioni, con 5 nuovi ansatz, $A6, ..., A10$, che sono simili ai precedenti ma con alcune modifiche, come mostrato nel codice (4.12). Abbiamo eseguito le simulazioni e ottenuto i risultati mostrati nelle Fig. (4.15).

```
# Creazione di nuove ansatz per il KP
A6 = TwoLocal(num_qubits, ['rx'], 'cx', reps=2, entanglement='full')
A7 = TwoLocal(num_qubits, ['rx', 'rz'], 'cx', reps=1,
    entanglement='full')
A8 = TwoLocal(num_qubits, ['rx', 'rz'], 'cz', reps=1,
    entanglement='full')
A9 = TwoLocal(num_qubits, ['rx', 'ry'], 'cx', reps=2,
    entanglement='full')
A10 = TwoLocal(num_qubits, ['ry', 'rz'], 'cz', reps=3,
    entanglement='full')
```

Codice 4.12: Creazione di ansatz per il KP

Possiamo subito notare che questi nuovi risultati sono decisamente migliori rispetto ai precedenti, in quanto l'errore relativo dell'energia minima è diminuita, ora assumendo valori tra il 7% ed il 25%; con gli ansatz migliori, ovvero $A8$ ed $A7$, si rimane con costanza sotto al 18%. In generale possiamo quindi dedurre che **un numero ridotto di `reps` è migliore**, in questo caso 1. Inoltre da questi esperimenti notiamo che la porta `CZ` ha più costanza rispetto alla porta `CX`, ma in generale **la scelta della porta non sembra influenzare molto i risultati**.

Per l'ansatz $A8$ abbiamo realizzato anche un grafico che ci mostra la sua distribuzione dei risultati riguardanti gli errori relativi - che abbiamo cumulato approssimando i valori a $\pm 5\%$, come mostrato nella Fig. (4.16). Notiamo che riusciamo ad arrivare a valori inferiori al 20% di errore relativo all'energia minima in modo consistente, con un picco a circa 10%, indipendentemente quindi dai valori iniziali dei parametri. L'errore relativo del valore massimo invece si alterna tra una risoluzione corretta del problema, dove quindi l'errore è 0%, ed un insieme di soluzioni errate, con un errore del $35\% \pm 5\%$.

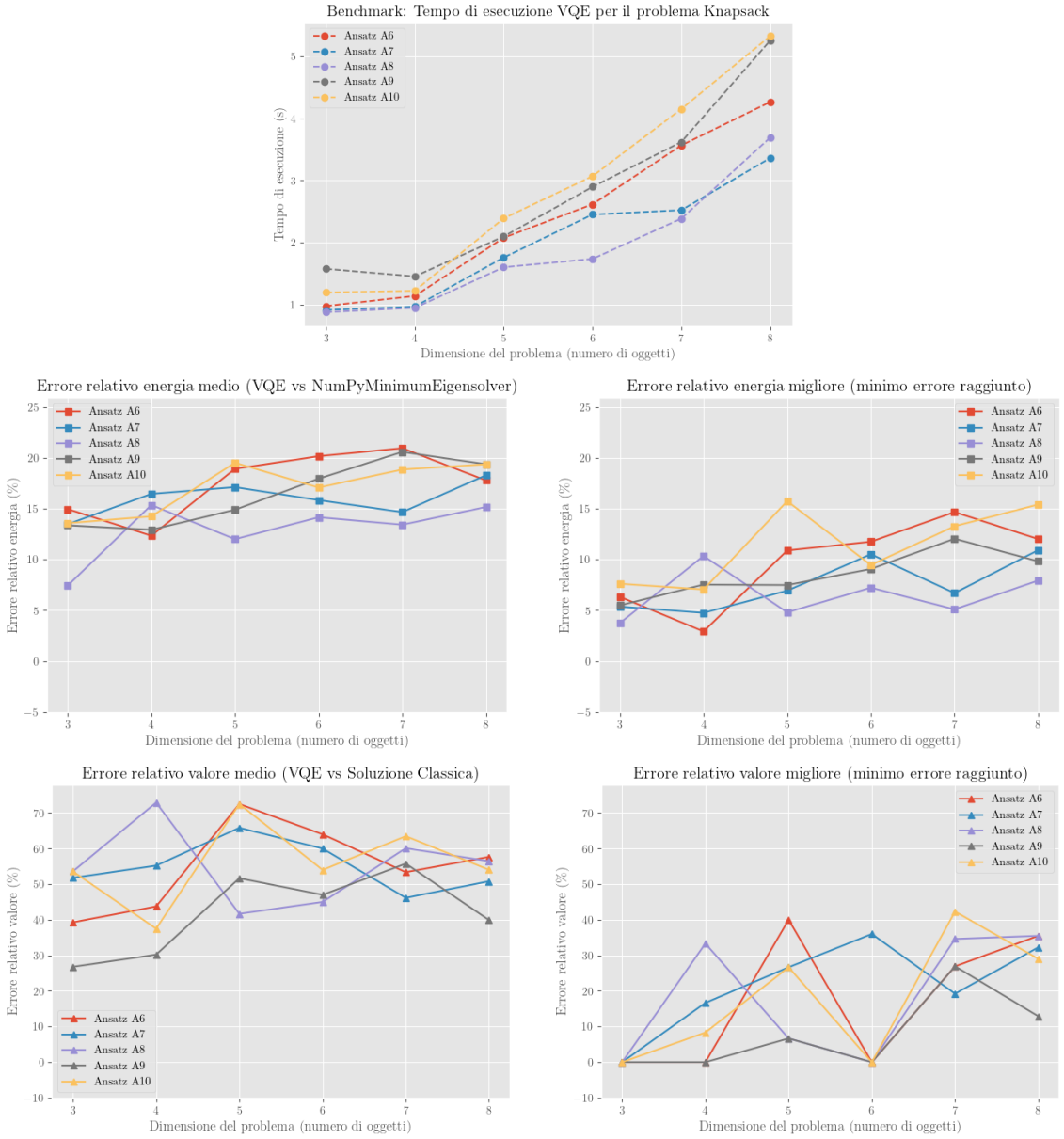

Figura 4.15: Risultati delle simulazioni con SPSA e nuovi ansatz.

