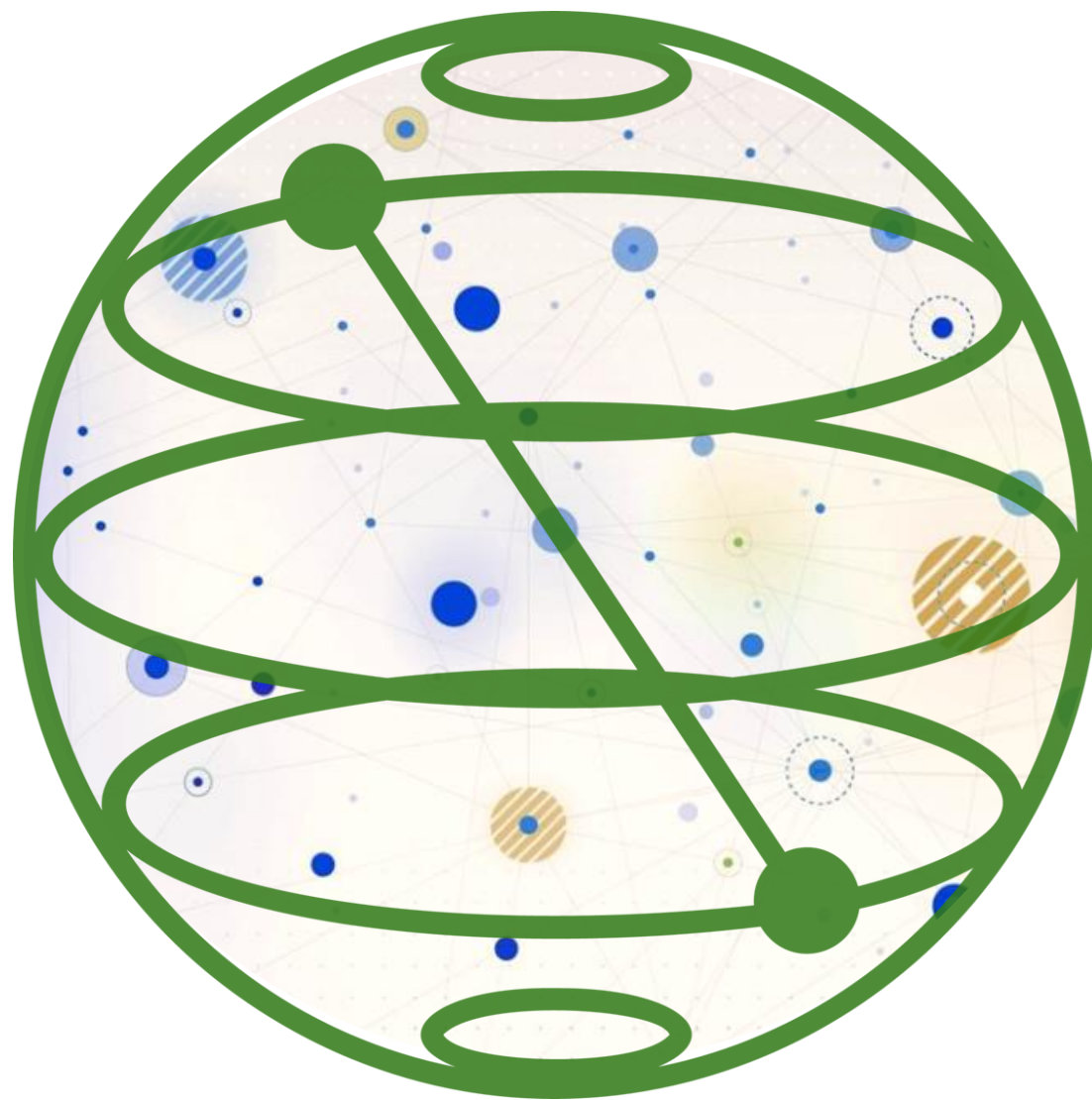


Corso di Quantum Computing - Giorno 5

Corso per Epigenesys s.r.l.

Docenti: Sara Galatro e Lorenzo Gasparini

Supervisore: Prof. Marco Pedicini





Simulazione di circuiti

Simulazione di circuiti quantistici

- Nonostante l'enorme potenza di calcolo teorica dei computer quantistici, la loro realizzazione non è affatto semplice.
- Gli attuali computer quantistici sono molto **limitati** e non permettono di portare a termine molte delle applicazioni della computazione quantistica già progettate in via teorica.
- La **simulazione** di computer quantistici su macchine classiche risulta fondamentale per lo **sviluppo** e la **progettazione** di applicazioni in ambito quantum computing.
- Sono fondamentali per l'analisi di algoritmi quantistici o per la verifica dell' output di computer quantistici fisici.
- Esistono molti approcci su come implementare una simulazione su un computer classico. Noi ci restringeremo a due metodi principali: simulazione di **Schrodinger** e simulazione di **Feynman** (anche detto **sum-over-paths**).

Simulazione di Schrodinger

- Questo tipo di simulazione è una tecnica di simulazione fondamentale.
- L'idea è quello di **rappresentare e memorizzare uno stato quantistico** e modificarlo attraverso le computazioni previste da uno specifico circuito.
- Se volessimo rappresentare uno stato quantistico ad n qubit, avremo bisogno di **rappresentare un vettore con 2^n entrate complesse**.
- Questo subito mette in evidenza le limitazioni di tale simulazione, essendo necessario memorizzare un numero esponenziale di entrate complesse. Possiamo memorizzare il vettore complesso tramite un **array di 2^n float**.
- Se usassimo, per esempio, 64 bit per la rappresentazione di un numero complesso come float, avremo bisogno globalmente di **$64 \cdot 2^n$ bit**. Ciò fa crescere molto velocemente la memoria necessaria per rappresentare un vettore. Già con $n = 32$, sarebbero necessari circa 8.5 GB di memoria.

Simulazione di Schrodinger

- Supponiamo di voler memorizzare lo stato quantistico $|0000\rangle$. Il vettore sarà di dimensione 16 e sarà il seguente:

$$[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]^T.$$

- Lo spazio necessario per la memorizzazione di tale vettore è di **128 byte**.
- L'altra parte fondamentale della simulazione riguarda la computazione, dunque l'applicazione di operatori quantistici sullo stato quantistico.
- Ricordiamo che una matrice U che agisce su k qubit, non è altro che una **matrice ad entrate complesse di dimensione $2^k \times 2^k$** . Essa può essere estesa ad un'azione globale applicando il prodotto tensoriale con l'operatore identità ed ottenere una matrice $2^n \times 2^n$.
- Rappresentare l'azione di un gate quantistico su uno stato quantistico è quindi implementata tramite una semplice **moltiplicazione matrice-vettore**.

Simulazione di Schrodinger

- Memorizzare una matrice che agisce su k qubit risulta estremamente dispendioso, ancora più di quanto serva per la rappresentazione di uno stato quantistico.
- Ricordiamo però che possiamo rappresentare ogni gate quantistico tramite composizioni di gate quantistici elementari.
- Considereremo solo **gate quantistici ad uno e due qubit**. Invece di considerare l'azione globale di un gate ad n qubit, considereremo azioni parziali di gate elementari (di dimensione 2×2 e 4×4).
- L'aggiornamento dello stato $|\psi\rangle$, in seguito all'applicazione di un gate unario sull' i -esimo qubit o all'applicazione di un gate binario sui qubit i e j , può essere portato a termine **in tempo** $O(2^n)$.

Simulazione di Schrodinger

- Se in totale il circuito quantistico che vogliamo simulare è formato da m gate elementari, allora **il costo in tempo totale per la simulazione è pari a $O(m \cdot 2^n)$** .
- Per quanto riguarda la memoria richiesta, vogliamo memorizzare tutto lo statevector, che viene aggiornato ad ogni gate che viene applicato.
- Dunque **lo spazio necessario è**, come già detto, **un $O(2^n)$ bit** più un eventuale memorizzazione delle **matrici** che rappresentano i gate quantistici.
- Con questo tipo di simulazione, **spazio e tempo crescono esponenzialmente** con il numero di qubit presenti nel circuito quantistico.
- La simulazione di Schrodinger è, per tale motivo, più **indicato per le simulazioni di circuiti quantistici che prevedono pochi qubit**.

Simulazione di Feynman

- L'idea qui è completamente differente dalla simulazione di Schrodinger.
- Non vogliamo memorizzare tutto lo stato quantistico e vogliamo risparmiare sull'utilizzo di spazio. La **simulazione di Feynman** è progettata allo scopo di poter calcolare la probabilità di misurare in output una certa stringa $b \in \{0,1\}^n$.
- Supponiamo di voler simulare un circuito di m gate elementari $[U_1, \dots, U_m]$ con vettore iniziale $|0^n\rangle$. La probabilità di misurare la stringa booleana b è data dalla seguente:

$$\langle b | U'_m \cdot \dots \cdot U'_1 | 0^n \rangle = \sum_{x_1, \dots, x_{m-1} \in \{0,1\}^n} \langle b | U'_m | x_{m-1} \rangle \langle x_{m-1} | U'_{m-1} | x_{m-2} \rangle \dots \langle x_1 | U'_1 | 0^n \rangle.$$

- Ogni scelta $\{x_1, \dots, x_{m-1}\}$ rappresenta uno specifico **path**.
- Tale algoritmo **richiede un tempo pari a $O(4^m)$ e uno spazio pari a $O(n + m)$** . Questo tipo di approccio in generale peggiora considerevolmente il costo di tempo ma migliora il costo della memoria richiesta.

Simulazione priva di entanglement

- Un caso di simulazione di circuito quantistico estremamente semplificato è previsto quando si è certi che nel corso della computazione quantistica non ci siano stati in entanglement.
- Ogni volta lo stato quantistico può essere rappresentato come **stato prodotto di n qubit**.
- Ogni qubit è rappresentato con un **vettore complesso a due entrate**.
- Senza calcolare necessariamente lo stato vettore globale, **memorizziamo solo gli n vettori indipendenti**.
- L'applicazione di un gate quantistico a questo punto si ha su un singolo qubit, indipendente dagli altri.
- Tale operazione è una semplice operazione matrice-vettore di dimensione 2×2 .

Up-to-date Shor

Tornando a Shor...



	Shor	Beauregard	Gidney-Ekera
<i>Elementary Gates</i>	$O(n^3)$	$O(n^3 \log n)$	$O(n^2 \log n)$
<i>Logical Qubit</i>	$3n$	$2n + 3$	$3n + O(n \log n)$

Breaking RSA-2048

- Seguendo l'implementazione proposta da Gidney-Ekera nella loro variante dell'algoritmo di Shor, per rompere lo schema RSA a 2048 bit, sarebbero necessari **14238 qubit logici**.
- Si è stimato inoltre che per la realizzazione di un qubit logico siano necessari circa **1583 qubit fisici a superconduttori**.
- Ciò comporta la possibilità di riuscire a **rompere efficientemente RSA-2048 con circa 20 milioni qubit fisici**.
- Le stime riportano inoltre che **il tempo di esecuzione risulta pari a 8 ore circa**.
- Utilizzando per esempio come alternativa, la variante di Beauregard, in numero di qubit diminuirebbe ma il costo computazione subirebbe un aumento non trascurabile.



Hidden Subgroup Problem

Hidden Subgroup Problem

- L'**Hidden Subgroup Problem** è un problema di teoria dei gruppi puramente teorico.
- La sua importanza è nata soprattutto in riferimento al contesto quantistico, dato che il problema generalizza una categoria di problemi significativi.
- In particolare **l'HSP risulta essere un problema a cui si riconducono molti problemi classici la cui risoluzione su un computer quantistico è esponenzialmente più veloce** rispetto ad un computer classico.
- Vedremo brevemente come possiamo ridurre tre dei problemi che abbiamo visto nell'ottica dell'Hidden Subgroup Problem.

Hidden Subgroup Problem

- Sia G un gruppo e $H \leq G$. Sia $f: G \rightarrow S$ una funzione che mappa un elemento del gruppo in una stringa binaria. Diremo che f **nasconde** H se vale la seguente:

$$f(g_1) = f(g_2) \Leftrightarrow g_1H = g_2H \Leftrightarrow g_1^{-1}g_2 \in H.$$

- Ciò significa che f assume un valore costante su differenti laterali di H .
- **HSP** → Sia G un gruppo e $H \leq G$ un sottogruppo sconosciuto. Sia $f: G \rightarrow S$ una funzione che nasconde H . Il problema consiste nel trovare un insieme generante per H solo usando le informazioni date da f .
- Lo strumento fondamentale alla base della soluzione quantistica che vedremo è la Quantum Fourier Transform, alla cui base c'è il cosiddetto **carattere** $\chi_g(h) = e^{\frac{2\pi i gh}{N}}$.

QFT abeliana

- Analizzeremo il problema nel caso abeliano (l'operazione del gruppo è commutativa).
- La **QFT abeliana** definita su un gruppo G è un operatore quantistico definito come segue:

$$QFT_G = \frac{1}{\sqrt{|G|}} \sum_{g,h \in G} \chi_g(h) |g\rangle \langle h|$$

- In questo caso stiamo codificando ogni elemento del gruppo G con uno stato quantistico della base computazionale standard.
- Definiamo anche due operatori ausiliari:

$$\tau_t = \sum_{g \in G} |t + g\rangle \langle g| \qquad \varphi_h = \sum_{g \in G} \chi_g(h) |g\rangle \langle g|$$

Soluzione quantistica abeliana

▪ L'algoritmo opera su due registri, uno ad n qubit e l'altro ad m qubit.

1) Inizializziamo il circuito nello stato $|0^n\rangle|0^m\rangle$.

