Quantum Portfolio Optimization

Colli Simone[1] and Merenda Saverio Mattia[2]

[1] simone.colli@studenti.unipr.it
[2] saveriomattia.merenda@studenti.unipr.it

December 29, 2024

Abstract

In questo elaborato esploriamo l'applicazione del calcolo quantistico all'ottimizzazione del portafoglio in ambito finanziario, confrontando i metodi classici con gli approcci quantistici basati su VQE (Variational Quantum Eigensolver) e QAOA (Quantum Approximate Optimization Algorithm). Formuliamo il problema dell'ottimizzazione del portafoglio come un problema QUBO (Quadratic Unconstrained Binary Optimization) e lo implementiamo utilizzando il framework Qiskit. Lo studio include simulazioni sia in assenza che in presenza di rumore per valutare le prestazioni degli algoritmi in condizioni realistiche. La ricerca evidenzia le attuali limitazioni nel scalare l'ottimizzazione quantistica del portafoglio alle applicazioni del mondo reale, principalmente a causa dei vincoli hardware e dell'impatto del rumore su sistemi più grandi.

1 Introduzione

L'ottimizzazione del portafoglio (PO) è un'attività finanziaria di primaria importanza, con applicazioni significative in diversi contesti, come i fondi di investimento, i piani pensionistici e altre strategie di allocazione del capitale. Data una disponibilità di budget e/o un insieme di asset, l'obiettivo è individuare operazioni ottimali all'interno di un mercato che può includere un numero elevato di asset. La corretta allocazione degli asset ha un impatto diretto sulla redditività degli investimenti, consentendo di ottenere rendimenti più elevati e una migliore gestione del rischio. Data la rilevanza economica del problema, l'ottimizzazione del portafoglio rappresenta un'area strategica sia per le istituzioni finanziarie sia per gli investitori.

Limiti dei metodi classici I metodi tradizionali, come gli approcci geometrici o gli algoritmi euristici, presentano significative limitazioni, soprattutto in termini di scalabilità ed efficienza. Con l'aumentare della complessità e delle dimensioni del mercato, la risoluzione del problema diventa rapidamente intrattabile per i computer classici (CPU). Ad esempio, algoritmi come il branch-and-bound (Land and Doig, 2010), utilizzati per trovare soluzioni esatte, faticano a gestire mercati con un numero elevato di asset.

Quantum computing L'introduzione del calcolo quantistico apre nuove possibilità per affrontare i limiti dei metodi classici. Sfruttando i principi della meccanica quantistica, come la sovrapposizione e l'entanglement, i computer quantistici (QPU) promettono di risolvere problemi di ottimizzazione in modo più efficiente. In particolare, i problemi di ottimizzazione quadratica, come quello del portafoglio, possono beneficiare di algoritmi quantistici in grado di trovare soluzioni quasi ottimali in tempi significativamente ridotti rispetto ai metodi tradizionali.

Computer classici vs computer quantistici Le CPU elaborano le informazioni utilizzando i bit, che possono assumere esclusivamente due stati, 0 o 1. Questa caratteristica limita la capacità di esplorare lo spazio delle soluzioni in parallelo, costringendo i calcoli a procedere in modo sequenziale o attraverso tecniche di parallelismo limitate.

Al contrario, i QPU sfruttano i qubit, che possono trovarsi in una sovrapposizione di stati, rappresentando simultaneamente sia 0 che 1. Grazie a questa proprietà unica, i computer quantistici sono in grado di eseguire calcoli in parallelo, esplorando uno spazio di soluzioni molto più vasto rispetto ai computer classici e rendendoli particolarmente adatti per affrontare problemi complessi come quelli di ottimizzazione.

2 Costruzione del problema

Per lo svolgimento di questo progetto, sono stati analizzati dataset di dimensioni contenute, selezionando un massimo di n asset distinti. Per ciascun asset i, con $1 \leq i \leq n$, è stato considerato l'intervallo temporale tra il 01/01/2016 e il 01/01/2020. Per ogni giorno t in questo intervallo ($0 \leq t \leq T$), la performance di un asset è rappresentata dal suo prezzo di chiusura p_i^t.

La prima informazione estratta da questo dataset consiste nell'elenco P dei prezzi correnti P_i degli asset considerati:

$$P_i = p_i^t. \tag{1}$$

Inoltre, per ciascun asset, il rendimento r_i^t tra i giorni $t-1$ e t può essere calcolato come:

$$r_i^t = \frac{p_i^t - p_i^{t-1}}{p_i^{t-1}}. \tag{2}$$

Grazie a questi rendimenti, è possibile definire il rendimento atteso di un asset come una stima ragionata della sua futura performance. Supponendo una distribuzione normale dei rendimenti, la media dei loro valori in ogni momento t nel set di osservazioni storiche è un buon stimatore del rendimento atteso. Pertanto, dato l'intero dataset storico, il rendimento atteso di ciascun asset μ_i è calcolato come:

$$\mu_i = E[r_i] = \frac{1}{T}\sum_{t=1}^{T} r_i^t. \tag{3}$$

Seguendo lo stesso principio, la varianza del rendimento di ciascun asset, σ_{ij}, e la covarianza tra i rendimenti di asset differenti nel corso delle serie storiche, σ_i^2, possono essere calcolate come segue:

$$\sigma_{ij} = E[(r_i - \mu_i)(r_j - \mu_j)] = \frac{1}{T-1}\sum_{t=1}^{T}(r_i^t - \mu_i)(r_j^t - \mu_j), \tag{4}$$

$$\sigma_i^2 = E[(r_i - \mu_i)^2] = \frac{1}{T-1}\sum_{t=1}^{T}(r_i^t - \mu_i)^2. \tag{5}$$

Un portafoglio è definito come un insieme di investimenti x_i (misurati come frazione del budget o del numero di unità allocate) per ciascun asset i del mercato. Pertanto, il portafoglio è composto da un vettore di numeri reali con dimensioni pari al numero di asset considerati.

Una strategia ottimale di allocazione del portafoglio punta a **massimizzare** il rendimento del portafoglio $\mu^\top x$ **minimizzando** il rischio, definito come la varianza del portafoglio $x^\top \Sigma x$, la cui radice quadrata rappresenta la volatilità del portafoglio. In questo caso, μ è il vettore dei rendimenti medi per ciascun asset i calcolato con la Formula 3,

$$\mu = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_n \end{bmatrix}, \tag{6}$$

Σ è la matrice di covarianza calcolata con le Formule 4 e 5,

$$\Sigma = \begin{bmatrix} \sigma_0^2 & \sigma_{10} & \cdots & \sigma_{n0} \\ \sigma_{01} & \sigma_1^2 & \cdots & \sigma_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{0n} & \sigma_{1n} & \cdots & \sigma_n^2 \end{bmatrix}, \tag{7}$$

e x è il vettore delle frazioni di budget allocate per ciascun asset.

L'obiettivo di trovare un portafoglio ottimale consiste quindi nel trovare il vettore x che minimizza la funzione obiettivo seguente:

$$\mathcal{L}(x) = qx^\top \Sigma x - \mu^\top x, \tag{8}$$

dove il parametro di avversione al rischio q esprime la propensione dell'investitore al rischio, i.e., un compromesso tra rischio e rendimento.

In uno scenario realistico, il budget disponibile B è fisso. Pertanto, il vincolo secondo cui la somma degli x_i deve essere pari a 1 è valido, e può essere espresso nel seguente modo:

$$B = \sum_{i=1}^{n} x_i = 1. \tag{9}$$

Di conseguenza, il problema può essere espresso come segue:

$$\min_x (qx^\top \Sigma x - \mu^\top x), \tag{10}$$

dove:

- $x \in \{0,1\}^n$ denota il vettore delle variabili decisionali binarie, che indicano quali asset selezionare e quali no, identificati con $x_i = 1$ e $x_i = 0$, rispettivamente;
- $\mu \in \mathbb{R}^n$ definisce i rendimenti attesi degli asset;
- $\Sigma \in \mathbb{R}^{n \times n}$ specifica le covarianze tra gli asset;
- $q > 0$ controlla l'avversione al rischio del decisore;
- B denota il budget, ovvero il numero di asset da selezionare tra gli n disponibili.

Per poter risolvere il problema mediante algoritmi quantistici, è necessario formularlo senza vincoli espliciti. Per questo motivo, introduciamo un termine di penalità che favorisce le soluzioni in cui il numero di asset selezionati, i.e., il numero di 1 nel vettore x, sia il più vicino possibile al budget B.