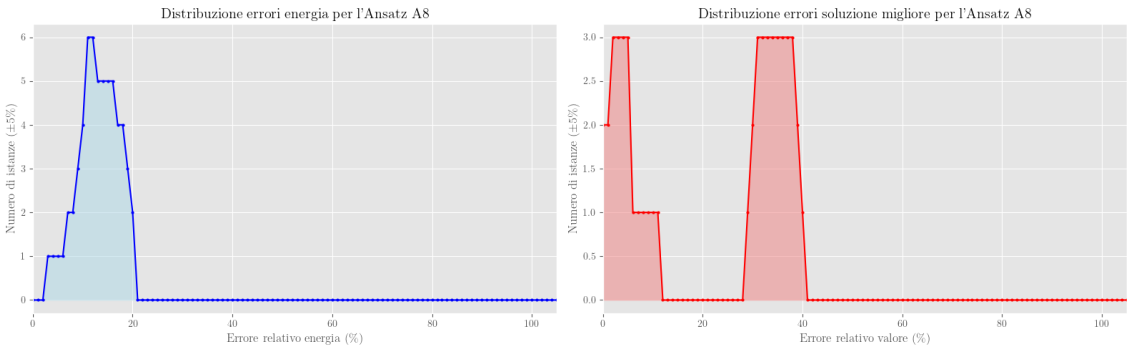

Figura 4.16: Distribuzione dei risultati con SPSA per l'ansatz A8.

Facendo ulteriori prove con ansatz che continuano lungo questo filo logico non ho ottenuto un miglioramento dei risultati, pertanto mi limiterò a considerare questi come il meglio raggiungibili utilizzando questo metodo *naive* di approccio alla creazione dell'ansatz.

4.3.2 Traveling Salesman Problem

Il **TSP** è un problema più complesso del KP, in quanto la sua natura combinatoriale lo rende più difficile da risolvere. Abbiamo quindi seguito lo stesso schema di prima, ma con alcune modifiche:

- Le istanze generate sono dai 3 ai 4 nodi, con i costi associati agli archi generati casualmente attraverso un **seed**, in modo da poter riprodurre i risultati. I costi variavano tra 1 e 10.

  Sono state eseguite più prove per cercare di raggiungere i 5 nodi, ma il tempo di esecuzione era troppo lungo per poter ottenere risultati significativi.

- Abbiamo generato 5 diversi ansatz, $A11, ..., A15$, attraverso `TwoLocal`, ispirandoci a quanto appreso con il KP - ovvero di mantenerle di piccole dimensioni e piuttosto semplici, come mostrato nel codice (4.13).

- Abbiamo utilizzato unicamente **SPSA** e **COBYLA** come ottimizzatori.

- Abbiamo realizzato 3 simulazioni per ogni combinazione di ansatz, ottimizzatore e dimensione del problema in modo da rendere i risultati più accurati.

```
# Creazione di ansatz per il TSP
A11 TwoLocal(num_qubits, ['rx'], 'cx', reps=1, entanglement='full')
A12 = TwoLocal(num_qubits, ['rx', 'rz'], 'cx', reps=1,
    entanglement='full')
A13 = TwoLocal(num_qubits, ['rx', 'rz'], 'cz', reps=1,
    entanglement='full')
A14 = TwoLocal(num_qubits, ['ry', 'rx', 'rz'], 'cx', reps=1,
    entanglement='full')
A15 = TwoLocal(num_qubits, ['rx', 'rz'], 'cx', reps=2,
    entanglement='full')
```

Codice 4.13: Creazione di ansatz per il TSP

L'implementazione è uguale a quella proposta per il KP, con la differenza che il modello è generato con `Tsp` al posto di `Knapsack`, come mostrato nel codice (4.14).

```
from qiskit_optimization.applications.tsp import Tsp

# Creazione del modello: costruzione del problema TSP
# @costs - matrice dei costi degli archi
tsp_app = Tsp(costs)
qp = tsp_app.to_quadratic_program()

# Conversione in QUBO e in problema Ising
qubo_converter = QuadraticProgramToQubo()
qubo = qubo_converter.convert(qp)
qubit_op, offset = qubo.to_ising()
```

Codice 4.14: Creazione del modello per il TSP

L'esecuzione del VQE rimane invariata. Le modifiche apportate sono state quelle di cambiare l'interpretazione dei risultati ed ottenere la sequenza dei nodi visitati, come mostrato nel codice (4.15).

```
# Impostazione del VQE
vqe = SamplingVQE(
    sampler=Sampler(),
    optimizer=optimizer, # SPSA o COBYLA
    ansatz=ansatz      # Ansatz generato separatamente
)

# Esecuzione della simulazione VQE con controllo del tempo
start_time = time.time()
result = vqe.compute_minimum_eigenvalue(qubit_op)
elapsed_time = time.time() - start_time

# Interpretazione della soluzione
binary_solution = tsp_app.sample_most_likely(result.eigenstate)
nodes_sequence = tsp_app.interpret(binary_solution)
```

Codice 4.15: Esecuzione del VQE per il TSP

Gli obiettivi che volevamo misurare sono rimasti gli stessi del KP, ovvero tempo di esecuzione, errore relativo dell'energia minima e errore relativo del valore massimo. Come nel KP, in caso di violazione dei vincoli, l'errore relativo del valore massimo è impostato a 100%.

I risultati ottenuti dalle nostre simulazioni sono visibili nella Fig. (4.17) per **SPSA** e nella Fig. (4.18) per **COBYLA**.