2) Applichiamo la QFT_G sul primo registro:


$$(QFT_G \otimes \mathbb{I}^{\otimes m}) |0^n\rangle |0^m\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |0^m\rangle$$

3) Procediamo con l'applicazione di U_f :

$$U_f \left(\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |0^m\rangle \right) = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |f(g)\rangle = \frac{1}{\sqrt{|T|}} \sum_{t \in T} \tau_t |H\rangle |f(t)\rangle$$

Soluzione quantistica abeliana

4) Applichiamo nuovamente la QFT_G sul primo registro:

$$QFT_G\left(\frac{1}{\sqrt{|T|}} \sum_{t \in T} \tau_t |H\rangle |f(t)\rangle\right) = \frac{1}{\sqrt{|T|}} \sum_{t \in T} QFT_G \tau_t |H\rangle |f(t)\rangle$$

$$\frac{1}{\sqrt{|T|}} \sum_{t \in T} \varphi_t QFT_G |H\rangle |f(t)\rangle = \frac{1}{\sqrt{|H^\perp|}} \sum_{t \in T} \varphi_t |H^\perp\rangle |f(t)\rangle$$

5) Infine misuriamo il primo registro. Otteniamo così un **elemento randomico di H^\perp** .

- Bastano poche iterazioni per ottenere con buona probabilità un insieme generante per H^\perp , da cui poi si può applicare un semplice algoritmo classico per ricavare H .

Alcuni casi di HSP

- Passiamo a vedere il motivo per cui questo problema è importante, mettendo in luce tre problemi importanti che si riducono all'HSP.

Simon



Data una funzione booleana $f: \{0,1\}^n \rightarrow \{0,1\}^n$ di periodo s , trovare s .

Logaritmo discreto



Dato un gruppo $G = \langle g \rangle$ di ordine N e un elemento $A \in G$, trovare l'intero a tale che $g^a \equiv_N A$.

Period-finding



Data una funzione $f: \mathbb{N} \rightarrow \mathbb{Z}_N$ tale che esista r per cui $f(a) = f(b) \Leftrightarrow a \equiv_r b$, trovare l'ordine r .

Riduzione ad HSP

- L'idea generale per ridurre un problema ad un altro è quella di riuscire a tradurre l'input di un problema in un input dell'altro e far sì che una soluzione del problema a cui ci riduciamo, identifichi la soluzione del problema da ridurre.
- Nello specifico dell'HSP: convertiranno l'input in un input HSP, formata dal gruppo G e la funzione f che nasconde H , il quale identifica la soluzione al problema da ridurre.

Simon



- $G = \{0,1\}^n$ con lo \oplus
- $H = \{0, s\} \leq G$
- Come f prendiamo la f in input al problema

Logaritmo discreto



- $G = \mathbb{Z}_N \times \mathbb{Z}_N$
- $f: G \rightarrow \langle g \rangle$ con $f(u, v) = A^u g^v$
- $H = \{(u, v) \mid f(u, v) = 1\} \leq G$

Period-finding



- $G = \mathbb{Z}_{\varphi(N)}$
- $H = \langle r \rangle \leq G$
- Come f prendiamo la stessa f in input al problema

The background features a complex, abstract pattern of concentric circles and digital elements. The circles are composed of various shades of blue and grey, with some segments filled with diagonal lines or small squares, creating a sense of depth and movement. The overall effect is reminiscent of a stylized, futuristic tunnel or a digital data stream.

Quantum & Post-quantum Cryptography

Quantum Cryptography

- La maggior parte dei sistemi crittografici a chiave pubblica possono essere rotti dall'algoritmo di Shor se in possesso di un computer quantistico sufficientemente potente.
- La **Quantum Cryptography** è una branca della crittografia che si occupa dello sviluppo di sistemi crittografici la cui sicurezza si basa esplicitamente su effetti quantistici.
- Ovviamente uno dei principali svantaggi riguarda la necessità che tutte le parti coinvolte nello schema crittografico siano in possesso di un computer quantistico.
- Analizzeremo in particolare un protocollo di **Quantum Key Distribution** molto importante, il famoso **protocollo BB84**.
- L'idea è sempre quella in cui due parti, Alice e Bob, devono scambiarsi una chiave da poter riutilizzare per comunicazioni future.

BB84 QKD

- Descriviamo l'importante protocollo di Bennett e Brassard, il quale permette ad Alice e Bob di ottenere una stringa binaria in comune come partenza per la chiave condivisa.
- Consideriamo le seguenti due possibili basi computazionali:

Z-basis



$\{|0\rangle, |1\rangle\}$

X-basis



$\{|+\rangle, |-\rangle\}$

- Identificheremo la **base Z con il bit 0** e la **base X con il bit 1**.

BB84 QKD

- 1) Alice sceglie n bit randomici a_1, \dots, a_n e n basi randomiche b_1, \dots, b_n . Alice codifica il bit a_i nella base b_i e invia tali qubit a Bob su un canale quantistico pubblico.
- 2) Bob sceglie n basi randomiche b'_1, \dots, b'_n e misura i qubit ricevuti rispetto a tali basi b'_i , ottenendo così gli n bit a'_1, \dots, a'_n .
- 3) Bob invia le basi b'_1, \dots, b'_n ad Alice (Alice è sicura che Bob abbia misurato i qubit da lei inviati). Alice invia le sue basi b_1, \dots, b_n a Bob. In media, per metà delle volte, le basi b_i e b'_i coincidono. Per queste basi abbiamo $a_i = a'_i$. Sia Alice che Bob conoscono le posizioni per cui tale uguaglianza è vera; chiamiamo **stringa comune** tale sequenza di indici.
- 4) Alice sceglie $\frac{n}{4}$ indici della stringa comune ed invia a Bob sia tali posizioni che i relativi valori a_i . Bob procede verificando se tali valori coincidono con i propri a'_i .
- 5) Se la verifica ha avuto successo, scartano i bit di test e usano i restanti bit come chiave condivisa.

BB84 QKD

- Dove risiede la sicurezza del protocollo?
- C'è una proprietà fondamentale della meccanica quantistica che è alla base della sicurezza di questo schema di scambio della chiave.
- Dato che un eventuale attaccante Eve, che vuole intercettare la comunicazione, non conosce le basi b_i scelte da Alice, l'unico modo che ha per ottenere informazioni è quello di misurare i qubit sul canale quantistico.
- **Eve misura i qubit inviati ad Alice, facendo collassare il sistema o parte di esso, rendendo evidente a Bob il tentativo di intercettazione.**
- Sottolineiamo il fatto che per eseguire questo protocollo è necessario che:
 - *Alice e Bob siano in possesso di un computer quantistico,*
 - *Alice e Bob abbiano accesso ad un canale di comunicazione quantistico ed uno classico.*

Post-Quantum Cryptography

- Con **Crittografia Post-Quantum (PQC)** ci riferiamo al campo della crittografia che mira a sviluppare schemi crittografici resistenti a computer sia classici che quantistici.

Algorithm	Functionality	Security	Impact	Mitigation
Shor	Factoring	$poly(n)$	RSA	PQC
Shor	DLP	$poly(n)$	DH, DSA, ECC	PQC
Grover	Key search	$2^{\frac{n}{2}}$	Sym. algorithm	Sufficient Key length
Grover	Pre-image	$2^{\frac{n}{2}}$	Hash function	Sufficient Hash length
BHT	collision	$2^{\frac{n}{3}} \text{ o } 2^{\frac{2n}{5}}$	Hash function	Sufficient Hash length

Post-Quantum Cryptography

- Per lo sviluppo di sistemi post-quantum sono state usate diverse strutture matematiche come base per problemi computazionalmente difficili grazie ai quali poter migrare ad una crittografia post-quantum. Tra queste riportiamo:
 - **Lattice-based cryptography**: basato sulla difficoltà di problemi su reticoli come SVP e LWE.
 - **Code-based cryptography**: basato sulla difficoltà di decodificare un codice lineare.
 - **Multivariate cryptography**: basato sulle equazioni multivariate come HFE.
 - **Hash-based cryptography**: basato solo sull'uso di funzioni hash.
 - **Isogeny-based cryptography**: basato sulla difficoltà di problemi su curve ellittiche.
- Le tecniche post-quantum sono molte, ma ben poche sono *sopravvissute*.

NIST Standardization

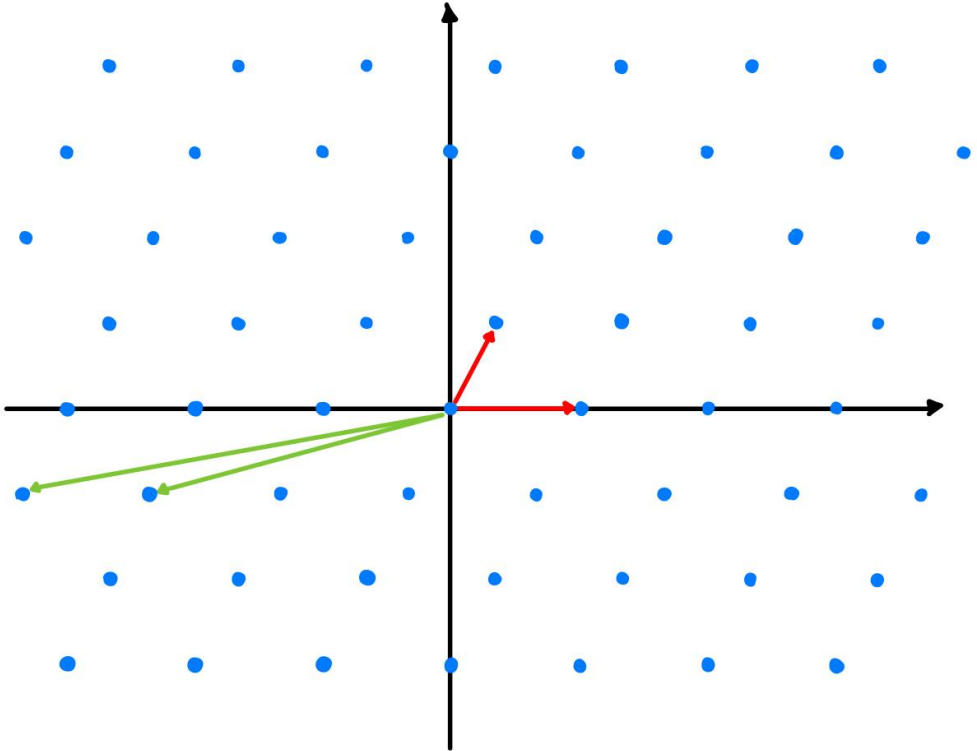


- Proprio per l'impatto che ha avuto l'algoritmo di Shor nella moderna crittografia a chiave pubblica, il National Institute of Standard Technologies ha avviato un programma di standardizzazione degli algoritmi post-quantum.
- Ci sono state moltissime valutazioni effettuate su moltissimi tipi di schemi crittografici proposti. Il NIST alla fine, nel 2022, ha annunciato i 4 finalisti come possibili schemi crittografici post-quantum.

PQC Algorithm	Cryptographic family	Application
CRYSTALS-Kyber	Lattice	Key encapsulation
CRYSTALS-Dilithium	Lattice	Digital Signature
FALCON	Lattice	Lightweight DS
SPHINCS+	Hash	Digital Signature

Lattice Based Cryptography

- Un **reticolo** è una collezione di punti regolarmente distribuiti all'interno di uno spazio.



- Un reticolo è l'insieme di tutte le combinazioni lineari intere di n vettori linearmente indipendenti:

$$\mathcal{L} = \{\sum_i c_i v_i \mid c_i \in \mathbb{Z}\}.$$

- Una base $B = \{b_1, \dots, b_n\}$ di un reticolo è un insieme di vettori linearmente indipendenti che genera l'insieme \mathcal{L} :

$$\mathcal{L}(B) = \{\sum_i c_i b_i \mid c_i \in \mathbb{Z}\}.$$

- La base di un reticolo \mathcal{L} **non è unica**.

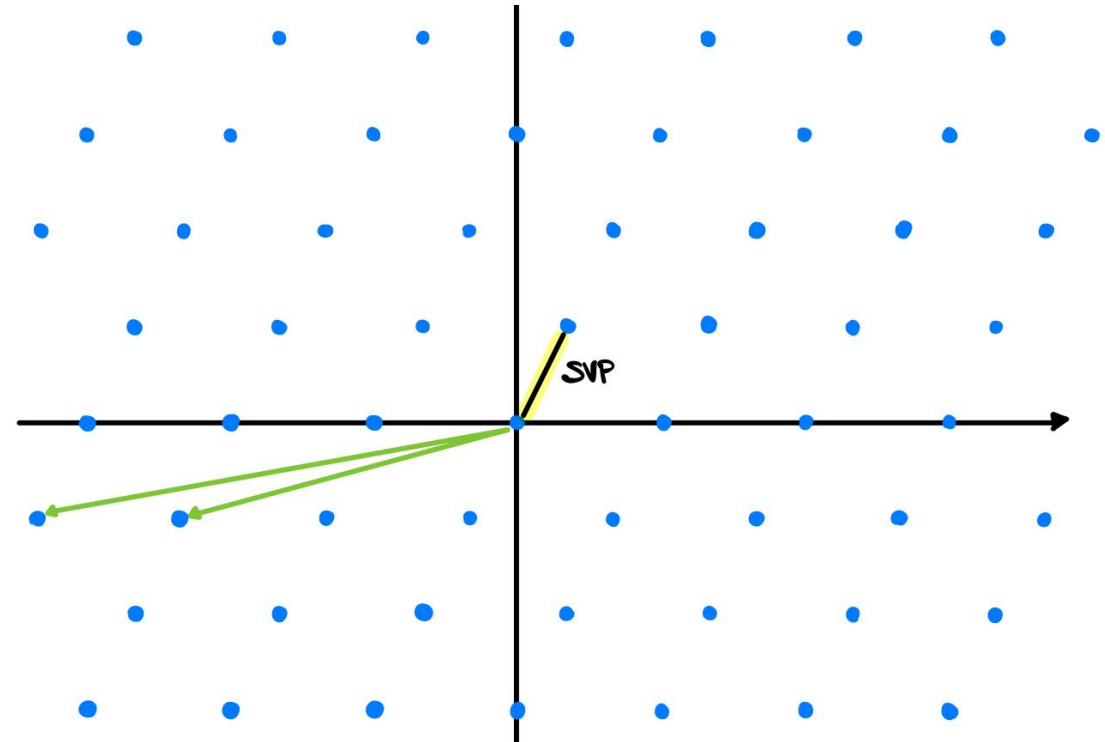
SVP e CVP

- Cominciamo con due problemi su reticoli di facile comprensione.