Il problema di ottimizzazione risulta quindi:

$$\min_x \left(qx^T \Sigma x - x\mu^T + (1^T x - B)^2 \right). \tag{11}$$

Questa formulazione rappresenta un Quadratic Unconstrained Binary Optimization problem (QUBO), che può essere risolto utilizzando algoritmi di ottimizzazione quantistica basati sul principio variazionale, come il Variational Quantum Eigensolver (VQE) e il Quantum Approximate Optimization Algorithm (QAOA), i quali verranno approfonditi nelle Sezioni 3.1 e 3.2, rispettivamente.

Perchè usare il Quantum Computing Nel contesto dell'ottimizzazione di portafoglio, l'analisi della complessità computazionale rivela differenze significative tra l'approccio classico e quello quantistico. Nel caso classico, il problema ricade nella classe dei problemi NP-hard, dove lo spazio delle soluzioni cresce esponenzialmente: per n variabili binarie, abbiamo 2^n possibili soluzioni, mentre per x variabili intere che variano da 0 a $x_{\max}$, lo spazio delle soluzioni diventa $(x_{\max}+1)^x$. Per gestire questa complessità, sono stati sviluppati metodi euristici che però mostrano limitazioni pratiche, risultando efficaci solo per portafogli con pochi asset.

L'approccio quantistico, invece, sfrutta fenomeni quantistici fondamentali come l'interferenza e l'entanglement per eseguire computazioni all'interno della classe di complessità BQP (Bounded-error Quantum Polynomial). Questa classe di problemi richiede un tempo polinomiale per la risoluzione su un computer quantistico, ritornando una soluzione corretta con probabilità maggiore o uguale a $\frac{2}{3}$ (Buonaiuto *et al.*, 2023).

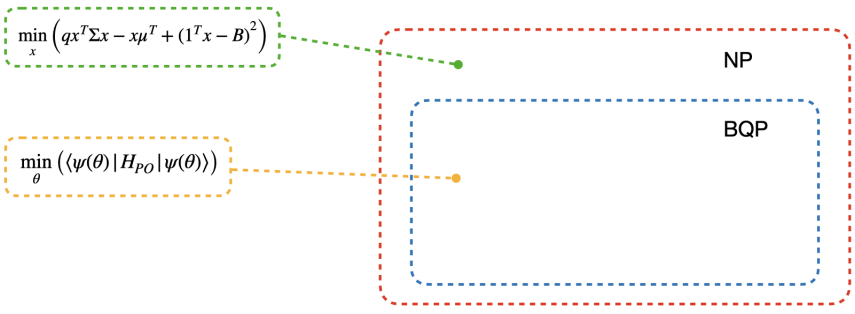


Figure 1: Gerarchia delle classi di complessità computazionale.

3 Quadratic Unconstrained Binary Optimization

I problemi Quadratic Unconstrained Binary Optimization (QUBO) rappresentano una classe fondamentale di problemi di ottimizzazione in cui si cerca di minimizzare (o massimizzare) una funzione quadratica con variabili binarie, cioè variabili che possono assumere solo i valori 0 o 1. Ciò che rende i QUBO particolarmente interessanti è il fatto che molti problemi complessi di ottimizzazione possono essere riformulati in questo formato.

La caratteristica principale dei QUBO è la loro semplicità strutturale: sono problemi non vincolati, il che significa che l'unico vincolo è la natura binaria delle variabili. Nonostante questa apparente semplicità, i QUBO sono problemi NP-completi, i.e., sono computazionalmente difficili da risolvere con metodi classici quando la dimensione del problema cresce.

L'importanza dei QUBO nel contesto del calcolo quantistico deriva dalla loro versatilità. Infatti, possiamo utilizzarli per risolvere una vasta gamma di problemi pratici come:

- la colorazione dei grafi, dove si cerca di assegnare colori a nodi di un grafo in modo che nodi adiacenti abbiano colori diversi;
- il partizionamento di numeri, dove si cerca di dividere un insieme di numeri in gruppi con somma simile;
- l'ottimizzazione di portafoglio finanziario, dove si cerca di bilanciare rischio e rendimento;

Nel contesto degli algoritmi quantistici come il VQE e il QAOA, i problemi QUBO sono particolarmente adatti perché la loro struttura si traduce naturalmente in termini di interazioni tra qubit. Questa caratteristica li rende ideali per l'implementazione su QPU, dove le interazioni tra qubit possono essere controllate e manipolate per trovare soluzioni ottimali.

3.1 Variational Quantum Eigensolver

Il Variational Quantum Eigensolver (VQE) è un algoritmo ibrido che combina l'uso di computer quantistici e classici per risolvere problemi di ottimizzazione. Il suo funzionamento si basa sul principio variazionale e mira a trovare lo stato di energia minima di un sistema quantistico.

L'idea principale è quella di parametrizzare un circuito quantistico attraverso un insieme di parametri θ e utilizzare questi parametri per minimizzare l'energia del sistema, definita come:

$$E(\theta) = \frac{\langle\psi(\theta)|H|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle}, \tag{12}$$

dove H è l'Hamiltoniano che descrive il nostro problema di ottimizzazione e $\psi(\theta)$ è la funzione d'onda parametrizzata. In particolare, lo stato fondamentale E_0 corrisponde allo stato fondamentale (*ground state*) di energia minima:

$$E_0 \leq \frac{\langle\psi(\theta)|H|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle}. \tag{13}$$

Quindi, il compito del VQE è trovare l'insieme ottimale di parametri, tale che l'energia associata allo stato sia quasi indistinguibile dal suo stato fondamentale, cioè trovare l'insieme di parametri θ, corrispondente all'energia $E_{\min}$, per il quale $|E_{\min} - E_0| < \epsilon$, dove ϵ è una costante arbitrariamente piccola. Questo problema può essere formulato su un QPU come una serie di gates che vengono applicati allo stato iniziale per realizzare un ansatz strutturato per il problema Hamiltoniano.

Convenzionalmente, lo stato iniziale è impostato per essere lo stato di vuoto, i.e., per un sistema di N qubit $|0\rangle^{\otimes N} = |0\rangle$. Quindi, il problema di minimizzare l'energia del sistema (Equazione 12) può essere espresso come:

$$E_{\min} = \min_{\theta} \langle 0|U^\dagger(\theta) H U(\theta)|0\rangle, \tag{14}$$

dove $U(\theta)$ è l'operatore unitario parametrizzato che fornisce la funzione d'onda ansatz quando applicato allo stato iniziale, e $E_{\min}$ è l'energia associata all'ansatz parametrizzato.

Per essere efficientemente implementato in un circuito quantistico, è importante che H venga espresso nella forma:

$$H = \sum_l c_l P_l = \sum_l c_l \otimes \sigma_j^i, \tag{15}$$

dove P_l sono stringhe di Pauli rappresentate dal prodotto tensore degli operatori di Pauli $\sigma_j^i \in \{I, \sigma_X, \sigma_Y, \sigma_Z\}$, con j che denota il qubit su cui l'operatore agisce e i il termine dell'Hamiltoniano. I coefficienti c_l sono pesi complessi, adeguati per definire l'Hamiltoniano specifico del problema. In questa rappresentazione, l'Hamiltoniano è espresso come una combinazione lineare di stringhe di Pauli.

Quindi, l'obiettivo del VQE è risolvere il seguente problema di ottimizzazione:

$$E_{\min} = \min_{\theta} \sum_{l} c_l \langle 0|U^\dagger(\theta) P_l U(\theta)|0\rangle. \tag{16}$$

In questo contesto, si cerca lo stato $|\psi(\theta)\rangle$ che corrisponde allo stato fondamentale di H.

Il calcolo del valore atteso di una stringa di Pauli P_l è essenziale. Questo valore è ottenuto misurando ogni qubit coinvolto nella stringa P_l, operando nella base di misura adeguata. Ad esempio, per uno stato generico del tipo $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, il valore atteso sull'operatore di Pauli σ_Z è dato da:

$$\langle \psi|\sigma_Z|\psi\rangle = |\alpha|^2 - |\beta|^2. \tag{17}$$

Questo valore rappresenta la probabilità di osservare lo stato $|0\rangle$ meno la probabilità di osservare lo stato $|1\rangle$. Misure relative a σ_X o σ_Y richiedono una rotazione preliminare nella base di misura σ_Z.

Dato che i risultati delle misurazioni quantistiche sono binari, è necessario ripetere l'esperimento più volte per approssimare al meglio il valore medio di ogni termine. Questo processo è ripetuto separatamente per ogni stringa P_l che compone l'Hamiltoniano.