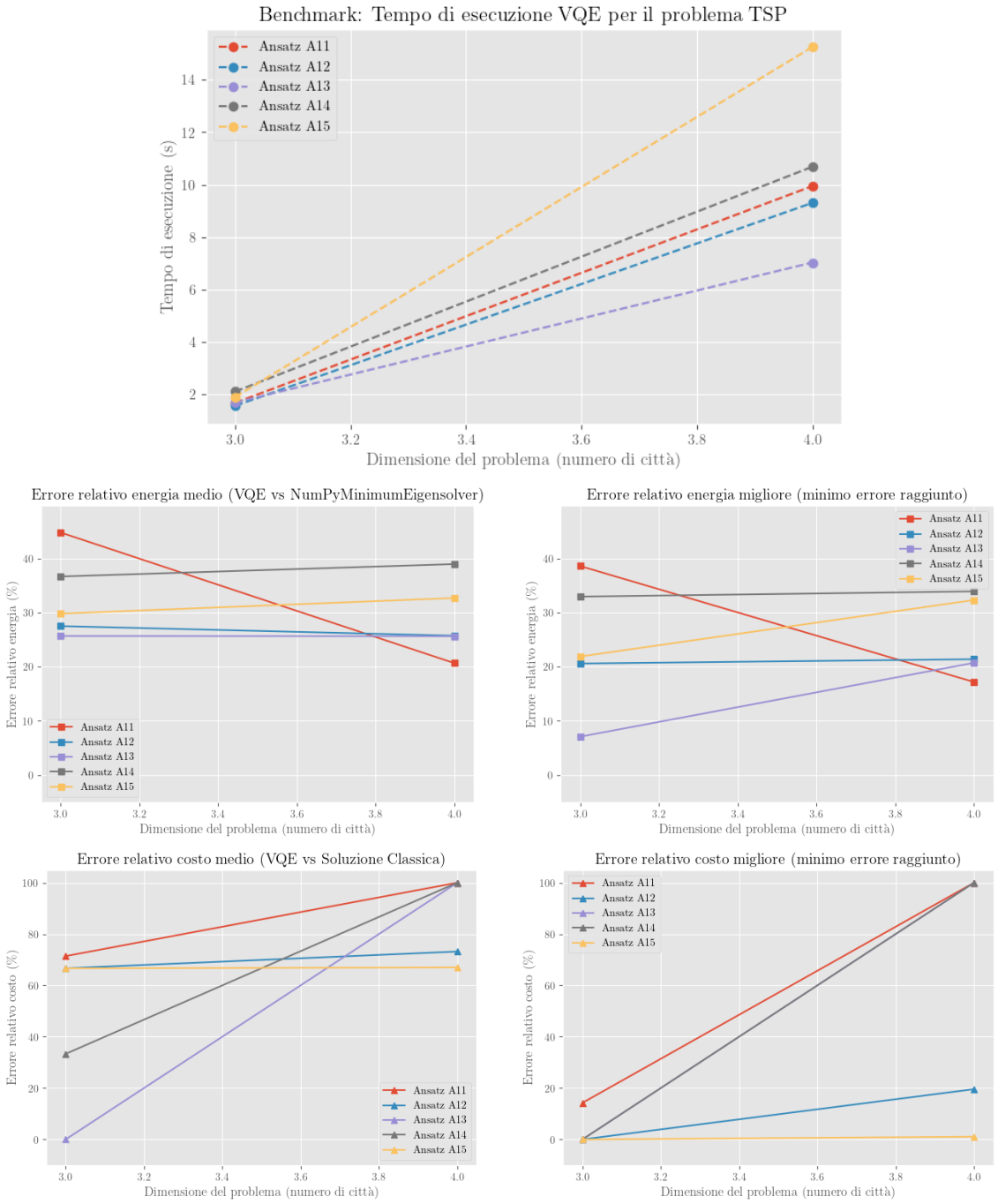

Figura 4.17: Risultati delle simulazioni con SPSA per il TSP.