- Shortest Vector Problem:** dato un reticolo \mathcal{L} , trovare il vettore non nullo $v \in \mathcal{L}$ più corto, cioè tale che

$$\|v\| = \min_{v'} \|v'\| := \lambda(\mathcal{L}).$$

- La difficoltà sta nel fatto che non si è in grado di ridursi ad un vettore corto partendo da una **bad-basis**.

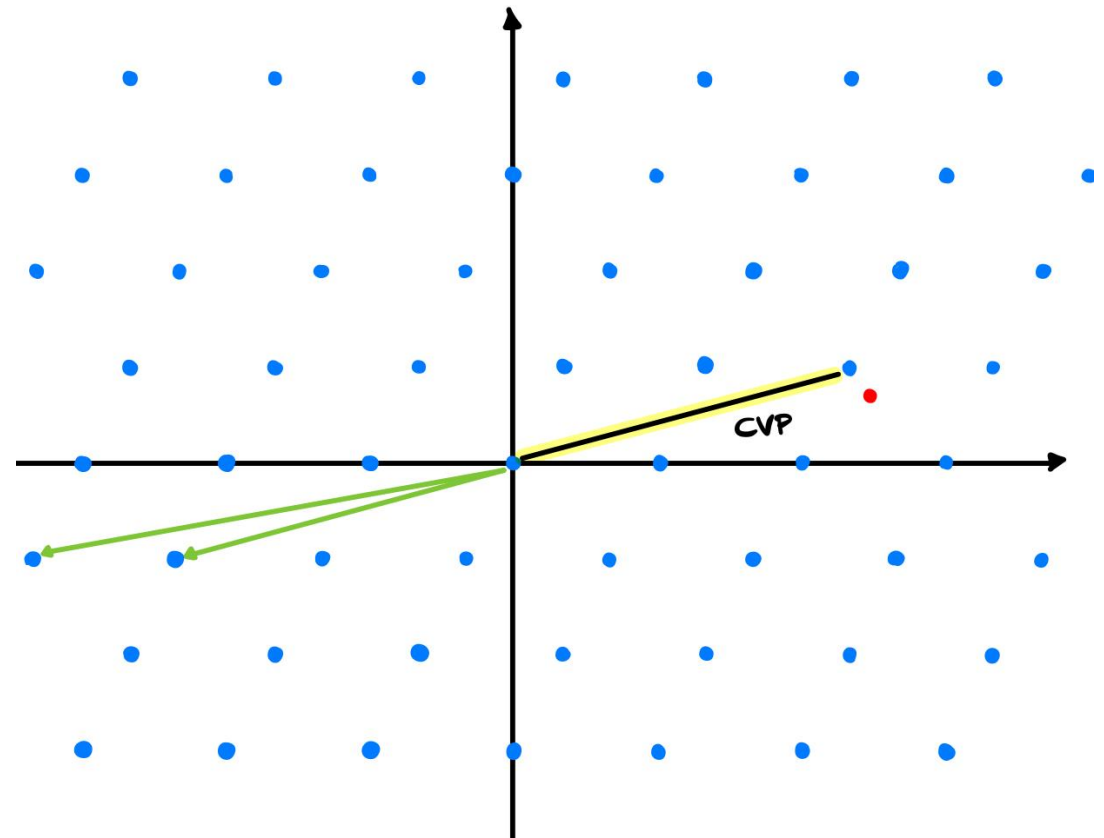


SVP e CVP

- **Closest Vector Problem:** dato un reticolo \mathcal{L} e un vettore target v , trovare il vettore non nullo $u \in \mathcal{L}$ più vicino a v , cioè tale che

$$d(u, v) = \min_{u'} d(u', v).$$

- Analogamente all'SVP, la difficoltà si basa sul fatto di non avere a disposizione una **good-basis**.
- Possiamo notare che SVP è un caso particolare di CVP.



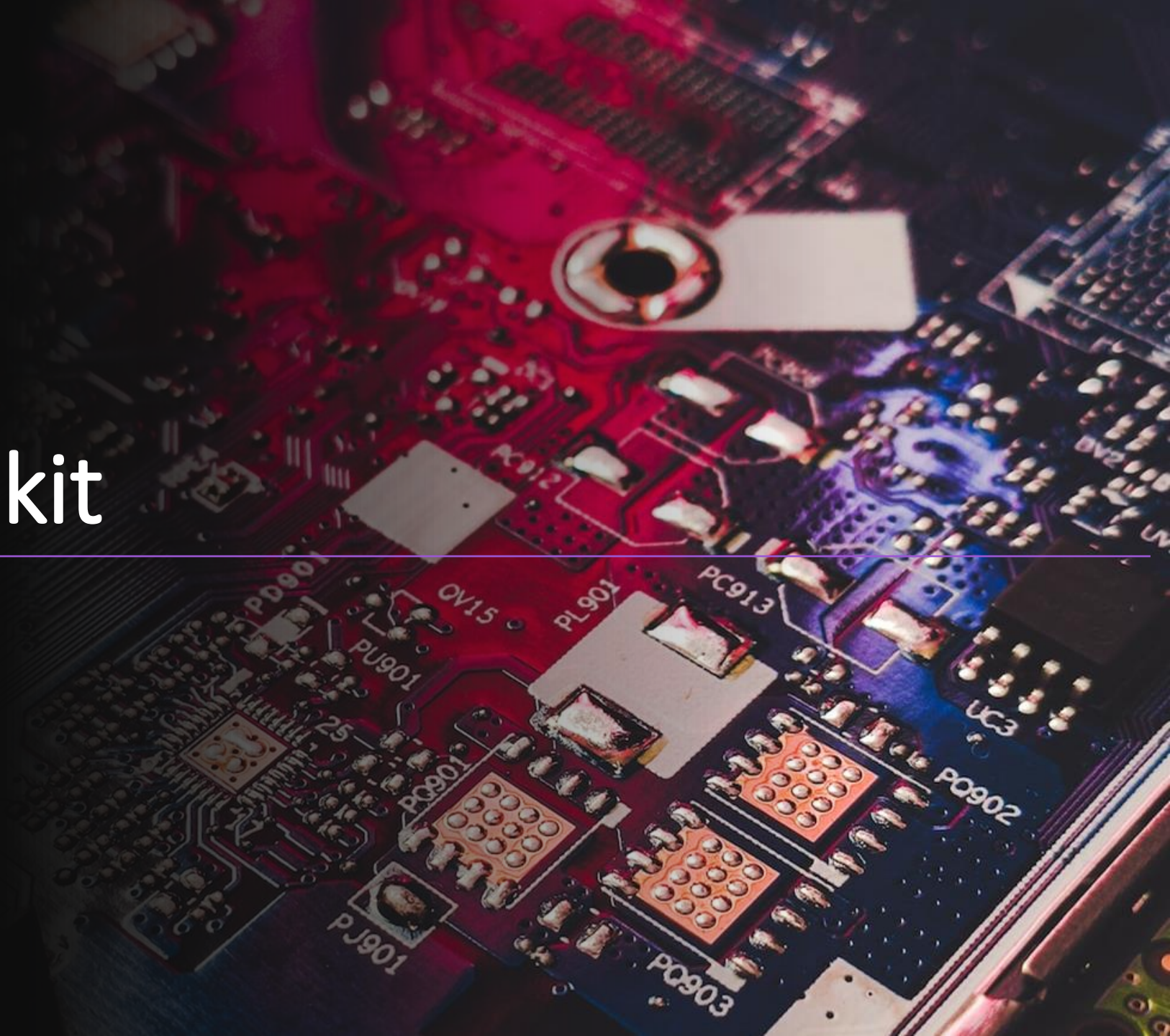
Learning With Errors

- Il problema fondamentale su reticoli alla base degli algoritmi post-quantum è il cosiddetto **Learning With Errors (LWE)**.
- L'idea di base è introdurre dell'errore all'interno di equazioni per rendere difficile l'inversione del calcolo.
- Sia $s \in \mathbb{Z}_q^n$ un vettore segreto e sia χ una distribuzione di probabilità d'errore su \mathbb{Z}_q . Viene costruita una **distribuzione** $A_{s,\chi}$ di m vettori in $\mathbb{Z}_q^n \times \mathbb{Z}_q$ nel seguente modo:
 - 1) Viene scelto uniformemente un elemento $a_i \in \mathbb{Z}_q^n$.
 - 2) Dalla distribuzione χ viene preso un errore $\epsilon_i \in \mathbb{Z}_q$.
 - 3) Si calcola $b_i = \langle a_i, s \rangle + \epsilon$.
 - 4) Si aggiunge il vettore (a_i, b_i) come sample per $A_{s,\chi}$.
- Dati come parametri q , χ e data in input la matrice $A_{s,\chi}$, **trovare** s .

LWE cryptosystem

- Il primo crittosistema (a chiave pubblica) nato dalla difficoltà dell'LWE è dovuto a **Regev**. Vediamone brevemente il funzionamento.
- **Chiave privata**: il vettore segreto $s \in \mathbb{Z}_q$.
- **Chiave pubblica**: si prendono m sample (a_i, b_i) dalla distribuzione LWE $A_{s,\chi}$.
- **Encryption**:
 - Viene cifrato bit per bit.
 - Per ogni bit del messaggio viene scelto randomicamente un $r \in \{0,1\}^m$.
 - Se il bit è 0, il cifrato sarà dato da $(\sum_i a_i r_i, \sum_i b_i r_i)$. Se il bit è 1, il cifrato sarà $(\sum_i a_i r_i, \lfloor \frac{q}{2} \rfloor + \sum_i b_i r_i)$.
- **Decryption**: decifriamo la coppia (a, b) come il bit 0 se $b - \langle a, s \rangle$ è più vicino a 0 che a $\lfloor \frac{q}{2} \rfloor$ modulo q .

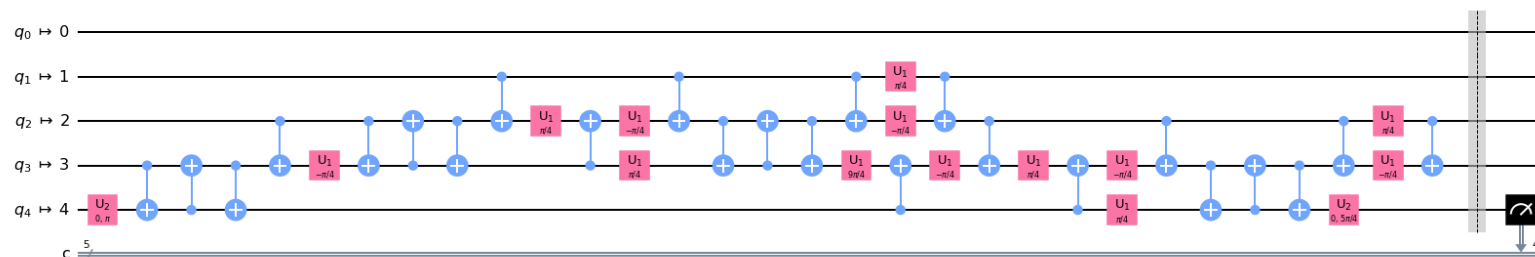
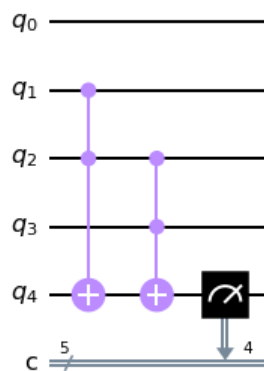
Come usare il transpiler di Qiskit



Introduzione



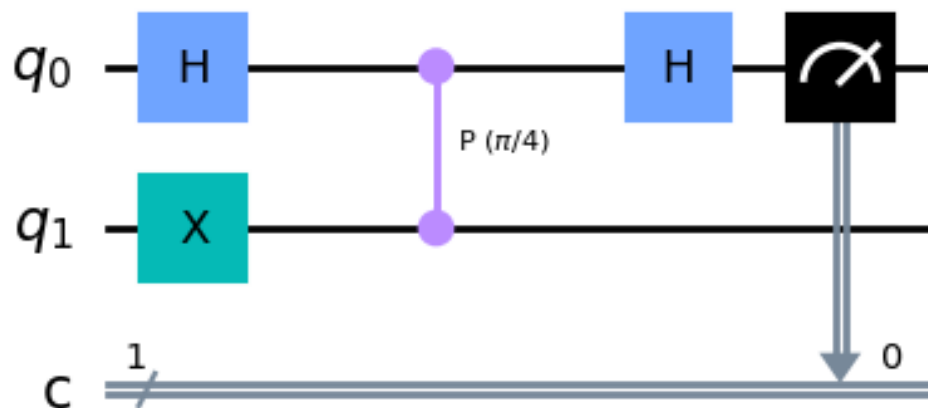
- Dato un circuito è necessario effettuare il **transpiling** prima di poterlo eseguire, così da renderlo compatibile con la macchina che vogliamo usare.
- Questo processo può anche essere sfruttato per **ottimizzare il circuito**, riducendo così l'effetto del rumore.
- Qiskit ha delle **funzioni predefinite** per tradurre i circuiti sulla topologia della macchina, funzioni che possiamo utilizzare o con parametri preimpostati o con specifiche ad-hoc per il nostro circuito.
- I **livelli di ottimizzazione** variano da 0 a 3, identificando un ordine crescente di ottimizzazione, e si basano tutti su **procedimenti euristici**.
- In generale, il processo di transpiling può essere suddiviso in **5 step**.