L'approccio VQE mira a bilanciare la complessità del circuito quantistico con il numero di misurazioni richieste, permettendo un'ottimizzazione efficiente degli stati parametrizzati. Il processo di ottimizzazione avviene in modo iterativo. Inizialmente, il QPU prepara uno stato quantistico utilizzando i parametri correnti. Successivamente, viene misurata l'energia di questo stato preparato. Sulla base di questa misura, un ottimizzatore classico aggiorna i parametri nel tentativo di minimizzare l'energia del sistema. Questo ciclo di preparazione, misurazione e aggiornamento viene ripetuto fino a raggiungere la convergenza, ovvero fino a quando l'energia non può essere ulteriormente minimizzata in modo significativo (Figura 2).

Questo approccio ibrido sfrutta i punti di forza di entrambe le tipologie di computer: il QPU si occupa della preparazione e della misurazione degli stati quantistici, mentre la CPU gestisce l'ottimizzazione dei parametri. Nel contesto della PO, il VQE viene impiegato per trovare la configurazione ottimale degli asset che minimizza una funzione obiettivo, tenendo conto sia del rischio che del rendimento atteso.

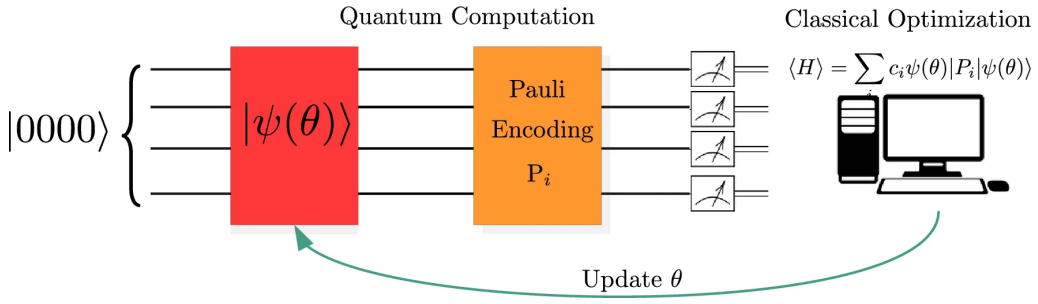


Figure 2: Schema del funzionamento del VQE (Buonaiuto *et al.*, 2023).

3.2 Quantum Approximate Optimization Algorithm

Il Quantum Approximate Optimization Algorithm (QAOA) è un algoritmo variazionale quantistico progettato per trovare soluzioni approssimate a problemi di ottimizzazione combinatoria. Il suo funzionamento si basa sull'alternanza di due operatori: un operatore di *costo* e un operatore di *mixing*, il quale ha il compito di esplorare lo spazio delle soluzioni.

L'algoritmo opera attraverso una sequenza di strati (*layers*) quantistici, dove ogni strato è composto da due parti principali:

$$\hat{U}_C(\gamma) = e^{-i\gamma \hat{H}_C} \quad \text{(operatore di costo)}, \tag{18}$$

$$\hat{U}_M(\beta) = e^{-i\beta \hat{H}_M} \quad \text{(operatore di mixing)}, \tag{19}$$

dove $\hat{H}_C$ è l'Hamiltoniano di costo che codifica il problema da ottimizzare, e $\hat{H}_M$ è l'Hamiltoniano di mixing che permette di esplorare lo spazio delle soluzioni. I parametri γ e β sono parametri variazionali che vengono ottimizzati durante l'esecuzione dell'algoritmo.

Il circuito quantistico inizia preparando tutti i qubit nello stato di sovrapposizione:

$$|s\rangle = |+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle. \tag{20}$$

Lo stato finale dopo p layers è dato da:

$$|\psi_p(\gamma, \beta)\rangle = e^{-i\beta_p \hat{H}_M} e^{-i\gamma_p \hat{H}_C} \cdots e^{-i\beta_1 \hat{H}_M} e^{-i\gamma_1 \hat{H}_C} |s\rangle \tag{21}$$

L'obiettivo è massimizzare il valore atteso dell'Hamiltoniano di costo:

$$F_p(\gamma, \beta) = \langle \psi_p(\gamma, \beta) | \hat{H}_C | \psi_p(\gamma, \beta) \rangle \tag{22}$$

Il processo di ottimizzazione avviene in modo ibrido: il circuito quantistico prepara e misura gli stati, mentre un ottimizzatore classico aggiorna i parametri γ e β per massimizzare F_p. Questo processo viene ripetuto fino a quando non si trova una soluzione soddisfacente al problema di ottimizzazione (Figura 3).

L'algoritmo QAOA rappresenta quindi un ponte tra il calcolo quantistico e quello classico, sfruttando i vantaggi di entrambi i paradigmi per risolvere problemi di ottimizzazione complessi. La sua efficacia dipende dal numero di layer p utilizzati e dalla qualità dell'ottimizzazione dei parametri.

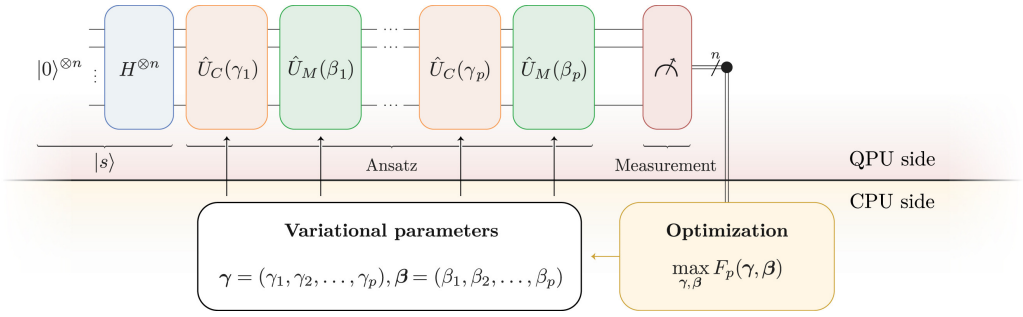


Figure 3: Schema del funzionamento del QAOA (Blekos *et al.*, 2024).

4 Risoluzione del problema

Per simulare il problema di ottimizzazione del portafoglio, abbiamo utilizzato un approccio basato su un modello generativo per produrre dati storici. La generazione dei dati è stata realizzata tramite una classe fornita da Qiskit, `RandomDataProvider`, che, a partire da parametri come il numero di asset, il periodo temporale e un seme per la riproducibilità, ha prodotto una tabella di valori rappresentante l'andamento dei prezzi degli asset nel tempo (Tabella I). Inoltre, per rendere più intuitiva la visualizzazione dei dati, abbiamo sviluppato uno script che crea un grafico per ciascun asset, rappresentando l'andamento temporale dei prezzi (Figura 4a).

Data	**Ticker_0**	**Ticker_1**	**Ticker_2**	**Ticker_3**
2016–01–01	80.0573	38.8648	69.2682	76.1653
2016–01–02	80.3390	38.7966	69.5354	76.6583
2016–01–03	79.4722	39.0100	70.7954	76.0500
...	...	...	...	...

Table I: Esempio di dati generati dalla classe `RandomDataProvider` con un numero di asset uguale a 4 e un periodo temporale che parte dal 01/01/2016.

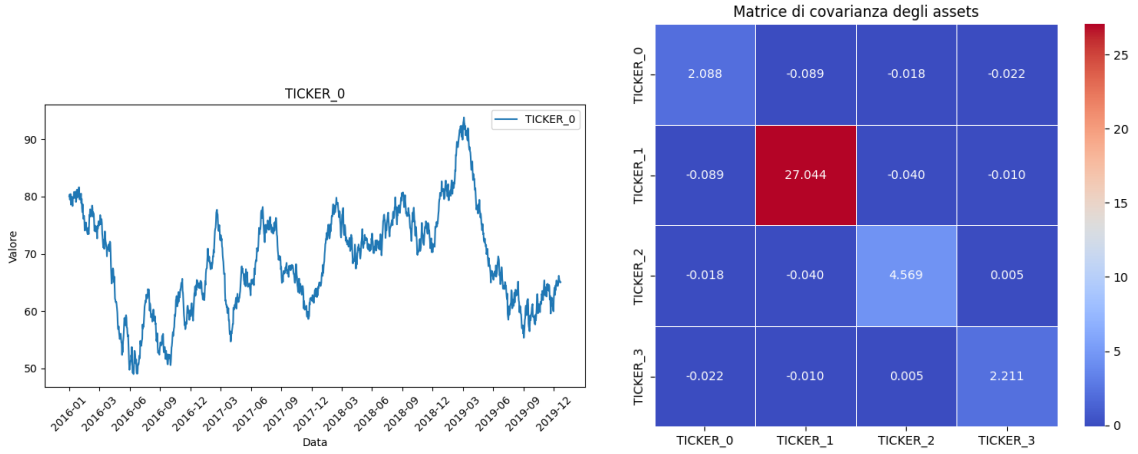
Successivamente, sono stati calcolati i rendimenti medi per ciascun asset (Formula 6) e la matrice di covarianza (Formula 7) utilizzando i metodi offerti dalla classe `RandomDataProvider`.