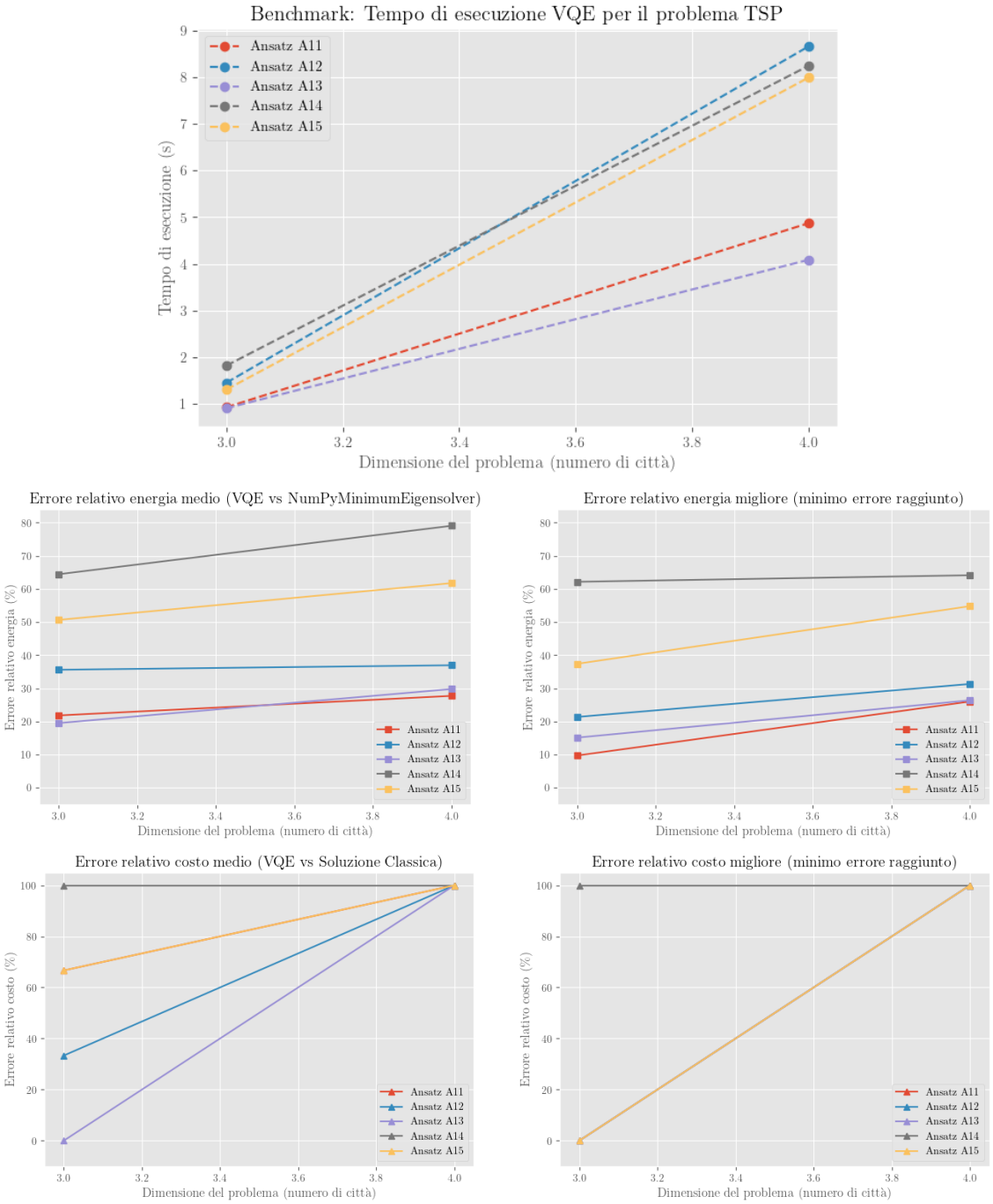

Figura 4.18: Risultati delle simulazioni con COBYLA per il TSP. Da notare che A11 si trova al di sotto di A15 nel grafico in basso a sinistra e nel grafico in basso a destra, tutti - tranne A14 - si trovano al di sotto di A15.

Analisi dei risultati del TSP

I risultati ottenuti sono molto peggiori di quanto avevamo inizialmente immaginato durante la fase di progettazione delle simulazioni. I punti principali che possiamo evidenziare sono:

- **Tempo di esecuzione**: il tempo di esecuzione è molto elevato, salendo in maniera molto veloce già solo confrontando istanze con 3 o 4 nodi. Anche in questi casi, ansatz con più porte richiedono simulazioni più lunghe, con $A15$ che è il peggiore in questo senso.

- **Errore relativo dell'energia minima**: l'errore relativo non è particolarmente basso: i migliori ansatz avevano una media del 25% con SPSA e del 20% con COBYLA. Nei casi migliori, questo errore è arrivato anche sotto al 10%, ma in generale i risultati non sono stati molto buoni e non sono stati costanti.

- **Errore relativo del costo medio**: tenendo in mente che abbiamo assegnato un errore del 100% in caso di violazione dei vincoli, notiamo che abbiamo spesso situazioni di errore molto alto. In particolare, esistono alcune ansatz che non hanno mai trovato una soluzione valida, né per 3 nodi né per 4 nodi, anche partendo da diversi parametri iniziali, ovvero $A14$ con COBYLA. SPSA in generale ha avuto risultati migliori, dove per istanze di 3 nodi tutti gli ansatz hanno trovato almeno una volta una soluzione valida, mentre per 4 nodi solo $A12$ ed $A15$ sono riusciti a trovare una soluzione valida. Con COBYLA invece, nessun ansatz ha trovato una soluzione valida per 4 nodi, con successi comunque limitati rispetto a SPSA per 3 nodi.

  Il problema quindi di non riusciere a convergere ad una soluzione ammissibile affligge più COBYLA che SPSA, ma in generale si tratta di un problema che persiste e che abbassa notevolmente la qualità delle soluzioni trovate.

4.4 Analisi dei risultati delle simulazioni

Le simulazioni hanno mostrato che il VQE è un metodo che può funzionare, ma che ha bisogno di molta cura affinché si ottengano risultati ragionevoli. Vi sono un ampio spettro di scelte che si devono prendere in considerazione nella realizzazione di questo algoritmo ed ognuna di esse ha un impatto notevole sui risultati che otteniamo, in particolare:

- La **scelta dell'ottimizzatore** ci determina i tempi di esecuzione ed ha anche un ruolo piuttosto importante per la qualità della soluzione trovata. SPSA è stato il migliore tra quelli testati, riuscendo velocemente ad ottenere buoni parametri con costanza, permettendoci di eseguire anche misure della soluzione

ottimale, seguito da COBYLA che si è comportato in maniera simile, ma con risultati leggermente peggiori, notevoli soprattutto nel TSP. L-BFGS-B, testato unicamente per KP, è stato considerevolmente più lento degli altri due ed ha avuto i risultati peggiori, quindi - almeno da queste simulazioni - non risulta essere un'opzione valida.

- La **scelta dell'ansatz** è fondamentale per la qualità delle soluzioni. Abbiamo notato che ansatz con un numero ridotto di `reps` sono migliori, in particolare 1 o 2, e che l'entanglement `full` è migliore rispetto a `circular` o `linear`. Abbiamodo determinato sperimentalmente che si tratta di una delle variabili più importanti nella costruzione dell'algoritmo, ma risulta anche essere la più difficile da ottimizzare.