1**Translation
Stage****2****Layout
Stage****3****Routing
Stage****4****Optimization
Stage****5****Scheduling
Stage**

1

Translation Stage

Supponiamo di avere il seguente circuito



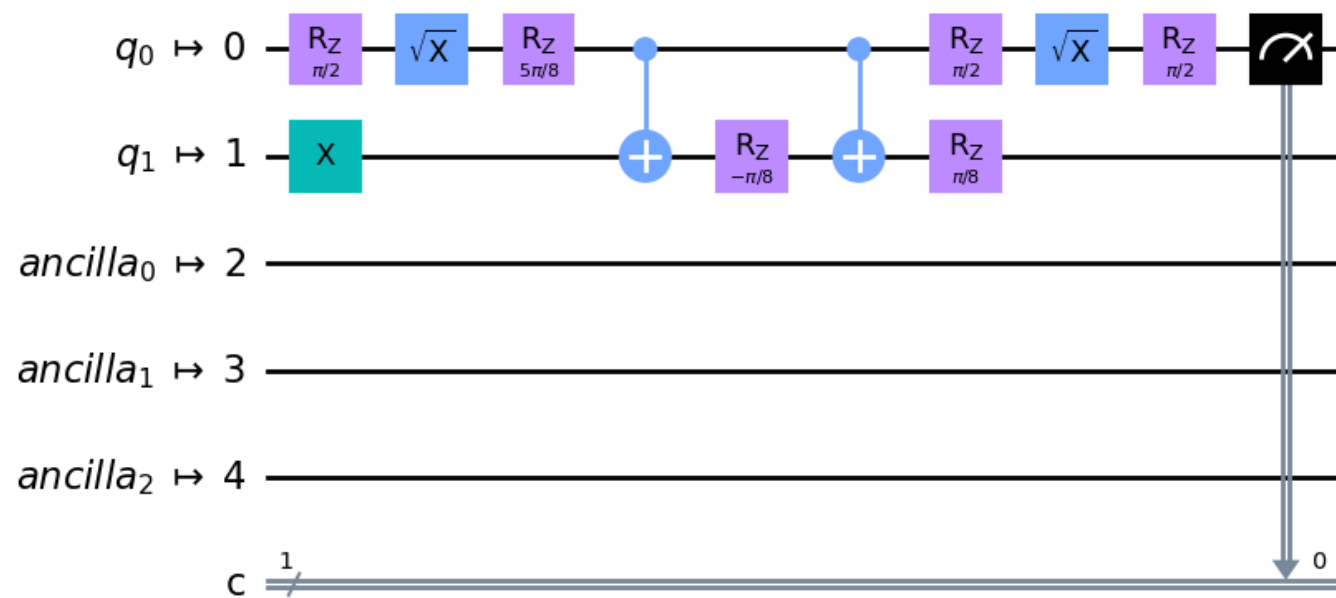
e di volerlo eseguire sulla macchina **FakeVigoV2**, una macchina dotata di cinque qubit che supporta le seguenti operazioni:

```
['id', 'rz', 'sx', 'x', 'cx', 'measure', 'delay']
```

1

Il risultato di questa traduzione è il seguente:

Global Phase: $9\pi/16$



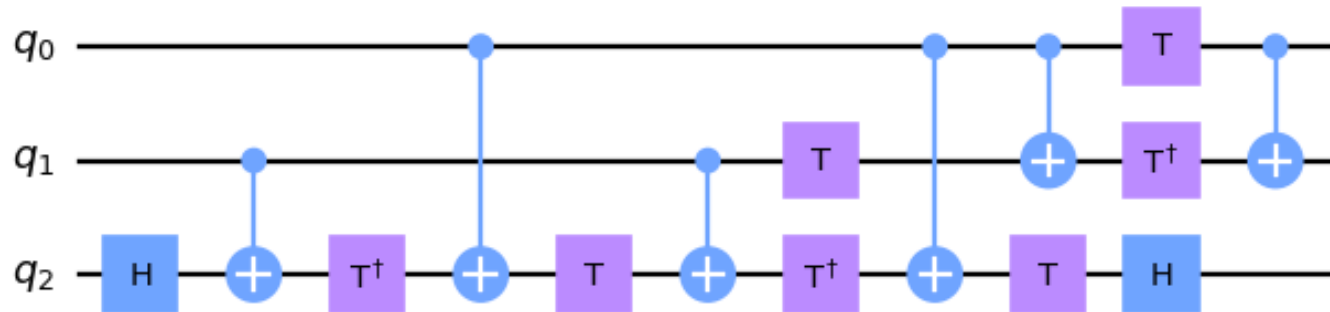
Translation Stage

1

Translation Stage

Due osservazioni importanti:

- 1) Se lo **SWAP gate** non è nella base, allora serviranno tre **CNOT** per simularlo;
- 2) Un **Toffoli gate (CCX)** è un gate a tre qubit e pertanto dovrà essere **sempre decomposto**:



Il circuito che definiamo è un **entità astratta**: dobbiamo trovare una **mappa iniettiva** che metta in relazione i nostri qubit virtuali con quelli fisici.

Qiskit si occupa di questo passaggio in **autonomia** e la sua scelta dipende dalle proprietà del circuito, dalla macchina utilizzata, dal livello di ottimizzazione scelto e dal **layout iniziale**.

Possiamo definire manualmente tale layout oppure cercarlo tramite i metodi di Qiskit, nei quali si susseguono due step:

- 1) Trovare, se esiste, un **layout perfetto**;
- 2) Procedere **per metodi euristici** per trovare il layout migliore (se quello perfetto non esiste).

Più in dettaglio, il primo step si può effettuare tramite due strategie:

- a) **TrivialLayout**, che mappa ogni qubit virtuale sul qubit fisico con lo stesso indice, ossia

$$[0,1,2,3,4] \rightarrow [0,1,2,3,4]$$

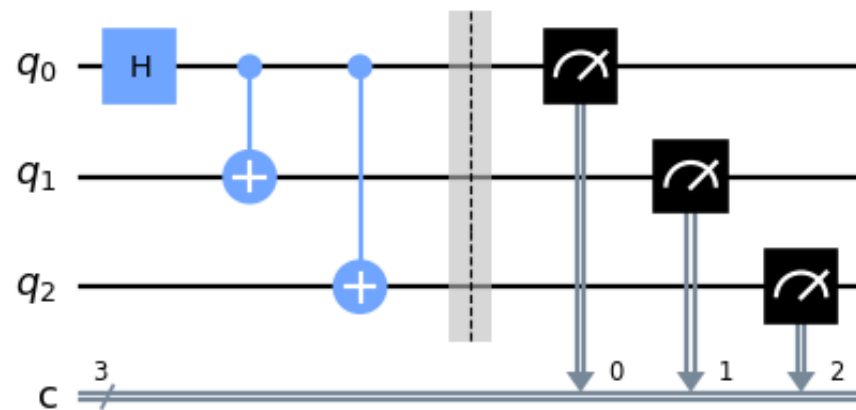
Se fallisce, allora invoca la seconda strategia.

- b) **VF2Layout**, che modella la ricerca di un layout come un problema di isomorfismo tra grafi. Se più di una soluzione è trovata, allora si eseguono alcune prove per scegliere quella con minor errore (in media).

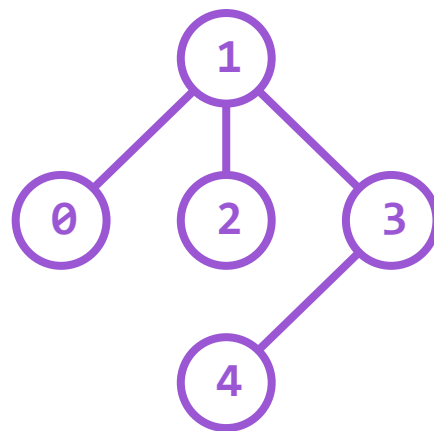
Per il secondo step ci sono tre strategie possibili:

- a) **SabreLayout**, che consiste nel partire da un'inizializzazione randomica ed eseguire una routine più volte, avanti e indietro nel circuito, permutando e aggiungendo SWAP gate per sistemare la configurazione;
- b) **TrivialLayout**, usato sempre per ottimizzazione di livello 0;
- c) **DenseLayout**, usato per ottimizzazione di livello 1 se nel circuito sono presenti operazioni di controllo. Trova il sottografo con maggiore connettività dato il numero di qubit nel circuito.

Esempio:



Device Topology:



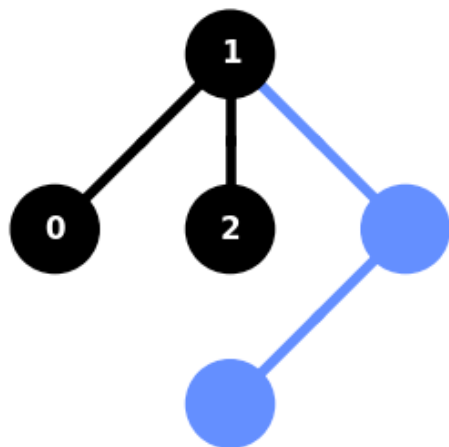
2

Layout
Stage

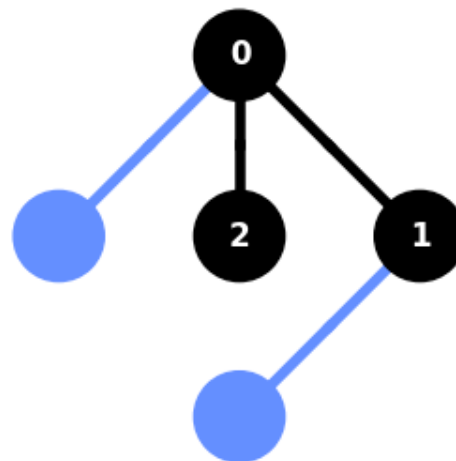
2

Layout Stage

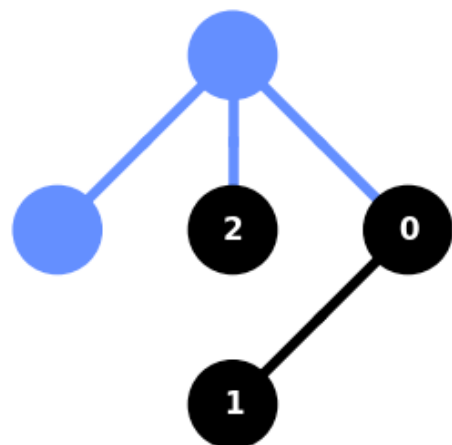
Opt. Lv. 0



Opt. Lv. 3



Manual layout:



<i>Virtual</i>	->	<i>Physical</i>
0	->	3
1	->	4
2	->	2

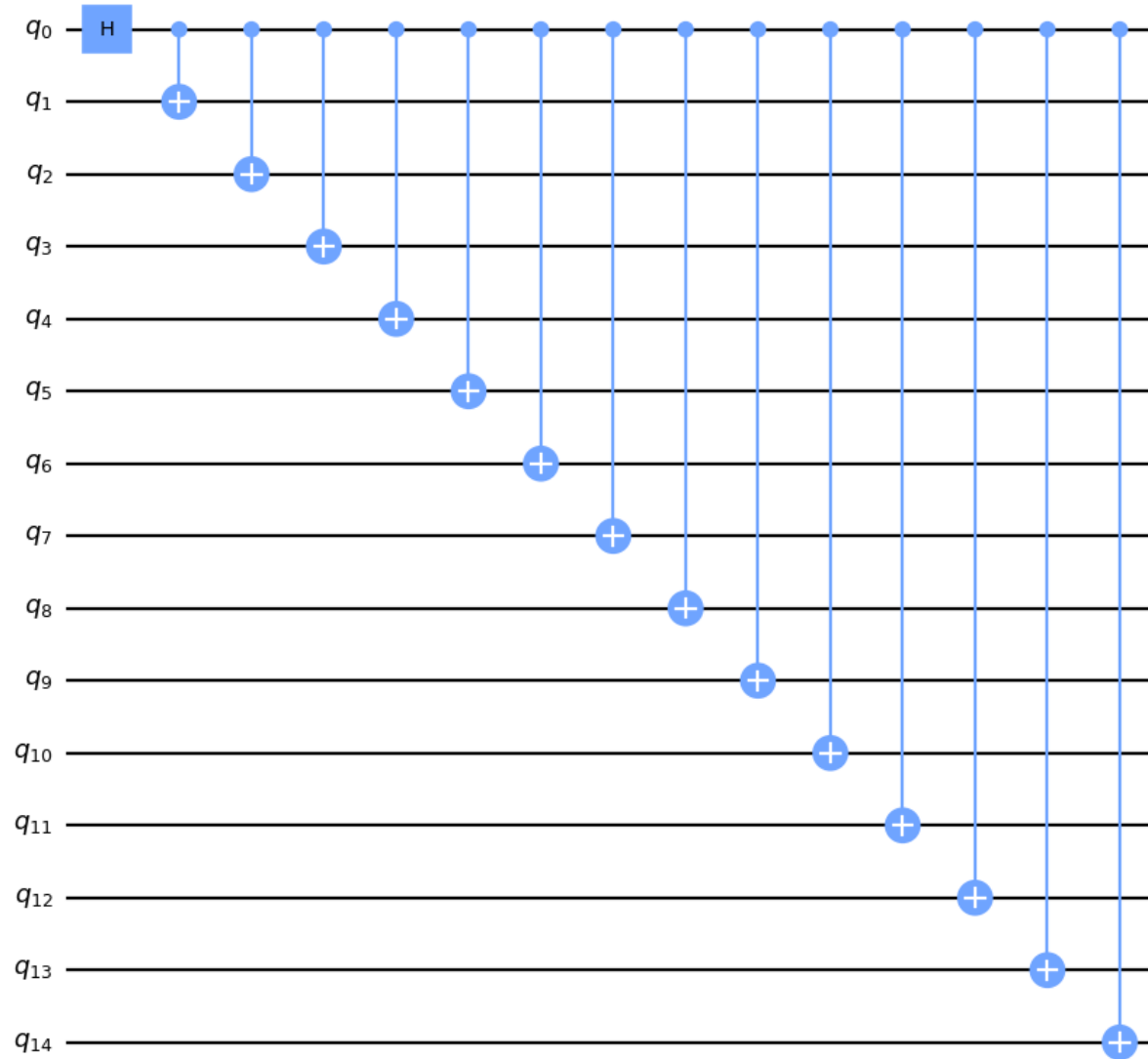
Sappiamo che in un circuito tradotto avremo solo **gate a 1 o a 2 qubit**.