Questi due elementi rappresentano una componente fondamentale per la PO, in quanto i rendimenti medi forniscono una misura dell'aspettativa di guadagno, mentre la matrice di covarianza descrive la relazione tra le variazioni dei prezzi dei diversi asset.

```
mu = data_provider.get_period_return_mean_vector()
sigma = data_provider.get_period_return_covariance_matrix()
```

Listing 1: Calcolo dei rendimenti attesi e della matrice di covarianza.

Per favorire una migliore comprensione e visualizzazione della matrice di covarianza, ne è stato generato un grafico (Figura 4b): questo consente di evidenziare visivamente le correlazioni tra i diversi asset.



(a) Andamento temporale di un asset.

(b) Esempio di matrice di covarianza.[1]

Figure 4: (a) Andamento dei prezzi degli asset generati, e (b) matrice di covarianza calcolata.

Dopo aver calcolato i rendimenti medi e la matrice di covarianza per ciascun asset, procediamo con la configurazione del problema di ottimizzazione. Il fattore di rischio viene impostato a *0.2*, permettendo di bilanciare il trade-off tra rendimento e rischio del portafoglio. Il budget viene definito come i due terzi del numero totale di asset (`assets / 3 * 2`). Viene inoltre definita la penalità (`penalty`) uguale al valore del budget per gestire la violazione dei vincoli.

Il problema viene quindi configurato attraverso l'inizializzazione di un'istanza della classe `PortfolioOptimization`, alla quale vengono passati i parametri fondamentali: il vettore dei rendimenti attesi (`expected_returns`), la matrice di covarianza (`covariances`), il fattore di rischio (`risk_factor`) e il budget (`budget`). La configurazione finale produce un output che specifica il numero totale di asset disponibili, il budget allocato, la penalità impostata e il fattore di rischio utilizzato. Il problema viene infine convertito in un programma quadratico attraverso il metodo `to_quadratic_program` per la successiva ottimizzazione.

```
risk_factor = 0.2
budget = int(assets / 3 * 2)
penalty = budget

po = PortfolioOptimization(
    expected_returns=mu,
    covariances=sigma,
    risk_factor=risk_factor,
    budget=budget
)
qp = po.to_quadratic_program()
```

Listing 2: Configurazione del problema di PO.

Risoluzione classica del problema Per ottenere una soluzione di riferimento al nostro problema di ottimizzazione, utilizziamo un approccio classico basato su autovalori. In particolare, impieghiamo il risolutore `NumPyMinimumEigensolver`, un algoritmo che calcola in modo esatto gli autovalori del problema. Questo risolutore viene integrato attraverso `MinimumEigenOptimizer`,

[1] I valori della matrice sono stati moltiplicati per 10^5 per facilitare la lettura nella relazione, mentre per i calcoli sono stati utilizzati i valori originali.

un wrapper che fornisce un'interfaccia conveniente per utilizzare il solver e risolvere il problema di ottimizzazione.

```
exact_mes = NumPyMinimumEigensolver()
exact_eigensolver = MinimumEigenOptimizer(exact_mes)
result = exact_eigensolver.solve(qp)
```

Listing 3: Risoluzione classica del problema utilizzando `NumPyMinimumEigensolver`.

Questa implementazione classica serve come *ground truth* per valutare le prestazioni dei metodi quantistici che verranno implementati successivamente.

Si può osservare che nel risultato ottenuto nella Figura 9c, il risolutore classico fornisce un unico risultato, il quale corrisponde a quello ottimale, con una probabilità del 100%. Come già discusso nella Sezione 2, questo metodo è rapido quando tratta un numero ridotto di asset, ma diventa sempre più lento man mano che il numero di quest'ultimi aumenta. In definitiva, come già accennato, l'obiettivo principale del quantum computing non è tanto migliorare il risultato ottenuto, quanto piuttosto accelerare la risoluzione dei problemi.

Risoluzione con VQE Per risolvere il problema del PO con un approccio ibrido quantistico-classico, abbiamo scelto, come prima soluzione, di implementare il VQE (Qiskit, 2024). Come già spiegato nella Sezione 3.1, questo algoritmo combina la potenza computazionale dei circuiti quantistici con l'efficienza degli ottimizzatori classici, utilizzando un ansatz parametrizzato per esplorare lo spazio degli stati quantistici e individuare la configurazione che minimizza l'energia del sistema.

L'ansatz è ottenuto dal circuito `TwoLocal`, caratterizzato da porte R_y per la rotazione dei qubit, porte CZ per l'entanglement, e l'inserimento di barriere per favorire la leggibilità.

```
ry = TwoLocal(assets, "ry", "cz",
              reps=1,
              entanglement="full",
              insert_barriers=True)
```

Il circuito parametrizzato è stato integrato nel metodo `SamplingVQE`, che utilizza un campionatore (`Sampler()`) e l'ottimizzatore COBYLA per la minimizzazione dei parametri. La soluzione del problema è stata ottenuta tramite `MinimumEigenOptimizer`, come mostrato nel seguente frammento di codice.

```
svqe_mes = SamplingVQE(sampler=Sampler(),
                       ansatz=ry,
                       optimizer=cobyla)
svqe = MinimumEigenOptimizer(svqe_mes, penalty)
result_svqe = svqe.solve(qp)
```

La Figura 5 mostra lo schema del circuito generato dall'ansatz `TwoLocal`, evidenziando le porte R_y, le connessioni entangled e le barriere di separazione.

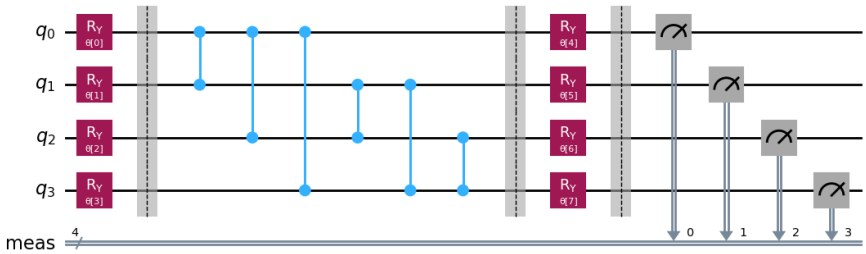


Figure 5: Circuito parametrizzato per il VQE, generato con l'ansatz `TwoLocal`.

Un esempio di risultato di un'esecuzione del VQE è mostrato nella Figura 9a, in cui si evidenziano la configurazione ottimale degli asset e il valore della funzione obiettivo. Questo risultato rappresenta l'allocazione di portafoglio ottimale secondo il modello definito.

Risoluzione con QAOA Un secondo approccio che abbiamo deciso di utilizzare per risolvere il problema del PO è stato quello del QAOA. Come descritto nella Sezione 3.2, questo algoritmo utilizza un'architettura parametrizzata che integra circuiti quantistici e ottimizzatori classici, con l'obiettivo di approssimare la configurazione ottimale attraverso un processo iterativo di adattamento dei parametri.

In una prima fase, abbiamo provato a implementare manualmente il QAOA, ma questa soluzione si è rivelata poco efficiente dal punto di vista computazionale, richiedendo tempi di esecuzione eccessivamente lunghi. Per una questione di ottimizzazione e scalabilità, è stato quindi deciso di utilizzare la classe `QAOA` offerta da Qiskit (Qiskit, 2024), che fornisce un'implementazione già ottimizzata dell'algoritmo.

Il circuito parametrizzato del QAOA è stato configurato con il numero di ripetizioni (`reps=1`) e l'ottimizzatore COBYLA per la ricerca dei parametri ottimali. Il codice per configurare e risolvere il problema è il seguente:

```
qaoa_mes = QAOA(sampler=Sampler(), optimizer=cobyla, reps=1)
qaoa = MinimumEigenOptimizer(qaoa_mes, penalty)
result_qaoa = qaoa.solve(qp)
```

La Figura 6 mostra lo schema del circuito generato, evidenziando i layer parametrizzati e le connessioni tra i qubit.