- I **valori iniziali dei parametri** per l'ottimizzatore hanno anch'essi una grande influenza sulle soluzioni che troviamo. Come abbiamo notato però nella Fig. (4.16), in generale la varianza generata da valori iniziali diversi non è molto grande per il valore minimo dell'energia che si riesce a trovare, ma ha una grande influenza sulla soluzione al problema che effettivamente si ottiene, per le quali anche differenze piccole di energia possono portare a soluzioni molto diverse.

Attraverso il nostro approccio piuttosto *naive* della costruzione di ansatz e della scelta di parametri iniziali - impostati casualmente - siamo comunque riusciti ad ottenere dei valori ragionevolmente soddisfacenti quando parliamo dell'energia minima raggiunta dal VQE. Quando comparata con il valore di aspettazione calcolato in modo classico, si tratta solitamente di un errore relativo medio che, con gli ansatz migliori, è circa del 15% per il KP e del 30% per il TSP, e se combinati anche ai migliori parametri iniziali, arriviamo al 5% per il KP e sotto al 10% per il TSP (Fig. (4.15) e (4.17)). Notiamo che questo valore è piuttosto costante anche all'aumentare della dimensione del problema, il che ci suggerisce che risulterebbe possibile scalare questi risultati ad istanze più grandi di questi problemi.

Parlando invece delle soluzioni effettivamente ottenute per i problemi di ottimizzazione considerati, sono sorti problemi profondi che non possiamo tralasciare. Soprattutto nel caso del TSP c'è stata una grande difficoltà nel raggiungere soluzioni valide - cioè che rispettano i vincoli del problema - e anche per il KP le soluzioni trovate presentavano un alto valore per l'errore relativo

del valore massimo, che era in media il 50% anche con gli ansatz migliori. Se i parametri iniziali erano buoni, si è spesso riuscito a convergere alla soluzione ideale, o quasi, ma tendenzialmente questo non è avvenuto con una buona costanza.

Per riassumere pertanto ci è facile notare che nei casi ideali - dove si hanno buoni parametri iniziali, un buon ansatz e un buon ottimizzatore - il VQE è un metodo che può funzionare bene, anche raggiungendo i risultati ottimali associati all'istanza del problema di ottimizzazione che stiamo risolvendo. Tuttavia, questi casi sono piuttosto rari ed i valori che mediamente si ottengono non sono particolarmente promettenti, soprattutto se confrontati con i risultati che si possono ottenere con metodi euristici classici.

Infine, un sostanziale problema nel quale siamo incorsi durante queste simulazioni è stato il tempo dell'esecuzione all'aumentare delle dimensioni dei problemi, soprattutto per il TSP. Ci siamo dovuti limitare a istanze estremamente piccole, con soli 4 nodi per il TSP e 8 oggetti per il KP, per poter fare simulazioni in tempi gestibili e realizzare una quantità sufficiente di risultati per poterne estrarre delle conclusioni. Questo è un problema che si riscontra spesso con l'uso di simulatori quantistici[15], in quanto la complessità di questi problemi cresce in maniera esponenziale con la dimensione del problema e quindi il tempo di esecuzione cresce in maniera simile. L'utilizzo di hardware quantistico aiuterebbe a risolvere, almeno in parte, questo problema, quindi è importante notare che seppure per noi questi tempi computazionali rappresentino una barriera importante, in situazioni di utilizzo reale di questi metodi le tempistiche sarebbero più brevi.

4.4.1 Confronto con algoritmi classici

In vista dell'obiettivo iniziale di questa tesi, ovvero quello di riuscire a comparare i risultati ottenuti con il VQE con quelli che si possono ottenere con l'utilizzo di computer classici, dobbiamo tristemente arrenderci di fronte agli importanti limiti riscontrati. Non è ragionevole comparare le soluzioni da noi ottenute in media con l'uso del VQE con gli algoritmi euristici che avevamo introdotto nel Cap. 2, in quanto l'approccio greedy per la soluzione del KP offre soluzioni che mediamente hanno errori relativi che non vanno oltre al 5% nelle istanze di piccole dimensioni, come trovato sperimentalmente dalle prove

mostrate nella Fig. (2.2), mentre per il TSP la soluzione ottenuta usando LKH ha un errore relativo che si trova tra l'1% ed il 2%[26]. Entrambi questi algoritmi quindi riescono ad ottenre soluzioni che sono estremamente migliori di quelle ottenute con il VQE in termini di errore relativo del valore ottimo trovato, che ricordiamo aver riscontrato mediamente essere del 50%, ma con tempi di esecuzione molto più brevi. In casi in cui i parametri collaborino per una soluzione ottimale, dati i nostri ansatz nelle nostre sperimentazioni, il VQE riesce potenzialmente a raggiungere risultati migliori, però si tratta di casi decisamente non frequenti, pertanto non possiamo considerare il VQE così come da noi implementato come un candidato che possa competere con gli algoritmi classici per questi problemi di ottimizzazione.

Conclusioni

Il Variational Quantum Eigensolver è un algoritmo estremamente interessante e si pone come punto di aggancio tra il mondo classico e quello quantistico, permettendo di iniziare a sfruttare le potenzialità dei computer quantistici attuali per risolvere problemi di ottimizzazione. Tra le mani però non abbiamo un algoritmo perfetto, ma uno che ha tanti lati positivi quanto negativi, come abbiamo potuto constatare durante il nostro lavoro. Si tratta di un ecosistema complesso, dove allo sviluppatore sono delegate molte scelte con importanti impatti sulle soluzioni che si ottengono.

L'obiettivo principale che ci eravamo posti era quello di riuscire ad inquadrare, all'interno del panorama odierno degli algoritmi utilizzati per la soluzione dei problemi di ottimizzazione, il VQE, cercando di capire la sua efficacia quando messo fianco a fianco con algoritmi euristici classici. Il risultato è stato costatare che siamo, almeno all'interno del nostro ambiente di simulazione classica, ancora lontani dal riuscire a trarre vantaggi pratici dall'utilizzo di questo algoritmo in casi reali. Nondimeno, il VQE rimane promettente, in quanto anche con le nostre limitazioni siamo riusciti a migliorare di non poco i risultati ottenuti per le soluzioni del KP, ed oltre a ciò abbiamo avuto diverse istanze di soluzioni che combaciavano con le soluzioni esatte, pertanto c'è un forte istinto che mi motiva a sostenere che con sufficiente lavoro per migliorare l'implementazione si possano ottenere dei risultati decisamente migliori.