Dovremo quindi assicurarci che due qubit che interagiscano tra loro siano **vicini**, per poter effettivamente applicare il relativo gate.

A tal fine, sarà necessario dunque porre degli **SWAP gate**, che però sappiamo essere costosi da simulare. Vogliamo quindi trovare una configurazione che abbia un **numero minimo di SWAP**.

Trovare questa configurazione ottimale è però un problema **NP-Hard**. Si usa pertanto un algoritmo stocastico ed euristico chiamato **SabreSwap** per trovare una soluzione che sia la migliore possibile.

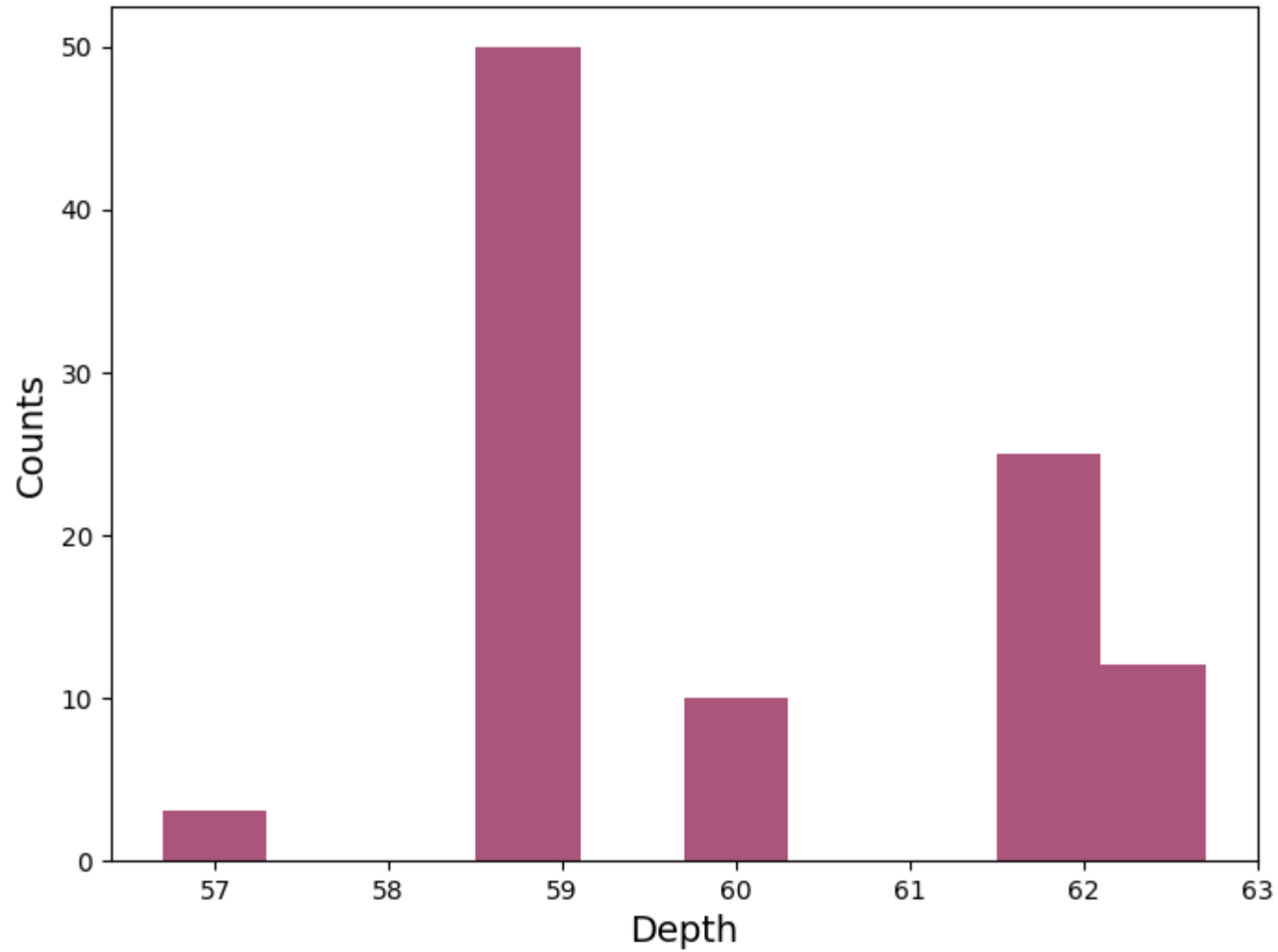
Esempio:



3

Routing
Stage

Esempio:



3

**Routing
Stage**

Decomporre i nostri circuiti nei gate base della macchina target e aggiungere gli SWAP gate fa aumentare velocemente la **profondità** del nostro circuito e il **numero di gate** presenti in esso.

Esistono però delle **routine per combinare ed eliminare i gate**, così da ottimizzare il circuito ove possibile.

Queste routine dipendono dal **livello di ottimizzazione** che scegliamo.

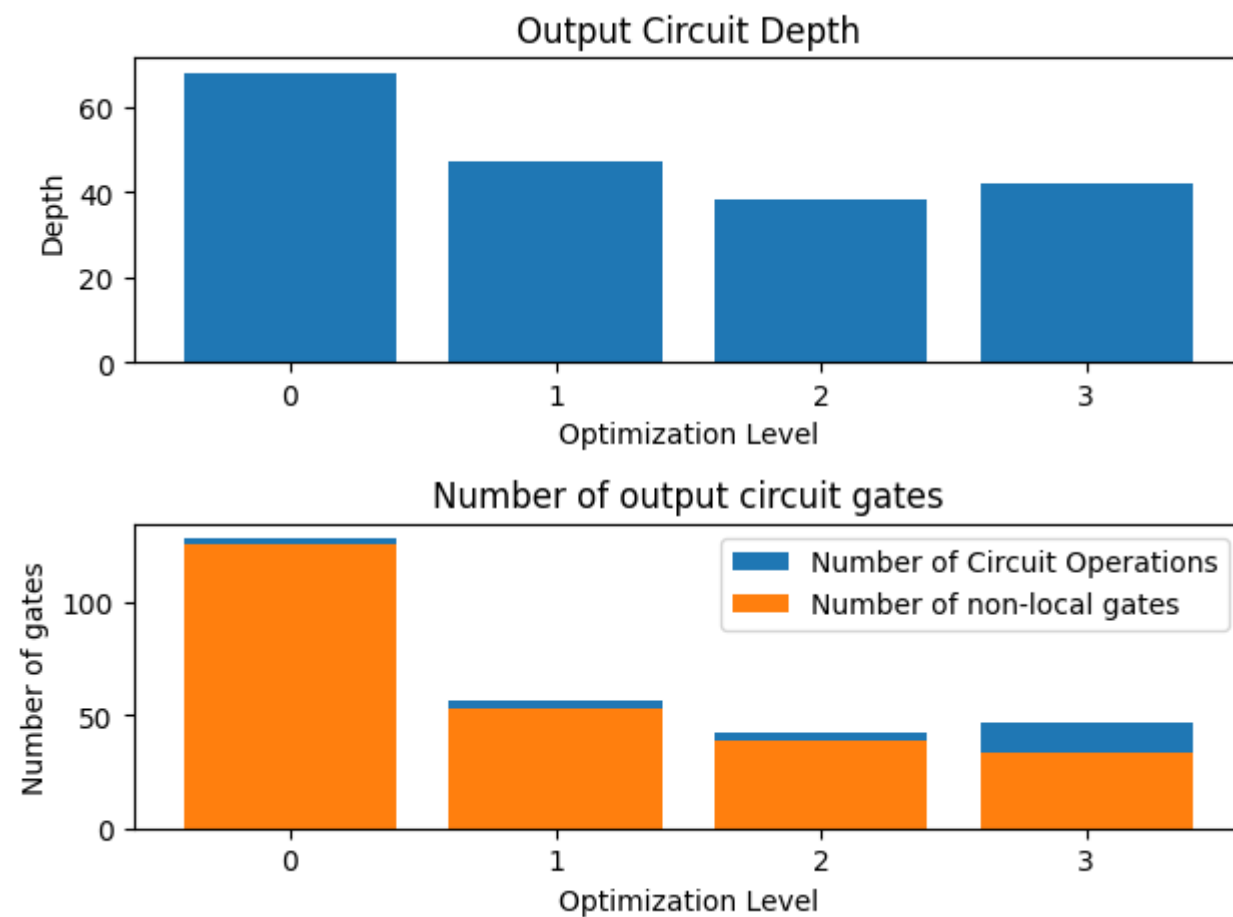
Se riprendiamo ad esempio le **rotazioni della QPE** della scorsa volta:



4

Optimization
Stage

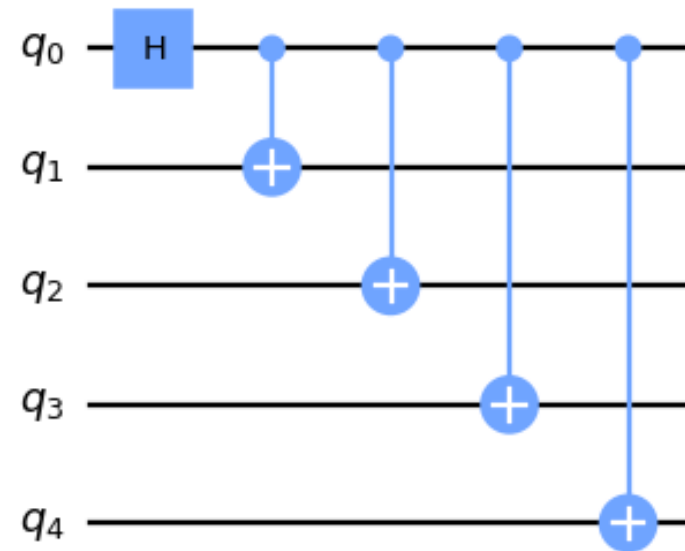
Riprendendo l'esempio dello step precedente:

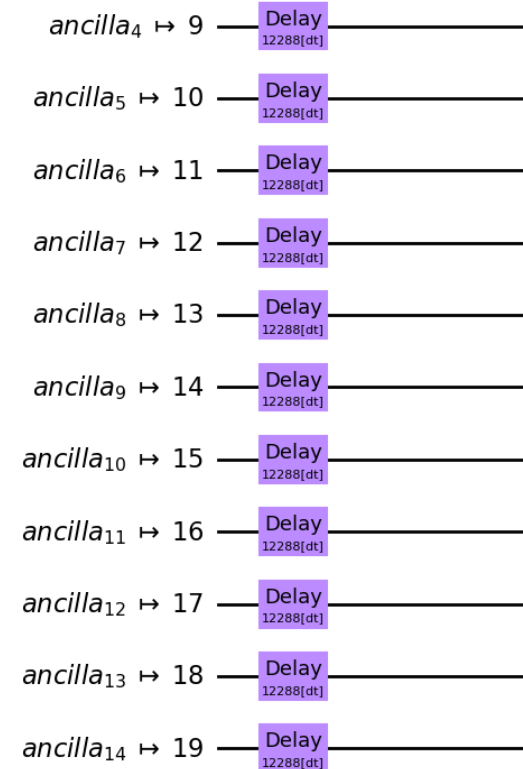
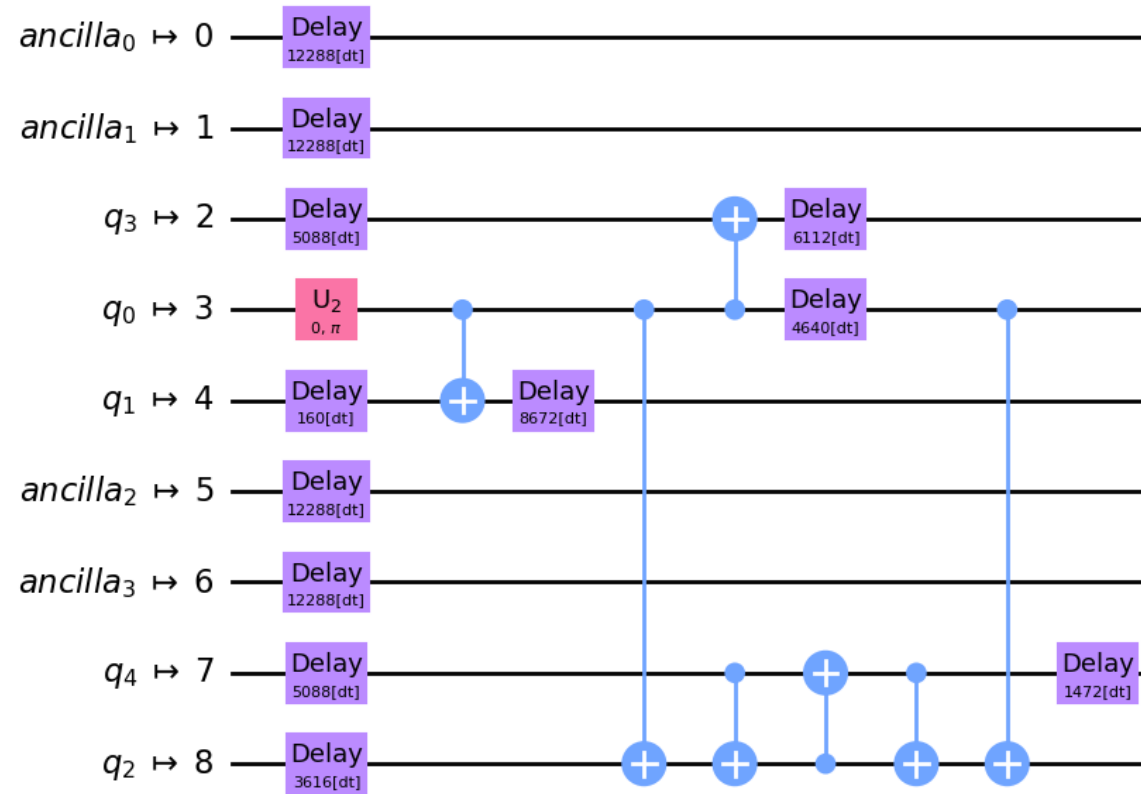


Dopo tutti questi stage, possiamo occuparci dello **scheduling** delle operazioni, così da tener conto del **tempo di inattività** nel circuito.