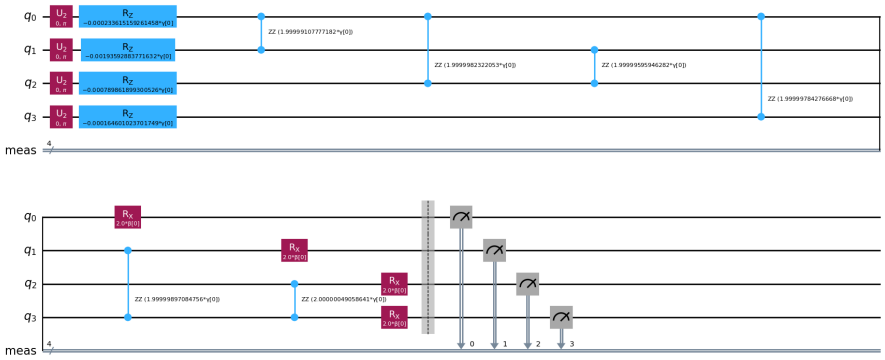


Figure 6: Circuito parametrizzato per il QAOA.

Un esempio di risultato di un'esecuzione del QAOA è mostrato nella Figura 9b, che evidenzia la configurazione ottimale degli asset e il valore della funzione obiettivo.

4.1 Introduzione del rumore

Finora, gli algoritmi VQE e QAOA sono stati simulati assumendo un sistema ideale, privo di rumore. Tuttavia, per avvicinare l'implementazione alla realtà dei computer quantistici attuali, abbiamo introdotto un modello di rumore basato su un backend generico, `GenericBackendV2`, offerto da Qiskit. Questo backend consente di configurare simulazioni personalizzate, adattandosi al numero di qubit richiesti e fornendo dettagli sul rumore associato al dispositivo simulato.

Il modello di rumore è stato generato utilizzando la classe `NoiseModel` e applicato al simulatore quantistico `AerSimulator`:

```
# Configurazione del backend generico
backend = GenericBackendV2(
    num_qubits=assets,
    noise_info=True,
    seed=seed,
    calibrate_instructions=True
)

# Generazione del modello di rumore
noise_model = NoiseModel.from_backend(backend)
simulator = AerSimulator(noise_model=noise_model)
```

Listing 4: Configurazione del modello di rumore per la simulazione.

Il backend `GenericBackendV2` è stato configurato per adattarsi al numero di qubit utilizzati nel problema di ottimizzazione del portafoglio (`assets`). L'opzione `noise_info` è stata abilitata per includere dettagli realistici sulle fonti di rumore, mentre la calibrazione automatica delle istruzioni (`calibrate_instructions=True`) garantisce una maggiore fedeltà nella simulazione del comportamento del dispositivo.

Il modello di rumore generato include informazioni relative ad errori di decoerenza, errori di misura e altre imperfezioni che caratterizzano i computer quantistici attuali. Più nel dettaglio, è stato implementato un modello che include un errore di flip del 5%, un errore di depolarizzazione

dell'1% per gate a 1 qubit e del 2% per gate a 2 qubit, e un errore del 2% sulle misure. Questa configurazione ha permesso di valutare le prestazioni degli algoritmi in un contesto più realistico rispetto a un sistema ideale, dove ci aspettiamo un moderato deterioramento delle prestazioni.

Implementazione del VQE con rumore Per il VQE con rumore, è stato utilizzato lo stesso circuito usato nella versione senza rumore.

```
ansatz = TwoLocal(
    num_qubits=assets,
    rotation_blocks="ry",
    entanglement_blocks="cz",
    reps=1,
    entanglement="full",
    insert_barriers=True
)
```

Listing 5: Configurazione dell'ansatz per il VQE.

Successivamente, l'ansatz è stato decomposto per ottenere una rappresentazione più dettagliata del circuito e adattato al backend rumoroso tramite la transpilazione.

```
decomposed_ansatz = ansatz.decompose()
transpiled_ansatz = transpile(decomposed_ansatz,
                              backend=backend)
```

Listing 6: Decomposizione e transpilazione dell'ansatz per il backend rumoroso.

Il circuito transpiled è stato poi utilizzato per configurare il metodo `SamplingVQE`, combinato con l'ottimizzatore COBYLA per minimizzare l'energia del sistema.

```
svqe_mes_noisy = SamplingVQE(sampler=Sampler(),
                             ansatz=transpiled_ansatz,
                             optimizer=cobyla)
svqe_noisy = MinimumEigenOptimizer(svqe_mes_noisy, penalty)
result_svqe_noisy = svqe_noisy.solve(qp)
```

Listing 7: Configurazione e risoluzione del VQE con rumore.

La Figura 7 mostra il circuito utilizzato per il VQE con rumore.

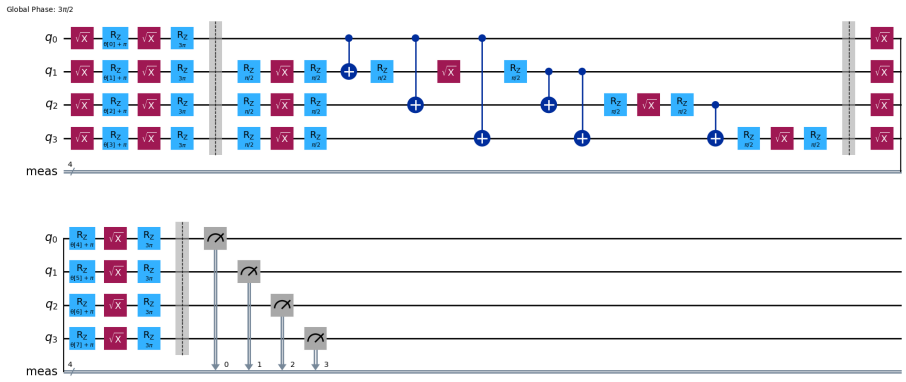


Figure 7: Circuito a 4 qubit parametrizzato per il VQE con rumore.

Implementazione del QAOA con rumore Anche il QAOA è stato implementato tenendo conto del rumore simulato tramite il backend `GenericBackendV2`. Come per il VQE, il circuito parametrizzato è stato adattato al backend rumoroso per rappresentare fedelmente le caratteristiche di un computer quantistico reale.

L'algoritmo è stato inizialmente configurato con la classe `QAOA`, utilizzando il sampler `Sampler()` e l'ottimizzatore COBYLA. Una prima esecuzione del metodo `solve` è stata necessaria per generare automaticamente il circuito iniziale, successivamente decomposto e transpiled per adattarsi al backend rumoroso. Con il circuito adattato, una seconda esecuzione del metodo `solve` ha permesso di risolvere il problema di ottimizzazione. Il codice completo è mostrato di seguito.

```
# Configurazione e prima esecuzione del QAOA
qaoa_mes_noisy = QAOA(sampler=Sampler(), optimizer=cobyla, reps=1)
qaoa_noisy = MinimumEigenOptimizer(qaoa_mes_noisy, penalty)
_ = qaoa_noisy.solve(qp)
```

```
# Decomposizione e transpilazione del circuito
decomposed_circuit_noisy = qaoa_mes_noisy.ansatz.decompose()
transpiled_circuit_noisy = transpile(decomposed_circuit_noisy, backend=backend)
qaoa_mes_noisy.ansatz = transpiled_circuit_noisy

# Risoluzione finale del QAOA
result_decomposed = qaoa_noisy.solve(qp)
```

Listing 8: Implementazione del QAOA con rumore.

La Figura 8 mostra il circuito transpiled utilizzato per il QAOA con rumore.

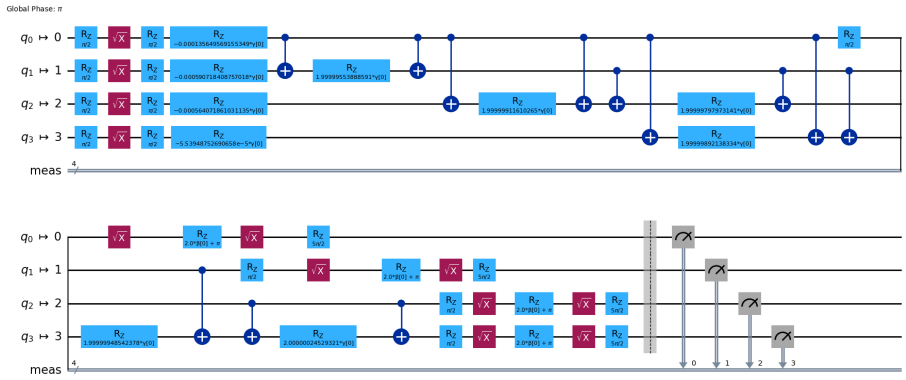


Figure 8: Circuito a 4 qubit parametrizzato per il QAOA con rumore.