Oltre a questo, è importante notare che questo algoritmo si dovrebbe porre come risolutore di problemi di grandi dimensioni e dovrebbe utilizzare vero hardware quantistico, due caratteristiche che nelle nostre simulazioni non ci è stato possibile neanche considerare. Anche questa sfaccettatura ci permette di giustificare le sue prestazioni subottimali e di mantenere ancora un certo livello di ottimismo per il futuro, in quanto le possibilità di successo sono ancora da esplorare in maniera più approfondita.

A partire da questa tesi ci sono un'infinità di strade percorribili per continuare a lavorare su questo algoritmo e, più ampiamente, su questo arogmento. Nello specifico, sarebbe interessante continuare a fare prove e sperimentazioni con la creazione delle ansatz, cercando di realizzare analisi più complete e complesse per categorizzarle e delineare in modo chiaro quali siano le scelte migliori da fare. Usare metodi intelligenti per la selezione dei parametri iniziali è un altro ambito che potrebbe portare piuttosto velocemente a risultati migliori, così come l'utilizzo di formulazioni QUBO leggermente diverse per i nostri problemi di ottimizzazione. Più ampiamente invece implementare altri algoritmi quantistici per la risoluzione di problemi di ottimizzazione, come il Quantum Approximate Optimization Algorithm (QAOA) e le sue varianti, potrebbe portarci ad avere intuizioni migliori su come lavorare con questi problemi.

Insomma, se da un lato possiamo affermare che il VQE non è ancora pronto per essere utilizzato in maniera competitiva rispetto ad altri algoritmi per la risoluzione di problemi di ottimizzazione, dall'altro esso ha un grande potenziale e tanti aspetti che lo rendono interessante e ne motivano lo studio e la ricerca. Questa tesi si pone quindi come un punto di partenza per ulteriori sviluppi in un ambito in continua evoluzione e che si avvicina sempre più ad una realtà tangibile e con usi pratici, dove algoritmi come il VQE saranno potenzialmente centrali nella dimostrazione dei punti di forza dei computer quantistici.

Bibliografia

[1] J. Powell, "The Quantum Limit to Moore's Law," *Proceedings of the IEEE*, vol. 96, pp. 1247–1248, set. 2008. DOI: 10.1109/JPROC.2008.925411.

[2] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, mag. 1980. DOI: 10.1007/BF01011339.

[3] P. Benioff, "Quantum mechanical Hamiltonian models of Turing machines," *Journal of Statistical Physics*, vol. 29, pp. 515–546, nov. 1982. DOI: 10.1007/BF01342185.

[4] P. Benioff, "Quantum mechanical Hamiltonian models of discrete processes that erase their own histories: Application to Turing machines," *International Journal of Theoretical Physics*, vol. 21, pp. 177–201, gen. 1982. DOI: 10.1007/BF01857725.

[5] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, n. 6, pp. 467–488, 1982, ISSN: 1572-9575. DOI: 10.1007/BF02650179. indirizzo: https://doi.org/10.1007/BF02650179.

[6] R. P. Poplavskiĭ, "Thermodynamic models of information processes," *Physics-Uspekhi*, vol. 18, pp. 222–241, 1975. indirizzo: https://api.semanticscholar.org/CorpusID:124340318.

[7] D. Deutsch e R. Penrose, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, n. 1818, pp. 97–117, 1985. DOI: 10.1098/rspa.1985.0070.

[8] M. A. Nielsen e I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[9] J. S. Bell, "On the Einstein Podolsky Rosen paradox," *Physics Physique Fizika*, vol. 1, pp. 195–200, 3 nov. 1964. DOI: 10.1103/PhysicsPhysiqueFizika.1.195. indirizzo: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195.

[10] A. Wu, G. Li, Y. Wang, B. Feng, Y. Ding e Y. Xie, *Towards Efficient Ansatz Architecture for Variational Quantum Algorithms*, 2021. arXiv: 2111.13730 [quant-ph]. indirizzo: https://arxiv.org/abs/2111.13730.

[11] C. P. Williams, *Explorations in Quantum Computing* (Texts in Computer Science). Springer, 2011. DOI: 10.1007/978-1-84628-887-6. indirizzo: https://doi.org/10.1007/978-1-84628-887-6.

[12] Google Quantum AI and Collaborators, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, pp. 920–926, 2024. DOI: 10.1038/s41586-024-08449-y. indirizzo: https://doi.org/10.1038/s41586-024-08449-y.

[13] S. McDowell, "IBM Advances Quantum Computing with New Processors & Platforms," *Forbes*, dic. 2023. indirizzo: https://www.forbes.com/sites/stevemcdowell/2023/12/05/ibm-advances-quantum-computing-with-new-processors--platforms/.

[14] L. Tremolada, "Microsoft svela Majorana 1, il primo chip quantistico con qubit topologici," *Il Sole 24 Ore*, feb. 2025. indirizzo: https://www.ilsole24ore.com/art/microsoft-svela-majorana-1-primo-chip-quantistico-qubit-topologici-AGRsL6zC.

[15] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, ago. 2018, ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. indirizzo: http://dx.doi.org/10.22331/q-2018-08-06-79.

[16] E. Gibney, "Hello quantum world! Google publishes landmark quantum supremacy claim," *Nature*, vol. 574, n. 7779, pp. 461–462, 2019. DOI: 10.1038/d41586-019-03213-z. indirizzo: https://www.nature.com/articles/d41586-019-03213-z.