Possiamo vedere i periodi di inattività come a delle «operazioni» di **delay**, così da poterne tenere conto tra l'esecuzione delle istruzioni.

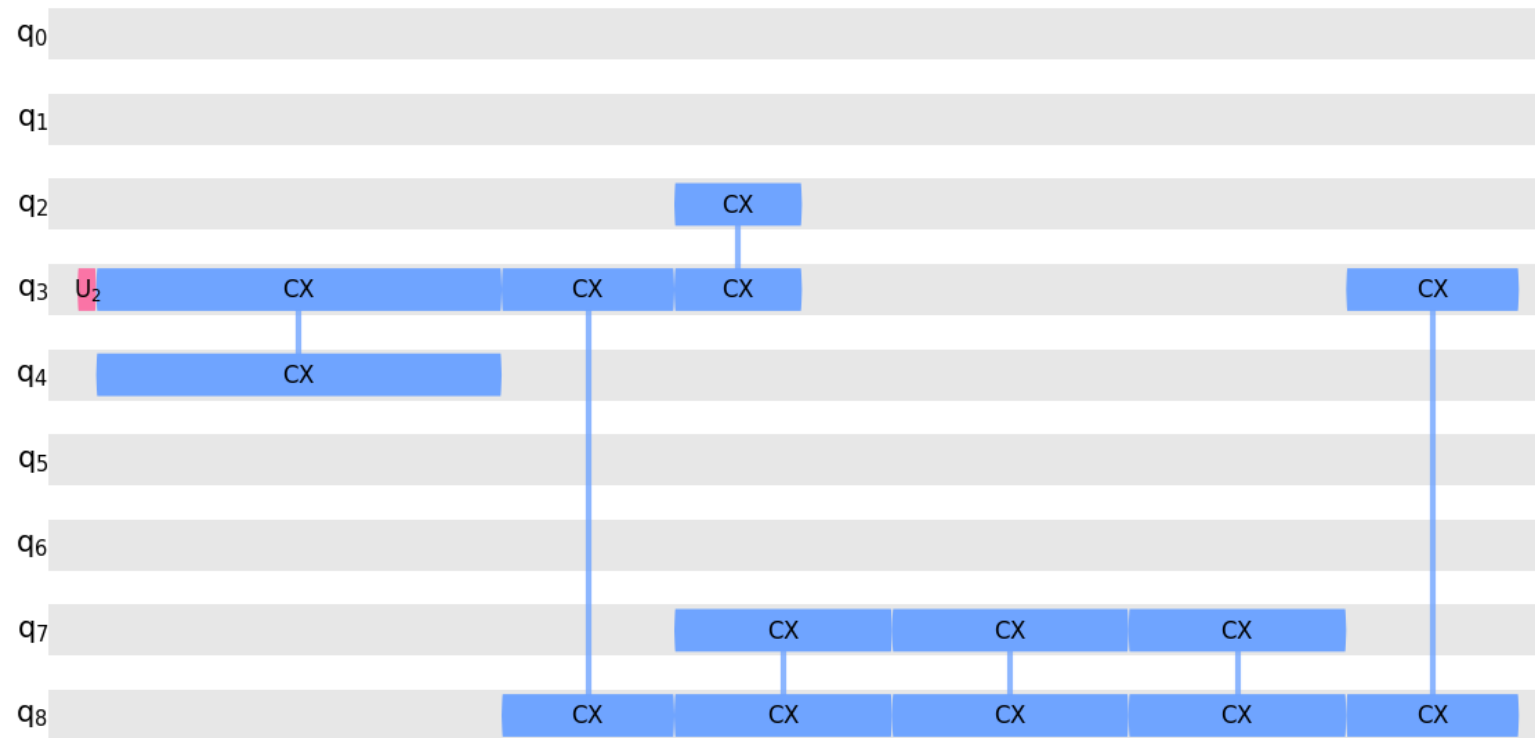
Ad esempio:





Scheduling Stage


Concentrandoci sui tempi di inattività:



Lo scheduling di un circuito si compone di tre parti:

- 1) **Analisi del circuito**, eseguita tramite i passaggi **ALAPSchedulingAnalysis** o **ASAPSchedulingAnalysis**, che tengono traccia dei tempi di inizio di ogni istruzione
- 2) **Mappatura dei vincoli**, eseguita tramite passaggi aggiuntivi che vengono eseguiti successivamente per tenere conto dei vincoli di macchina;
- 3) **Aggiunta dei delay** nel tempo di inattività di un qubit.

Per analisi più approfondite, vi rimandiamo alla documentazione del transpiler: [Qiskit Transpiler](#).



La realtà dei computer quantistici

Situazione Europea Attuale

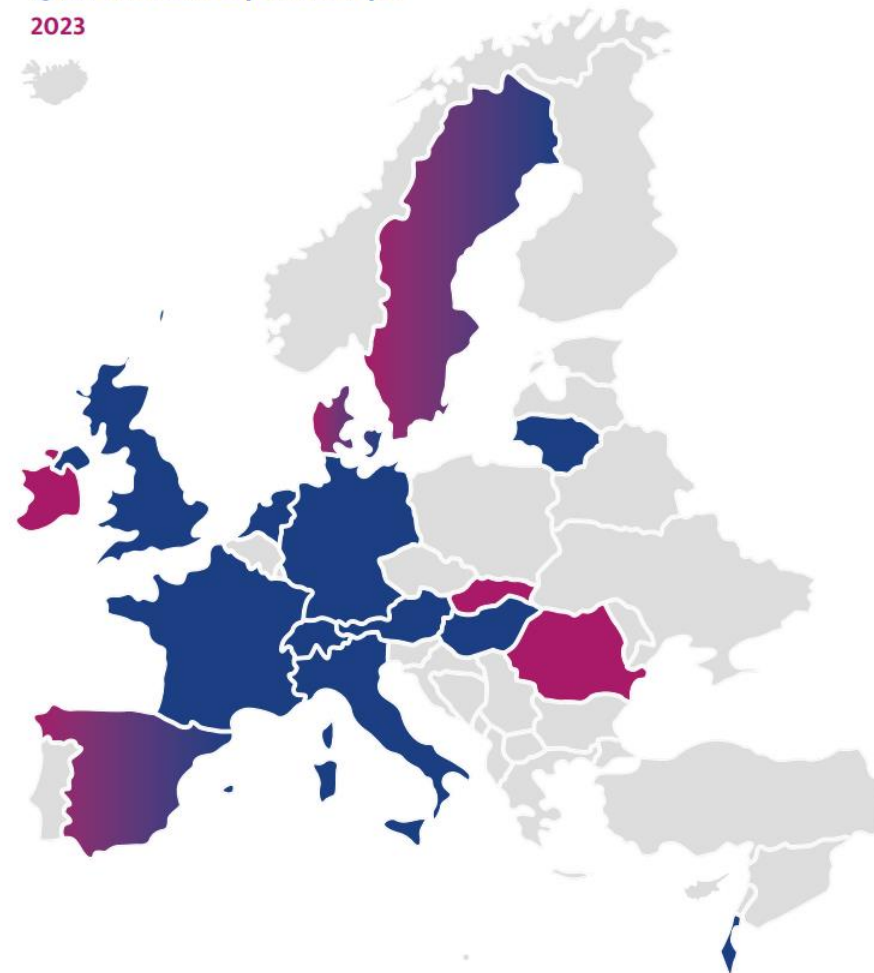
Quantum landscape in Europe

2020



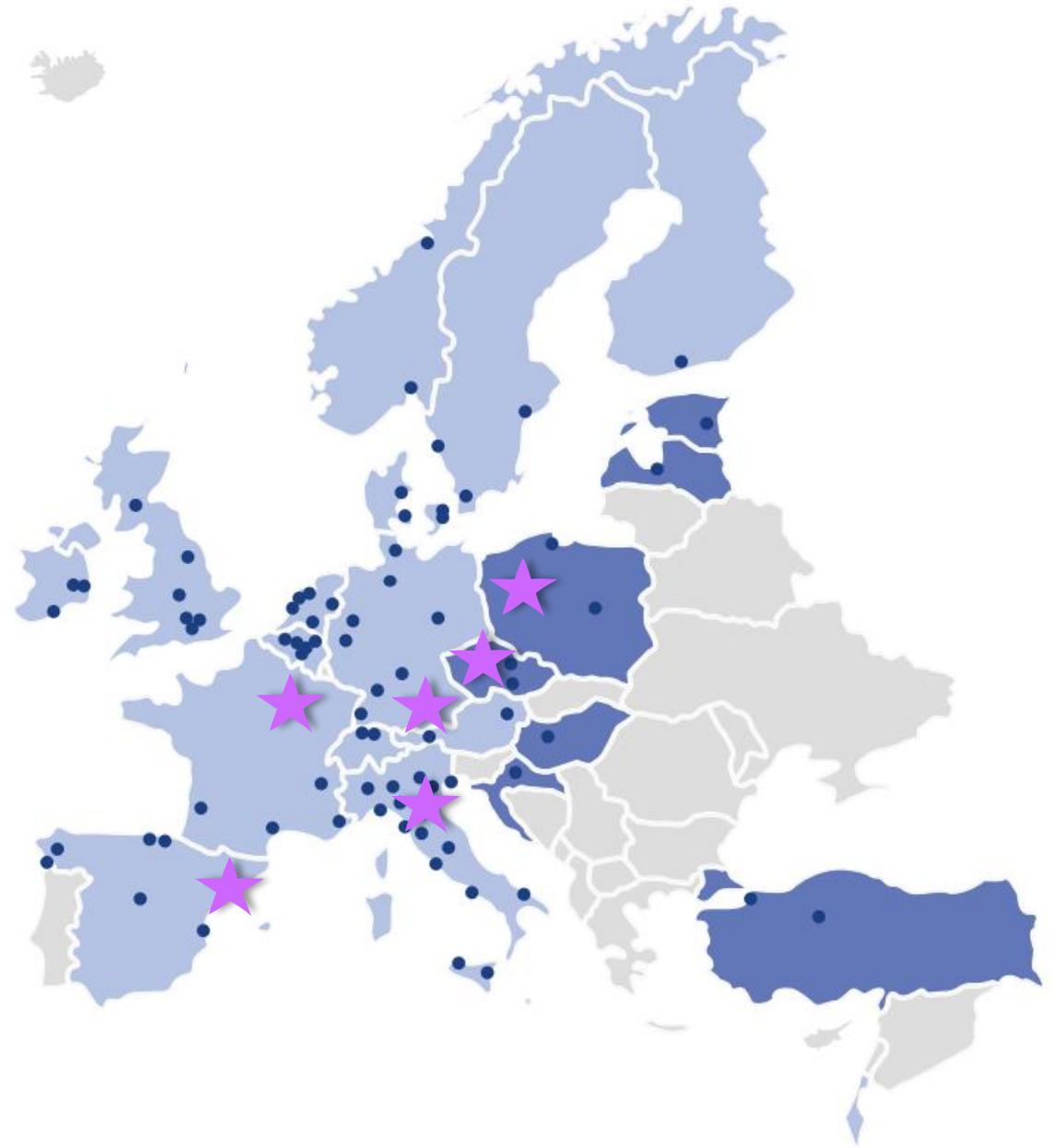
Quantum landscape in Europe

2023



Situazione Europea: Hubs

©Image courtesy of QuantERA, «Quantum Technologies
Public Policies Report», 2023



Focus on: Italia



- Nel 2022 il **PNRR** ha definito i seguenti obiettivi:
 - a) Far diventare l'Italia una dei protagonisti nell'ambito delle tecnologie quantistiche;
 - b) Rendere l'Italia indipendente dal punto di vista tecnologico;
 - c) Creare degli standard di misurazione universali e riproducibili;
 - d) Promuovere corsi interdisciplinari di educazione di alto livello nel settore.
- Le **principali aree di interesse** sono il Quantum Computing e le sue simulazioni, la Quantum Communication, Quantum Sensing and Metrology e la creazione di infrastrutture quantistiche.
- Si occupano di gestire i fondi e i progetti il **MIUR**, il **CNR**, l'**INFN** e il **NQSTI**.
- L'obiettivo principale è creare una **rete nazionale di ricerca in ambito quantistico**, sia da un punto di vista di capitale umano che di infrastrutture.

Come definire un computer quantistico?



- Tra il 1980 e il 1982, Paul Benioff pubblica una prima definizione di **macchina di Turing quantistica**, modello astratto per analizzare la computazione quantistica.
- Da un punto di vista più pratico, un qualsiasi computer quantistico deve rispettare i cosiddetti **Criteri di DiVincenzo**:
 - 1) Il sistema deve essere **scalabile** con un qubit ben definito;
 - 2) Il sistema deve poter essere **inizializzato** a uno stato di facile riproducibilità;
 - 3) Si devono avere lunghi **tempi di decoerenza**;
 - 4) Deve esistere un insieme di **gate universali** per il sistema;
 - 5) Deve esistere un'operazione di **misura**.
- Successivamente sono stati aggiunti **altri due criteri** riguardanti la comunicazione:
 - 1) Deve essere possibile passare le informazioni **dai qubit stazionari ai qubit mobili**;
 - 2) Si deve poter trasmettere fedelmente i qubit mobili **tra località diverse**.