Optimal: selection [1. 0. 1. 0.], value 0.0010

Full result

selection	value	probability
[1 0 1 0]	0.0010	0.5254
[0 1 1 0]	0.0027	0.4658
[0 1 0 0]	0.0019	0.0029
[0 0 1 0]	0.0008	0.0020
[1 0 0 0]	0.0002	0.0020
[1 1 0 0]	0.0022	0.0010
[0 0 0 0]	0.0000	0.0010

(a) Risultato ottenuto con VQE.

Optimal: selection [1. 0. 0. 1.], value 0.0004

Full result

selection	value	probability
[0 1 0 1]	0.0021	0.1494
[1 0 0 1]	0.0004	0.1309
[1 0 1 0]	0.0010	0.1250
[1 1 0 0]	0.0022	0.1172
[0 0 1 1]	0.0010	0.1123
[0 1 1 0]	0.0027	0.1104
[1 1 0 1]	0.0023	0.0352
[1 0 1 1]	0.0012	0.0332
[0 0 0 1]	0.0002	0.0312
[1 0 0 0]	0.0002	0.0303
[1 1 1 0]	0.0030	0.0293
[0 1 0 0]	0.0019	0.0264
[0 0 1 0]	0.0008	0.0254
[0 1 1 1]	0.0029	0.0225
[0 0 0 0]	0.0000	0.0117
[1 1 1 1]	0.0031	0.0098

(b) Risultato ottenuto con QAOA.

Optimal: selection [0. 1. 0. 1.], value -0.0002

Full result

selection	value	probability
-[0 1 0 1]	-0.0002	1.0000

(c) Risultato ottenuto con il risolutore classico.

Figure 9: Risultati ottenuti con (a) VQE, (b) QAOA e (c) risolutore classico.

5 Risultati

Per l'analisi dei risultati senza rumore (*noiseless*), è stata utilizza una configurazione con 8 asset, un budget massimo pari a 5, una penalità impostata sul valore del budget, un rischio pari al 20% e 50 ripetizioni per ciascun algoritmo. La Figura 10 mostra rispettivamente la distribuzione delle selezioni effettuate dagli algoritmi QAOA, VQE e Exact Eigensolver.

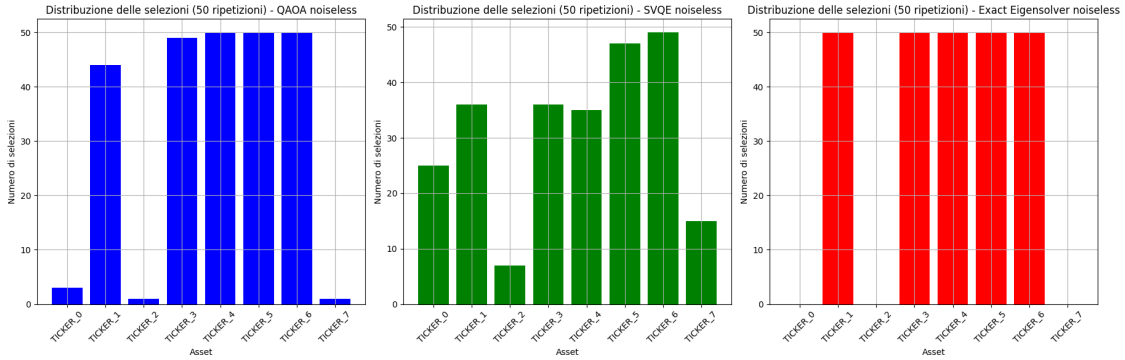


Figure 10: Distribuzione delle selezioni effettuate dagli algoritmi QAOA, VQE e Exact Eigensolver.

L'algoritmo QAOA evidenzia una preferenza marcata per specifici asset, che vengono selezionati con una frequenza significativamente maggiore rispetto agli altri. Questo comportamento è legato alla natura parametrizzata dell'algoritmo e alla sensibilità verso i valori iniziali dei parametri scelti per l'ottimizzazione. Tuttavia, questa limitata capacità esplorativa può risultare in una perdita di soluzioni ottimali o in una convergenza verso configurazioni subottimali.

D'altra parte, il VQE mostra una distribuzione più variegata delle selezioni. La capacità dell'algoritmo di adattare i parametri dell'ansatz consente di esplorare con maggiore efficacia lo spazio delle soluzioni. Nonostante questa maggiore diversificazione, i risultati non sono sempre allineati con quelli ottimali, evidenziando che alcune incertezze permangono anche in questo caso.

L'Exact Eigensolver rappresenta la *ground truth* del problema. Questo metodo fornisce sempre la soluzione esatta e ottimale del problema, servendo da riferimento principale per valutare la qualità dei risultati ottenuti con gli approcci approssimati.

La Figura 11 confronta le distribuzioni delle selezioni dei tre algoritmi, evidenziando le principali differenze. Il QAOA tende a concentrare le sue scelte su un insieme ristretto di asset, confermando una minore capacità esplorativa rispetto al VQE, che riesce invece a bilanciare meglio esplorazione ed exploitazione.

Infine, la Figura 12 analizza la frequenza delle combinazioni di asset selezionate nelle 50 ripetizioni. Anche qui si nota come il QAOA presenti una concentrazione delle combinazioni selezionate, mentre il VQE esplora una maggiore varietà di configurazioni.

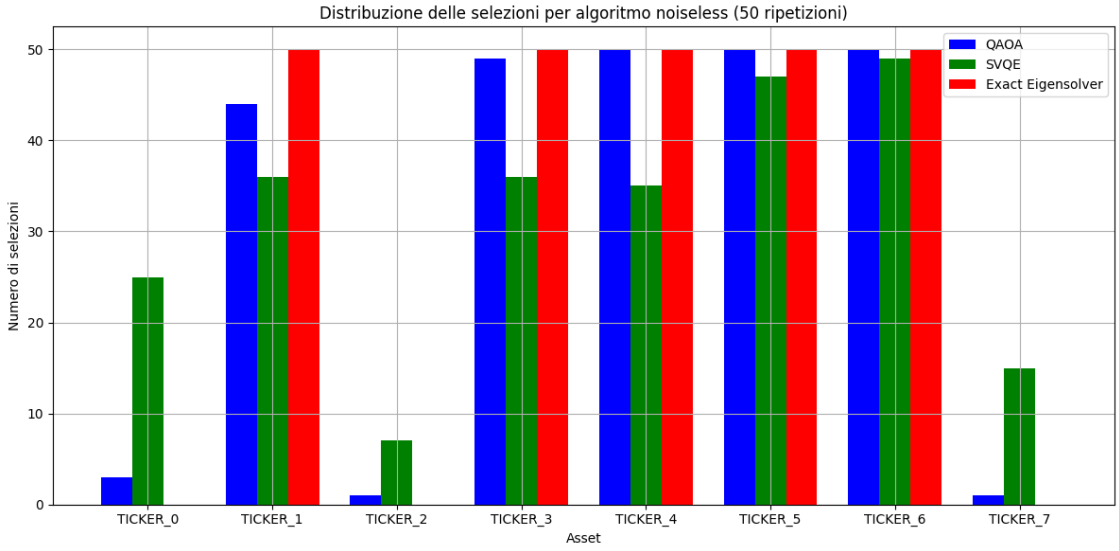


Figure 11: Distribuzione delle selezioni per algoritmo.