[17] eeNews Europe, "Atom shows record 24 logical qubit quantum computer," *eeNews Europe*, nov. 2024. indirizzo: https://www.eenewseurope.com/en/atom-shows-record-24-logical-qubit-quantum-computer/.

[18] R. Letzter, *Landmark IBM error correction paper published on the cover of Nature*, mar. 2024. indirizzo: https://www.ibm.com/quantum/blog/nature-qldpc-error-correction.

[19] Fujitsu, "Unraveling the Latest Trends in Error Correction and Error Mitigation in Quantum Computers," *Fujitsu Technology News*, mag. 2024. indirizzo: https://global.fujitsu/en-global/technology/key-technologies/news/ta-fault-tolerant-quantum-computation-20240515.

[20] S. Boyd e L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004, ISBN: 9780521833783. indirizzo: https://stanford.edu/~boyd/cvxbook/.

[21] M. Sipser, *Introduction to the Theory of Computation*, 3rd. Boston, MA: Cengage Learning, 2013, ISBN: 9781133187790. indirizzo: https://faculty.cengage.com/works/9781133187790.

[22] F. Klug, "Quantum Algorithms for Solving the Traveling Salesman Problem," *SSRN Electronic Journal*, 2024. DOI: 10.2139/ssrn.4836033. indirizzo: https://ssrn.com/abstract=4836033.

[23] W. J. Cook, *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2012. indirizzo: http://www.jstor.org/stable/j.ctt7t8kc.

[24] S. Martello e P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990, ISBN: 0-471-92420-2.

[25] D. L. Applegate, R. E. Bixby, V. Chvátal e W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton University Press, 2006, ISBN: 978-0-691-12993-8.

[26] R. Simha, *The Traveling Salesman Problem*, Parte della serie "Championship Algorithms" del Professor Simha, che esplora classici problemi di ottimizzazione. indirizzo: https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html.

[27] S. Lin e B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, vol. 21, n. 2, pp. 498–516, 1973. indirizzo: https://www.jstor.org/stable/169020.

[28] T. Narwadi e Subiyanto, "An Application of Traveling Salesman Problem Using the Improved Genetic Algorithm on Android Google Maps," in *5th International Conference on Education, Concept, and Application of Green Technology*, ser. AIP Conference Proceedings, vol. 1818, AIP Publishing, 2017, p. 020 035. DOI: 10.1063/1.4976899. indirizzo: https://doi.org/10.1063/1.4976899.

[29] D. Chan e D. Mercier, "IC Insertion: An Application of the Travelling Salesman Problem," *International Journal of Production Research*, vol. 27, n. 10, pp. 1837–1841, 1989. DOI: 10.1080/00207548908942657. indirizzo: https://doi.org/10.1080/00207548908942657.

[30] H. Kellerer, U. Pferschy e D. Pisinger, *Knapsack Problems*. Springer, 2004. DOI: 10.1007/978-3-540-24777-7. indirizzo: https://doi.org/10.1007/978-3-540-24777-7.

[31] A. Peruzzo, J. McClean, P. Shadbolt et al., "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, n. 1, lug. 2014, ISSN: 2041-1723. DOI: 10.1038/ncomms5213. indirizzo: http://dx.doi.org/10.1038/ncomms5213.

[32] A. Kandala, A. Mezzacapo, K. Temme et al., "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, pp. 242–246, 2017. DOI: 10.1038/nature23879. indirizzo: https://doi.org/10.1038/nature23879.

[33] E. Farhi, J. Goldstone e S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv preprint arXiv:1411.4028*, 2014. arXiv: 1411.4028.

[34] M. Stęchły, *Introduction to Variational Quantum Algorithms*, 2024. arXiv: 2402.15879 [quant-ph]. indirizzo: https://arxiv.org/abs/2402.15879.

[35] L. D. Landau e E. M. Lifshitz, *Quantum Mechanics: Non-Relativistic Theory* (Course of Theoretical Physics), 3rd. Oxford: Butterworth-Heinemann, 1991, vol. 3, p. 58.

[36] J. Tilly, H. Chen, S. Cao et al., "The Variational Quantum Eigensolver: A review of methods and best practices," *Physics Reports*, vol. 986, pp. 1–128, nov. 2022, ISSN: 0370-1573. DOI: 10.1016/j.physrep.2022.08.003. indirizzo: http://dx.doi.org/10.1016/j.physrep.2022.08.003.

[37] A. Anand, P. Schleich, S. Alperin-Lea et al., "A quantum computing view on unitary coupled cluster theory," *Chemical Society Reviews*, vol. 51, n. 5, pp. 1659–1684, 2022, ISSN: 1460-4744. DOI: 10.1039/d1cs00932j. indirizzo: http://dx.doi.org/10.1039/D1CS00932J.

[38] P. K. Barkoutsos, J. F. Gonthier, I. Sokolov et al., "Quantum algorithms for electronic structure calculations: Particle-hole Hamiltonian and optimized wave-function expansions," *Physical Review A*, vol. 98, n. 2, ago. 2018, ISSN: 2469-9934. DOI: 10.1103/physreva.98.022322. indirizzo: http://dx.doi.org/10.1103/PhysRevA.98.022322.

[39] E. Farhi, J. Goldstone e S. Gutmann, *A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem*, 2015. arXiv: 1412.6062 [quant-ph]. indirizzo: https://arxiv.org/abs/1412.6062.

[40] K. Blekos, D. Brand, A. Ceschini et al., "A review on Quantum Approximate Optimization Algorithm and its variants," *Physics Reports*, vol. 1068, pp. 1–66, 2024, A review on Quantum Approximate Optimization Algorithm and its variants, ISSN: 0370-1573. DOI: https://doi.org/10.1016/j.physrep.2024.03.002. indirizzo: https://www.sciencedirect.com/science/article/pii/S0370157324001078.