IBM Quantum Computing

- L'IBM ha iniziato il suo progetto di ricerca nel settembre del **1990**, focalizzando i suoi sforzi nel costruire **computer scalabili** e sulla **divulgazione scientifica**.
- Sono disponibili **oltre 20 macchine** tra computer quantistici e simulatori, ognuna delle quali è dotata di caratteristiche specifiche e ottimali per problemi diversi.
- Il paradigma sottostante ad ognuna di queste macchine è comunque quello dei **superconduttori**. Le proprietà della macchina dipendono dalla topologia della QPU.
- Ad oggi, sono disponibili **6 diverse QPU**:



Osprey



Eagle



Hummingbird



Egret



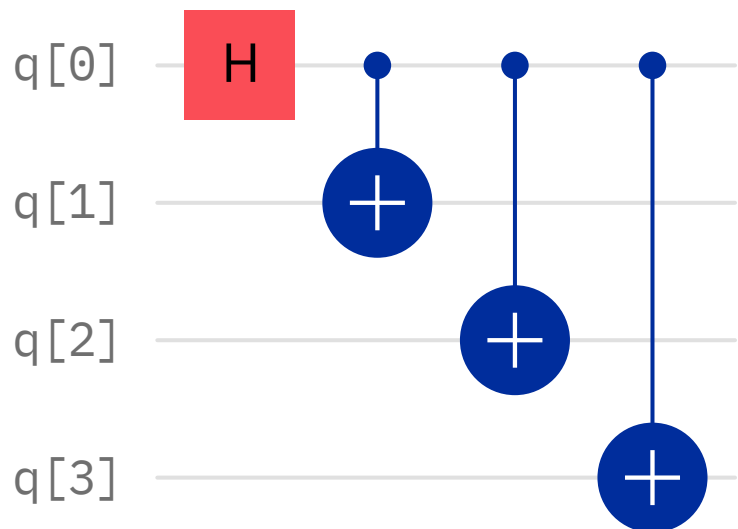
Falcon



Canary

IBM Quantum Computing

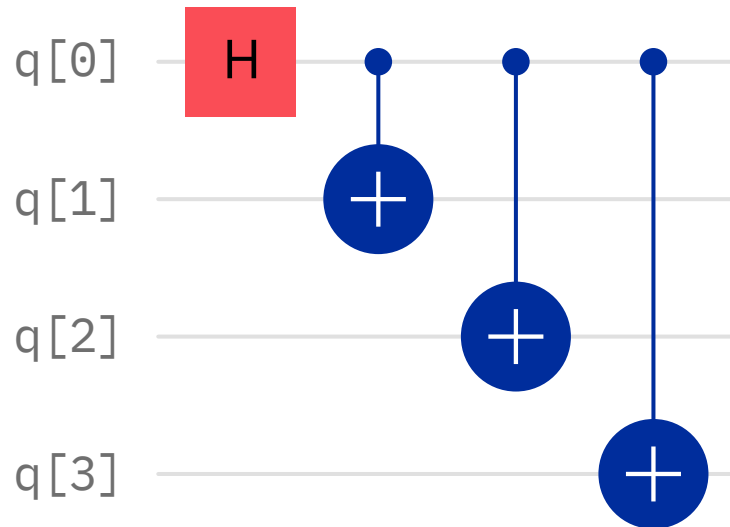
- L'IBM ha definito due approcci diversi per interagire con le loro macchine: il software **Qiskit** e il linguaggio **OpenQASM**.



```
from qiskit import QuantumRegister,  
    ClassicalRegister, QuantumCircuit  
from numpy import pi  
  
qreg_q = QuantumRegister(4, 'q')  
  
circuit = QuantumCircuit(qreg_q)  
  
circuit.h(qreg_q[0])  
circuit.cx(qreg_q[0], qreg_q[1])  
circuit.cx(qreg_q[0], qreg_q[2])  
circuit.cx(qreg_q[0], qreg_q[3])
```

IBM Quantum Computing

- L'IBM ha definito due approcci diversi per interagire con le loro macchine: il software **Qiskit** e il linguaggio **OpenQASM**.



```
OPENQASM 2.0;  
include "qelib1.inc";  
  
qreg q[4];  
h q[0];  
cx q[0], q[1];  
cx q[0], q[2];  
cx q[0], q[3];
```

Google

- Dal **maggio 2013**, anche Google ha preso parte alla corsa quantistica, costruendo la propria QPU sul modello basato sui superconduttori.
- La loro QPU conta 53 qubit e si chiama **Sycomore**.
- Anche Google ha sviluppato la sua personale libreria quantistica open source da usare con Python: tutta la documentazione di **Cirq** è disponibile online e sulla piattaforma Quantum AI sono presenti anche numerosi esempi.



Cirq

```
def grover_iteration(qubits, ancilla, oracle):  
    """Performs one round of the Grover  
    iteration."""  
    circuit = cirq.Circuit()
```

```
    # Create an equal superposition over input  
    qubits.
```

```
    circuit.append(cirq.H.on_each(*qubits))
```

```
    # Put the output qubit in the |-> state.
```

```
    circuit.append([cirq.X(ancilla), cirq.H(ancilla)])
```

```
    # Query the oracle.
```

```
    circuit.append(oracle)
```

```
    # Construct Grover operator.
```

```
    circuit.append(cirq.H.on_each(*qubits))
```

```
    circuit.append(cirq.X.on_each(*qubits))
```

```
    circuit.append(cirq.H.on(qubits[1]))
```

```
    circuit.append(cirq.CNOT(qubits[0], qubits[1]))
```

```
    circuit.append(cirq.H.on(qubits[1]))
```

```
    circuit.append(cirq.X.on_each(*qubits))
```

```
    circuit.append(cirq.H.on_each(*qubits))
```

```
    # Measure the input register.
```

```
    circuit.append(cirq.measure(*qubits, key="result"))
```

```
    return circuit
```

Quantum Supremacy



- Nel 2019, Google pubblica un articolo su Nature in cui sostiene di aver raggiunto la **supremazia quantistica** con la sua QPU Sycamore.
- Nel giro di poco tempo però la notizia è stata **smentita**: dai calcoli dell'IBM, sempre nel 2019, lo speedup ottenibile risultava molto più basso di quello presentato da Google.
- Inoltre, nel 2022, nuovi esperimenti classici tramite **supercomputer** hanno dimostrato inconfutabilmente che il paradigma classico risultasse molto meno lento rispetto a quanto osservato da Google, annullando la loro affermazione di essere arrivati alla supremazia quantistica.
- Nel 2020, il computer cinese **Jiuzhang** ha raggiunto la quantum supremacy, ma questo tipo di computer è altamente limitato negli algoritmi che può eseguire.

Microsoft Azure

- Nel 2010 Microsoft lancia la sua **piattaforma cloud** con servizi di cloud computing, tra cui il quantum computing.
- Non hanno una loro propria macchina, ma hanno **numerosi collaborazioni attive** che permettono l'utilizzo di diversi paradigmi.
- Sono supportati tutti i software più utilizzati, ossia **Q#**, **Qiskit** e **Cirq**.
- La loro piattaforma è **a pagamento**, ma possiamo vederne l'interfaccia con un semplice esempio a [questo link](#).



Amazon Braket

- Nel 2022 Amazon Web Services lancia **Amazon Braket**, fornendo ai suoi clienti l'accesso a diverse tecnologie quantistiche tramite una sua interfaccia e un suo software dedicati.
- La piattaforma è **a pagamento**. Alcuni esempi sono disponibili nella **cartella GitHub** di Amazon Braket o seguendo i loro workshop.
- I suoi **fornitori** sono:



Amazon Braket

Il software si chiama, per l'appunto, **Braket** ed è una libreria disponibile in **Python**.

```
# define three-qubit CCZ gate
ccz_gate = np.array([[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0]],
                    dtype=complex)

# instantiate circuit object
circ = Circuit()

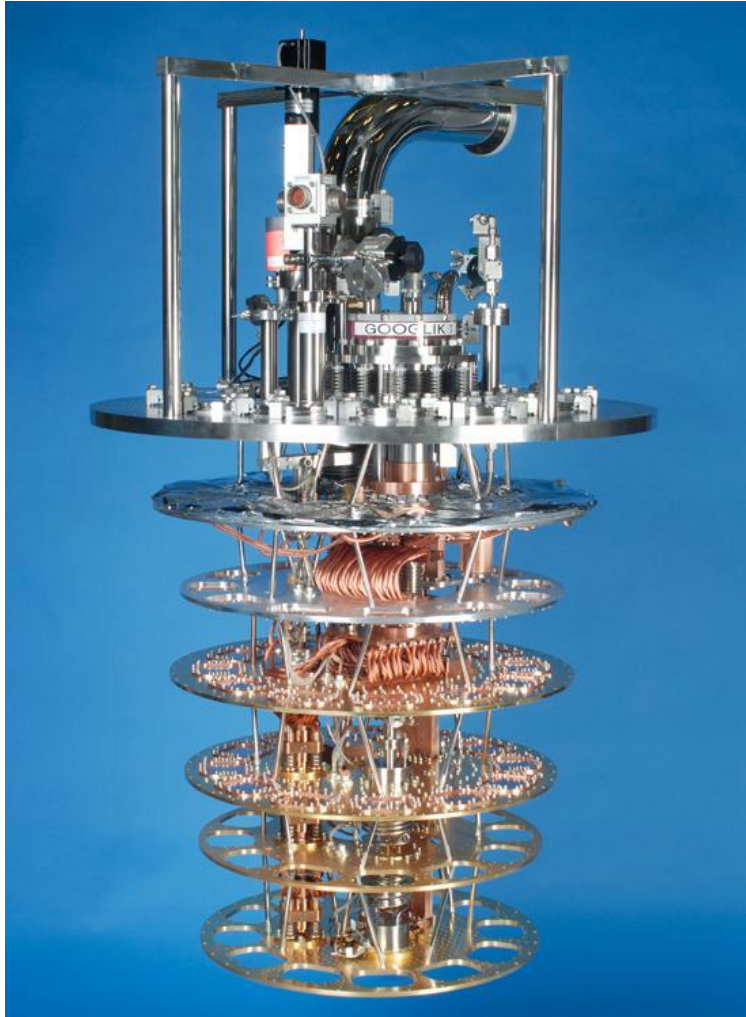
# add CCZ gate
circ.unitary(matrix=ccz_gate, targets=targets)
```

Atom Computing

- Atom Computing è una start-up californiana che il 24 ottobre 2023 ha annunciato di aver costruito una macchina gate-based con **1180 qubit**.
- Il paradigma utilizzato è quello dei **neutral atoms**, gestiti tramite l'utilizzo di laser e altri metodi ottici, e supporta algoritmi scritti in Qiskit e OpenQASM.
- L'utilizzo di questo tipo di atomi è una delle **ultime frontiere** nell'ambito del quantum computing e proprio di recente è stato pubblicato un articolo che ne analizza la performance, presentando risultati di **fedeltà fino al 95.5%** per gate a due qubit.
- Al momento c'è molto fermento riguardo la notizia, ma per capire se sia davvero il successo prospettato bisognerà attendere **pubblicazioni** ed **esperimenti**.

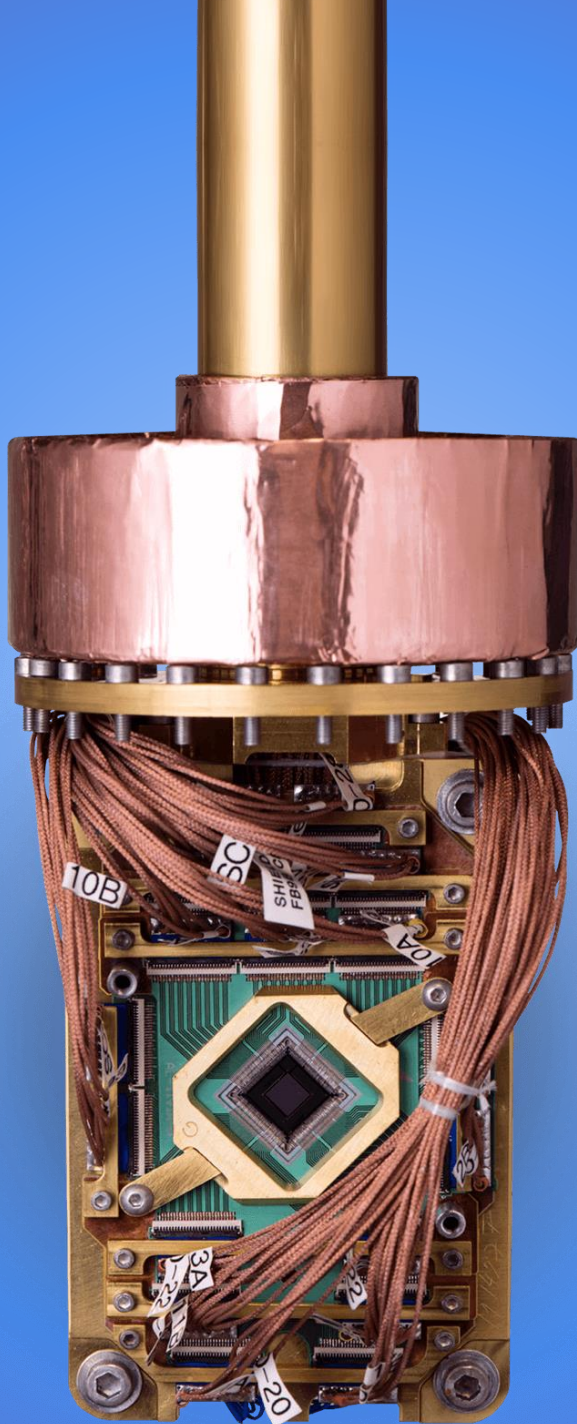


Sfide attuali: decoerenza e rumore



- Uno dei problemi principali che si riscontra quando si costruisce un computer quantistico è la **decoerenza**, ossia il collasso di uno stato quantistico a uno classico.
- Se non perfettamente isolato, il sistema **interagisce con l'ambiente** e perde coerenza.
- Si cerca dunque di costruire computer quantistici isolati, ma **altri fattori** possono creare decoerenza, come ad esempio i gate o le vibrazioni del reticolo fisico.
- Inoltre, tutti i sistemi quantistici sono **rumorosi**, per loro stessa definizione, il che porta ad errori nell'esecuzione dell'algoritmo.