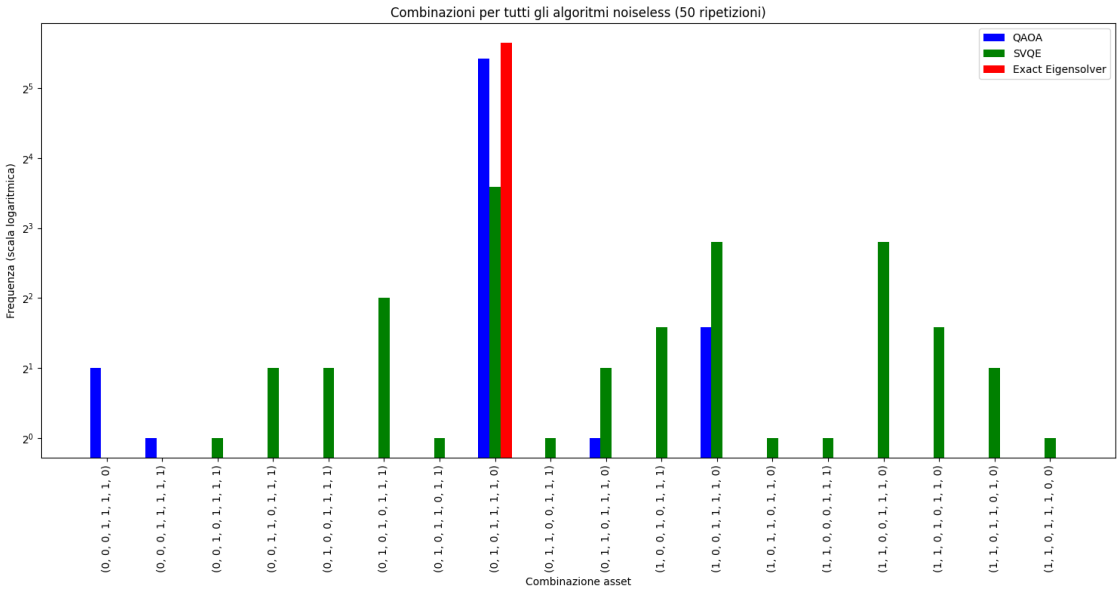


Figure 12: Frequenza delle combinazioni di asset selezionate nei 50 esperimenti.

5.1 Risultati con l'introduzione del rumore

Per l'analisi dei risultati condizionati dal rumore (*noisy*), è stata utilizzata la stessa configurazione di base con 8 asset, un budget massimo pari a 5, una penalità impostata sul valore del budget, un rischio pari al 20% e 50 ripetizioni per ciascun algoritmo. La Figura 13 mostra rispettivamente la distribuzione delle selezioni effettuate dagli algoritmi QAOA e VQE in presenza di rumore.

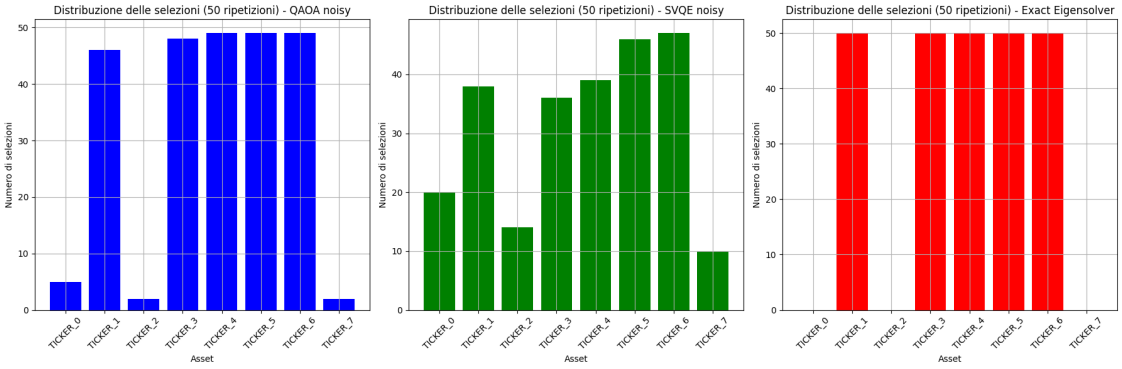


Figure 13: Distribuzione delle selezioni effettuate dagli algoritmi QAOA e VQE in presenza di rumore.

L'algoritmo QAOA, in presenza di rumore, mostra una distribuzione delle selezioni meno concentrata rispetto alla versione noiseless. Questo comportamento è dovuto alla sensibilità dell'algoritmo ai rumori di decoerenza e di gate, che influenzano negativamente la capacità di esplorare al meglio lo spazio delle soluzioni.

Il VQE, sebbene anch'esso influenzato dal rumore, riesce a mantenere una distribuzione delle selezioni più variegata rispetto al QAOA. La capacità dell'algoritmo di adattare i parametri dell'ansatz consente di mitigare parzialmente gli effetti del rumore, sebbene i risultati mostrino comunque una riduzione della qualità delle soluzioni rispetto alla versione noiseless.

La Figura 14 confronta le distribuzioni delle selezioni dei due algoritmi in presenza di rumore, evidenziando le principali differenze.

Infine, la Figura 15 analizza la frequenza delle combinazioni di asset selezionate nei 50 esperimenti in presenza di rumore.

Confronto algoritmi (noiseless vs noisy) Quest'ultimo confronto ci permette di valutare l'impatto del rumore sulle prestazioni degli algoritmi e sulla qualità delle soluzioni ottenute.

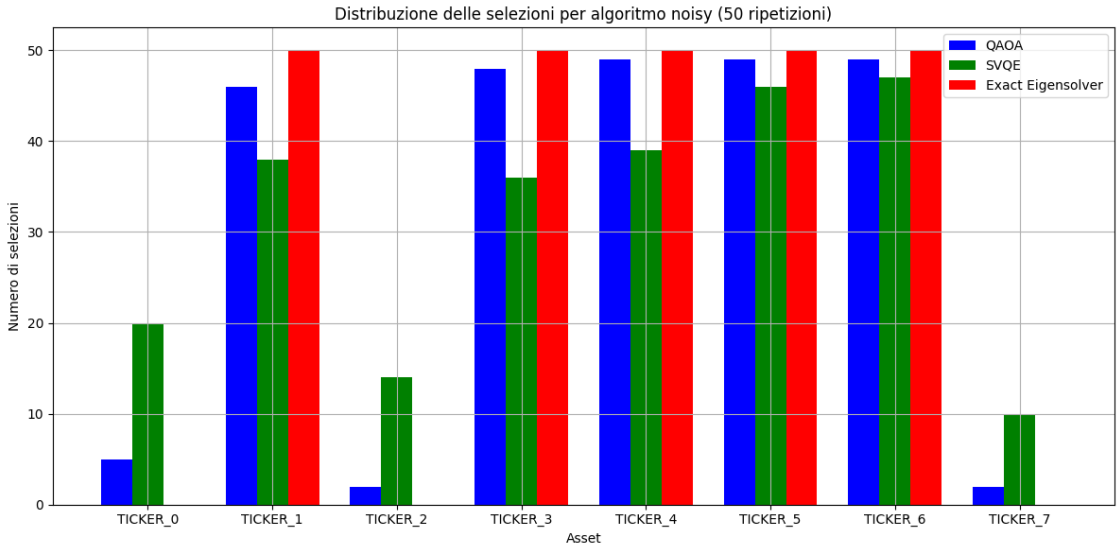


Figure 14: Distribuzione delle selezioni per algoritmo in presenza di rumore.

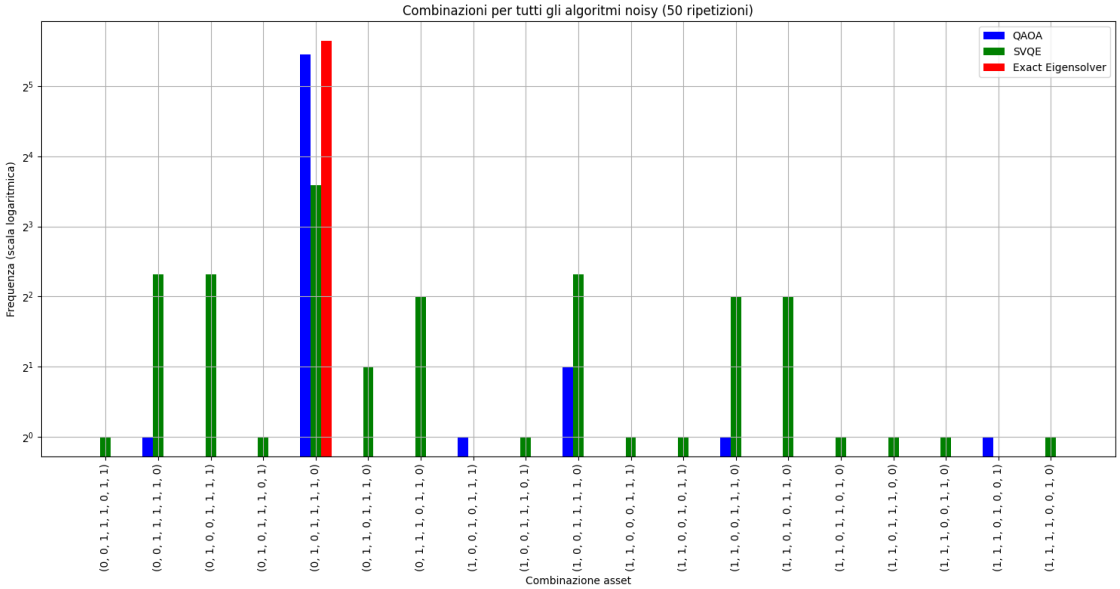


Figure 15: Frequenza delle combinazioni di asset selezionate nei 50 esperimenti in presenza di rumore.