[41] N. V. Tkachenko, J. Sud, Y. Zhang et al., "Correlation-Informed Permutation of Qubits for Reducing Ansatz Depth in the Variational Quantum Eigensolver," *PRX Quantum*, vol. 2, n. 2, p. 020 337, 2021. DOI: 10.1103/PRXQuantum.2.020337. indirizzo: https://doi.org/10.1103/PRXQuantum.2.020337.

[42] H. L. Tang, V. O. Shkolnikov, G. S. Barron, H. R. Grimsley, S. E. Economou e E. Barnes, "qubit-ADAPT-VQE: An Adaptive Algorithm for Constructing Hardware-Efficient Ansätze on a Quantum Processor," *PRX Quantum*, vol. 2, n. 2, p. 020 310, 2021. DOI: 10.1103/PRXQuantum.2.020310. indirizzo: https://doi.org/10.1103/PRXQuantum.2.020310.

[43] Y. Shen, "Prepare Ansatz for VQE with Diffusion Model," *arXiv preprint arXiv:2310.02511*, 2023. indirizzo: https://arxiv.org/abs/2310.02511.

[44] Q. C. Inc., *Ansatz*, https://www.quera.com/glossary/ansatz. (visitato il 16/03/2025).

[45] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, 2014, ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. indirizzo: http://dx.doi.org/10.3389/fphy.2014.00005.

[46] M. Schnaus, L. Palackal, B. Poggel et al., "Efficient Encodings of the Travelling Salesperson Problem for Variational Quantum Algorithms," in *2024 IEEE International Conference on Quantum Software (QSW)*, IEEE, lug. 2024, pp. 81–87. DOI: 10.1109/qsw62656.2024.00022. indirizzo: http://dx.doi.org/10.1109/QSW62656.2024.00022.

[47] D. Egger, *Solving Optimization Problems with Quantum Algorithms with Daniel Egger: Qiskit Summer School 2024*, Qiskit, 2024. indirizzo: https://www.youtube.com/watch?v=RyRKmZ6A25k (visitato il 16/03/2025).

[48] A. Montanez, *PennyLane Quadratic Unconstrained Binary Optimization (QUBO) tutorial*, https://pennylane.ai/qml/demos/tutorial_QUBO. (visitato il 16/03/2025).

[49] Wikipedia contributors. "Eigenvalue algorithm — Wikipedia, The Free Encyclopedia." (2025), indirizzo: https://en.wikipedia.org/wiki/Eigenvalue_algorithm (visitato il 16/03/2025).

[50] Q. C. Inc., *Quantum Phase Estimation*, https://www.quera.com/glossary/quantum-phase-estimation. (visitato il 16/03/2025).

[51] Q. D. Team. "Optimizers (qiskit_algorithms.optimizers) - Qiskit Algorithms 0.3.1." (2024), indirizzo: https://qiskit-community.github.io/qiskit-algorithms/apidocs/qiskit_algorithms.optimizers.html (visitato il 16/03/2025).

[52] IBM Quantum. "Qiskit | IBM Quantum Computing." (2025), indirizzo: https://www.ibm.com/quantum/qiskit (visitato il 16/03/2025).

[53] IBM Quantum. "IBM Quantum Documentation." (2025), indirizzo: https://docs.quantum.ibm.com/ (visitato il 16/03/2025).

[54] Qiskit Development Team. "Qiskit Aer 0.16.1 Documentation." (2025), indirizzo: https://qiskit.github.io/qiskit-aer/index.html (visitato il 16/03/2025).

[55] IBM Decision Optimization. "IBM Decision Optimization CPLEX Modeling for Python (DOcplex) V2.25 Documentation." (2025), indirizzo: https://ibmdecisionoptimization.github.io/docplex-doc/ (visitato il 16/03/2025).

[56] Qiskit Community. "Qiskit Optimization 0.6.1 Documentation." (2025), indirizzo: https://qiskit-community.github.io/qiskit-optimization/index.html (visitato il 16/03/2025).

[57] Qiskit Community. "Qiskit Algorithms 0.3.1 Documentation." (2025), indirizzo: `https://qiskit-community.github.io/qiskit-algorithms/index.html` (visitato il 16/03/2025).

[58] IBM Quantum. "TwoLocal Class Documentation." (2025), indirizzo: `https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.TwoLocal` (visitato il 16/03/2025).

Ringraziamenti

Devo questa tesi a tantissime persone che mi sono state affianco durante i miei anni di studio a Bologna. Sono un po' impacciato e non so se questo sia lo spazio giusto dove aprirmi tanto quanto mi piacerebbe - o quanto penso se lo meriterebbero - nel fare complimenti, ma le persone davvero importanti per me che ho conosciuto grazie al corso di studio, grazie allo studentato, grazie ai concerti, grazie al lavoro e grazie alle ridicole uscite organizzate all'ultimo minuto e, per farla breve, grazie a questa città e questa università, sono state fantastiche e senza di loro non sarei mai riuscito a concludere questo percorso con così tante soddisfazioni.

Vorrei anche ringraziare la mia famiglia, senza la quale nulla di questo sarebbe stato possibile. Mi sento estremamente fortunato ad avere dei genitori che mi hanno sempre supportato nelle mie idee e nei miei progetti, sia dandomi una mano quando ne avevo bisogno che motivandomi a dare sempre il massimo.

A tutti voi, vi voglio bene e vi ringrazio di cuore.

Un ringraziamento sentito va anche ai miei relatori, la Prof.ssa Elisa Ercolessi ed il Prof. Ugo Dal Lago, per avermi seguito e supportato durante la stesura di questa tesi. Sono stati disponibili e pazienti e mi hanno istradato verso la giusta direzione quando ne avevo bisogno, lasciandomi però anche la libertà di esplorare e sperimentare, riponendo fiducia nelle mie capacità.