La Figura 16 mostra il confronto delle combinazioni selezionate dall'algoritmo VQE in condizioni noiseless e noisy. Si può osservare che, in presenza di rumore, l'algoritmo VQE tende a selezionare un insieme più variegato di combinazioni rispetto alla versione noiseless. Questo comportamento è dovuto alla capacità dell'algoritmo di adattare i parametri dell'ansatz, che consente di esplorare con maggiore efficacia lo spazio delle soluzioni anche in presenza di rumore. Tuttavia, la qualità complessiva delle soluzioni ottenute dal VQE risulta inferiore rispetto a quella del QAOA, evidenziando una maggiore difficoltà nell'ottenere una scelta ottimale delle soluzioni in presenza di rumore.

La Figura 17 mostra il confronto delle combinazioni selezionate dall'algoritmo QAOA in condizioni noiseless e noisy. Anche in questo caso, si nota un insieme più variegato di soluzioni in presenza di rumore, ma comunque non ci si allontana di molto dalla soluzione ottimale. L'algoritmo QAOA, nonostante la presenza di rumore, riesce a mantenere una buona qualità delle soluzioni, dimostrando una maggiore robustezza rispetto al VQE.

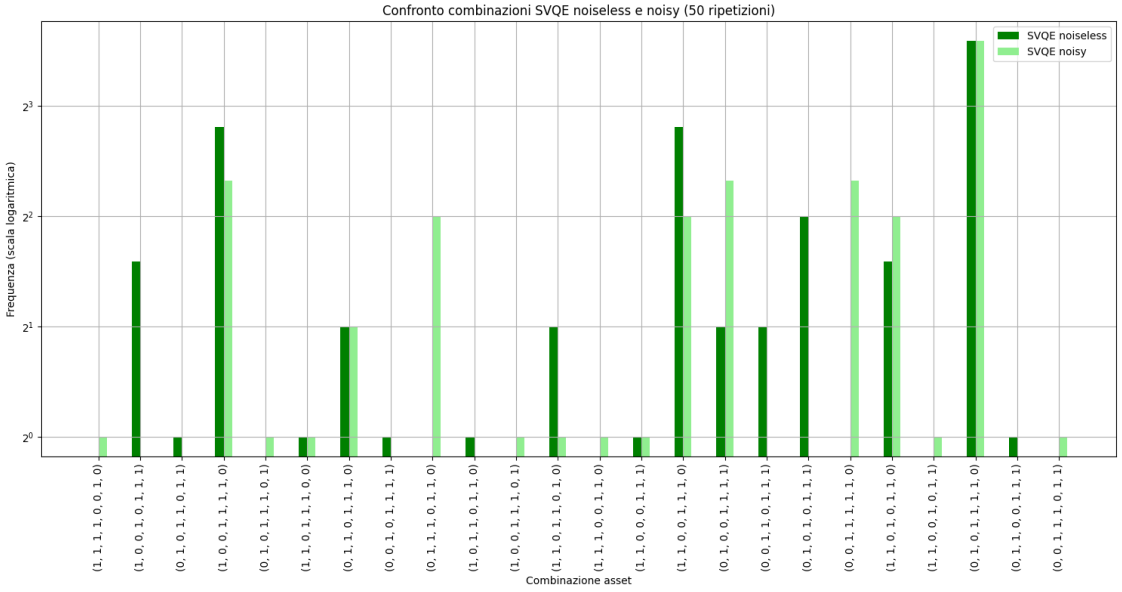


Figure 16: Confronto delle combinazioni selezionate dall'algoritmo VQE in condizioni noiseless e noisy.

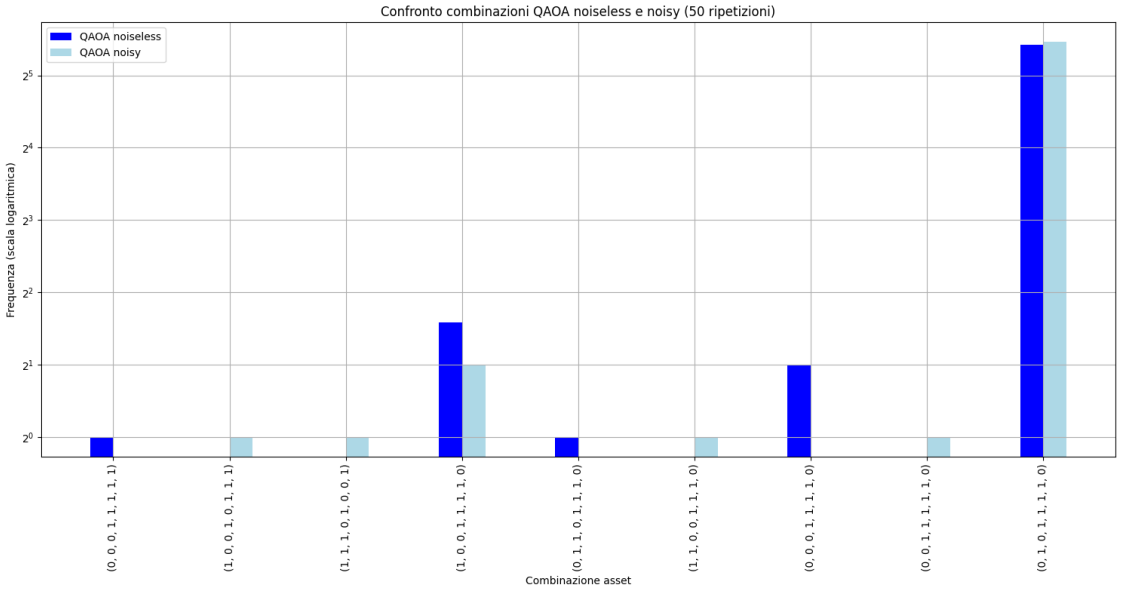


Figure 17: Confronto delle combinazioni selezionate dall'algoritmo QAOA in condizioni noiseless e noisy.

6 Conclusioni

I risultati ottenuti mostrano chiaramente l'impatto significativo del rumore sulle prestazioni degli algoritmi di ottimizzazione del portafoglio. L'algoritmo VQE, sebbene capace di esplorare una maggiore varietà di soluzioni, dimostra una sensibilità elevata al rumore, che porta a una qualità complessiva inferiore delle soluzioni rispetto al QAOA. D'altra parte, l'algoritmo QAOA riesce a mantenere una buona qualità delle soluzioni e una maggiore robustezza in presenza di rumore, pur mostrando una capacità esplorativa limitata.

Questi risultati evidenziano l'importanza cruciale di sviluppare tecniche di mitigazione del rumore per migliorare le prestazioni degli algoritmi quantistici in ambienti reali. Tuttavia, è importante sottolineare che, con la tecnologia attuale, l'utilizzo del quantum computing per risolvere problemi di ottimizzazione del portafoglio su larga scala, che richiederebbero migliaia di qubit, risulta impraticabile. La quantità di rumore introdotta in tali scenari sarebbe troppo elevata, compromettendo la qualità delle soluzioni proposte e rendendo inefficace l'approccio quantistico.

Acknowledgments

Questo progetto è stato realizzato durante il corso di Quantum Computing (a.a. 2024–25), presso l'Università degli Studi di Parma. Il codice sorgente del progetto è disponibile su Github:

`https://github.com/merendamattia/quantum-portfolio-optimization`.

References

Blekos, Kostas *et al.*, (2024). "A review on quantum approximate optimization algorithm and its variants", *Physics Reports*, Vol. 1068, pp. 1–66.

Buonaiuto, Giuseppe *et al.*, (2023). "Best practices for portfolio optimization by quantum computing, experimented on real quantum devices", *Scientific Reports*, Vol. 13 No. 1, p. 19434.

Land, Ailsa H and Doig, Alison G (2010). *An automatic method for solving discrete programming problems*, Springer.

Qiskit (2024). *Portfolio Optimization using Qiskit Finance*, `https://qiskit-community.github.io/qiskit-finance/tutorials/01_portfolio_optimization.html`.