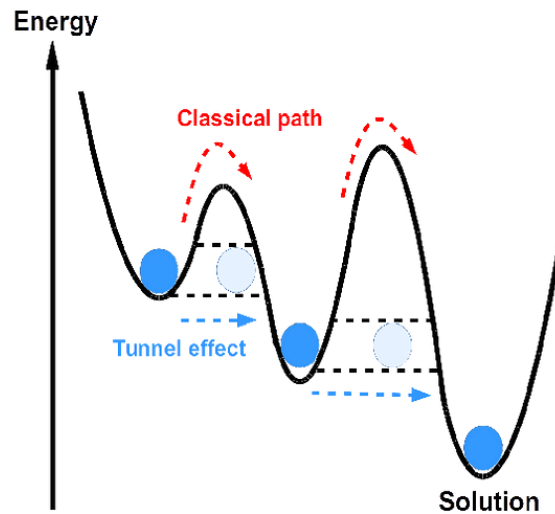
Quantum



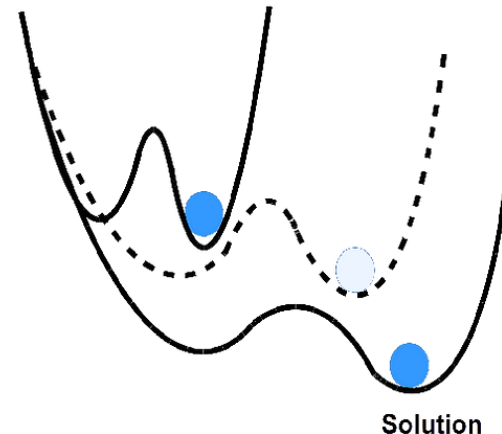
Annealing

Introduzione

- Il **Quantum Annealing** è un processo di ottimizzazione il cui fine è trovare il minimo globale di una funzione obiettivo all'interno di uno spazio di configurazioni possibili.
- In particolare, codificheremo sempre i nostri problemi come **energia del sistema** e il nostro obiettivo sarà trovare il **ground state** dell'Hamiltoniana associata.
- Ci sono molte analogie sia con il **Simulated Annealing** che con **l'Adiabatic Quantum Computation**.



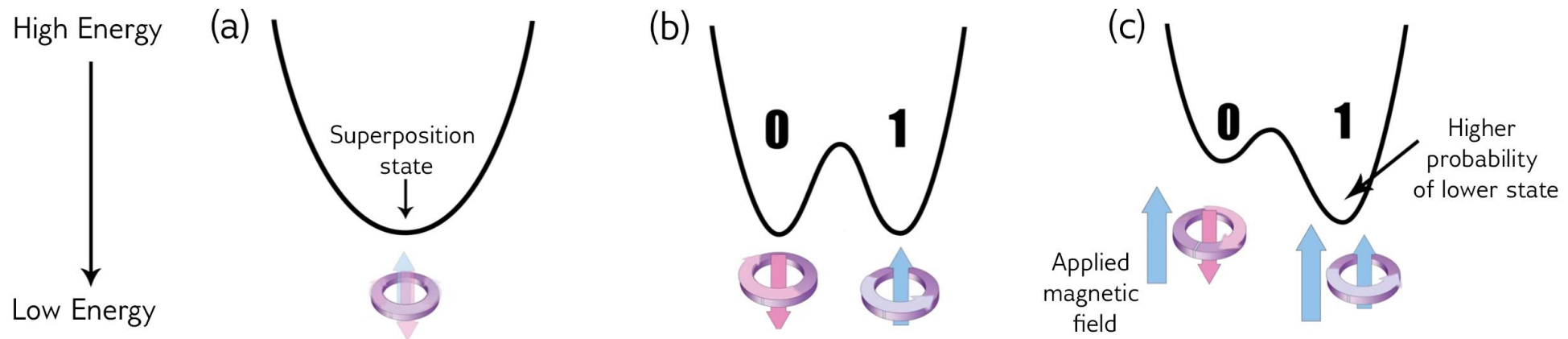
Quantum Tunnelling



Adiabatic evolution

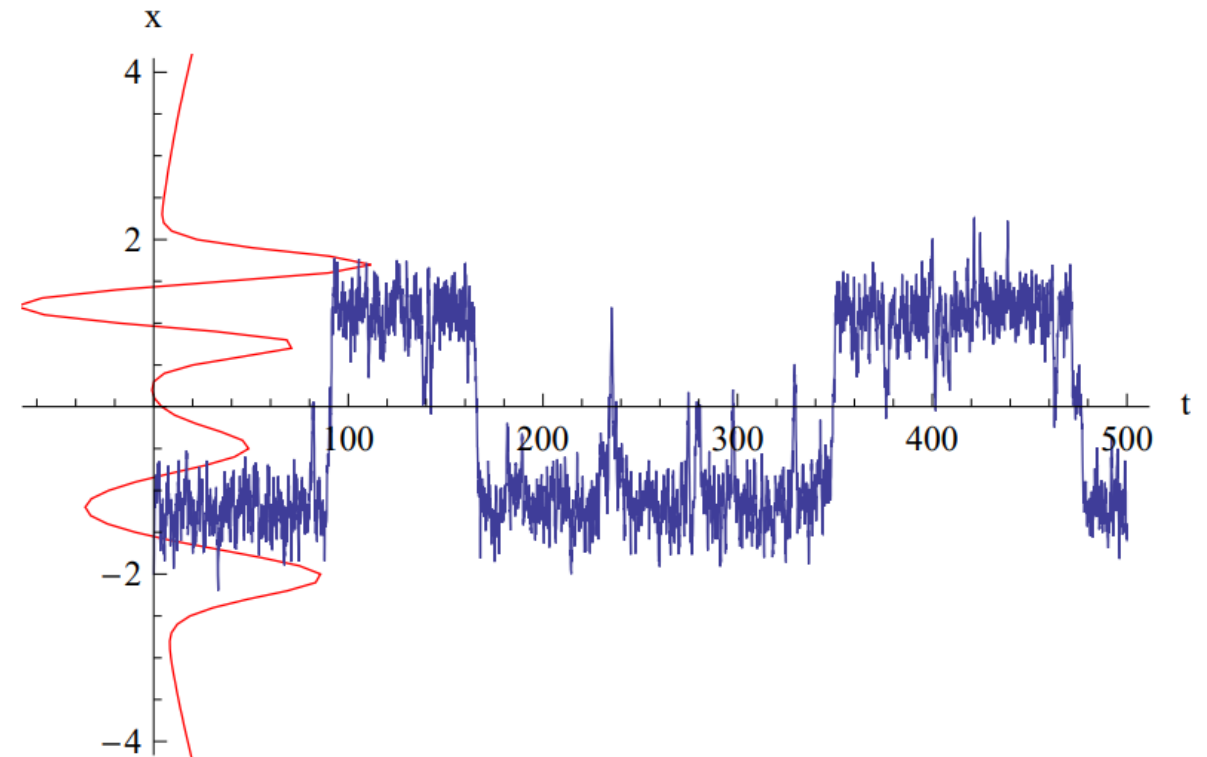
Fenomeno fisico

- Il sistema parte da una **sovrapposizione equiprobabile** di tutti i possibili stati e poi evolve secondo **l'equazione di Schrödinger**.
- Ad ogni passo, le ampiezze vengono modificate tramite l'applicazione di un **campo magnetico**, rendendo più probabile di trovarsi in una delle soluzioni e andando ad eliminare gli stati non soluzione.
- Tale campo provoca il **tunneling** tra le valle dei minimi.

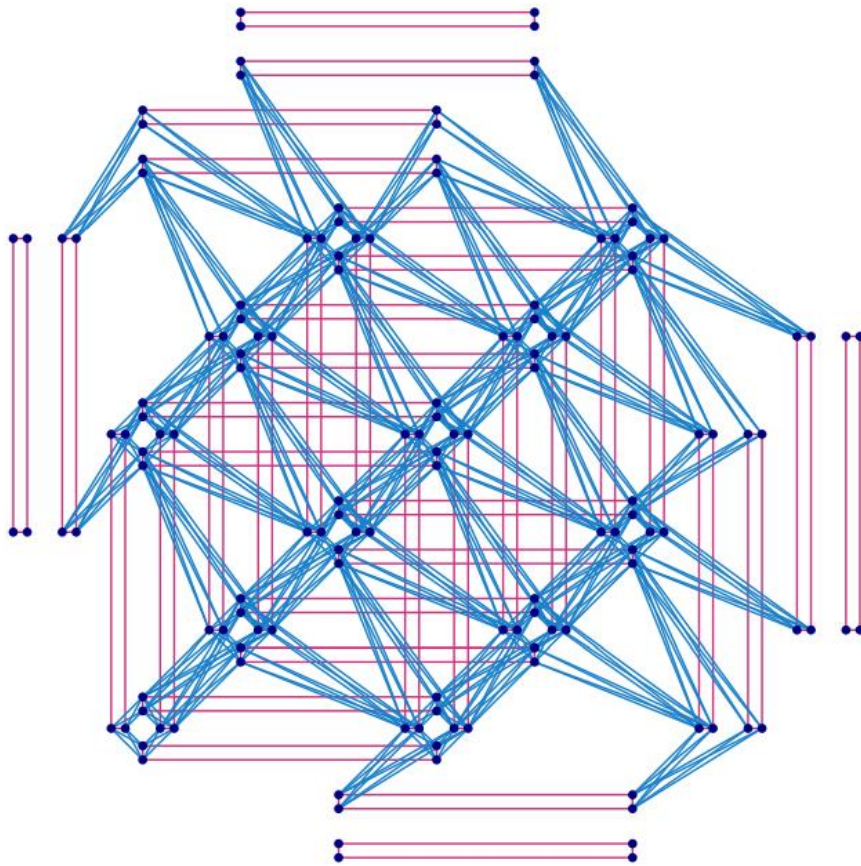


Fenomeno fisico/intro

- È possibile dimostrare matematicamente la correttezza del risultato tramite QA, ma noi riportiamo solo **l'idea generale**.
- Seguendo la **posizione** di un'ipotetica particella nell'evoluzione del sistema otteniamo il grafo in blu riportato qui di lato.
- La linea rossa rappresenta **l'energia potenziale** del sistema, di cui vogliamo il minimo.
- Per la giusta scelta di parametri, si ottiene che il QA si comporta come un **processo di Markov discreto** definito sulle soluzioni.



D-Wave



- La **D-Wave Quantum Systems Inc.** è una compagnia canadese fondata nel 1999. È al momento **la migliore azienda** sul mercato ad occuparsi di Quantum Annealing.
- Lanciano la loro prima QPU a 128 qubit e in meno di 5 anni arrivano a un computer di 1000 qubit. Dagli ultimi aggiornamenti, la loro ultima macchina ha **oltre 5000 qubit**.
- Le **applicazioni industriali** di questo approccio stanno dando ottimi risultati, come anche le collaborazioni e le ricerche scientifiche.
- I processori si possono distinguere soprattutto per la loro diversa topologia: partendo dalla passata **Chimera** fino alla futura **Zephyr**, passando per l'attuale **Pegasus**.

D-Wave

	2000Q	Advantage	Advantage <i>performance update</i>
Performance			
Better Solutions (Satisfiability problems)	--	3x more often than 2000Q	23x more often than 2000Q
Time-To-Solution (3D lattice problems)	--	10x faster than 2000Q	2x faster than Advantage
Annealing Quantum Processor Design			
Qubits	2000+	5000+	5000+
Couplers	6000+	35000+	35000+
Couplers Per Qubit	6	15	15
Topology			
Graph	Chimera	Pegasus	Pegasus
Graph Size	C16	P16	P16
Connectivity	Degree 6	Degree 15	Degree 15
Lattice	8x8x8	15x15x12	15x15x12
Chain Length (for problem size n=64)	17	7	6

D-Wave

- La libreria progettata dalla D-Wave si chiama **Ocean** ed è associata anch'essa a Python.
- Oltre alle QPU, sono presenti anche numerosi **risolutori classici e ibridi**, permettendo un'applicazione più ampia.
- Numerosi sono gli esempi presenti sul loro GitHub, facilmente consultabili ed eseguibili anche sulla loro piattaforma **Leap**.
- D-Wave permette di avere un account **completamente gratuito** (limitato) oppure a pagamento.

The screenshot displays the D-Wave Systems Examples page, which features four example cards arranged in a 2x2 grid. Each card includes a title, a brief description, a set of tags (Advanced, Optimization, Intermediate, Code Example, +2 More), a Jupyter Notebook icon, and a star rating. The cards are as follows:

- Decoding Cellphone Signals**: Use a quantum computer to decode cellphone signals. Tags: ADVANCED, OPTIMIZATION, JUPYTER NOTEBOOK. D-Wave Systems Examples 2023-07-25. 6 stars.
- Tour Planning**: Use a hybrid CQM solver to optimize the modes of locomotion for a multi-leg tour. Tags: OPTIMIZATION, CODE EXAMPLE, INTERMEDIATE, +2 MORE. D-Wave Systems Examples 2022-09-19. 4 stars.
- Not-All-Equal 3-Satisfiability (NAE3SAT)**: Compare the performance of the Advantage QPU and the Advantage2 prototype on not-all-equal 3-satisfiability problems. Tags: ADVANCED, OPTIMIZATION, CODE EXAMPLE, +2 MORE. D-Wave Systems Examples 2022-05-17. 1 star.
- 3D Bin Packing**: Use a hybrid solver to use the minimum number of bins to pack items with different dimensions. Tags: OPTIMIZATION, CODE EXAMPLE, INTERMEDIATE, +2 MORE. D-Wave Systems Examples 2022-03-07. 18 stars.

Formulazione del problema

- I problemi risolvibili tramite QA sono quelli che è possibile scrivere come un **Binary Quadratic Model** (BQM), ossia come problema **QUBO** o come modello di **Ising**.
- La forma generale di un BQM è

$$\min \left(\sum_i a_i v_i + \sum_i \sum_{j>i} b_{i,j} v_i v_j + c \right)$$

- Tale forma quadratica dovrà poi essere mappata nella **topologia della macchina**:

$$H(a, b) = 5a + 7ab - 3b \quad \text{---} \dots \text{---} \rightarrow \begin{array}{c} \boxed{a} \text{---} 7 \text{---} \boxed{b} \\ 5 \qquad \qquad -3 \end{array}$$

- Ocean ha una sua funzione dedicata per effettuare tale **minor embedding**.

Formulazione del problema

- La **formulazione QUBO** è analoga, solo che le variabili sono definite sul campo binario:

$$\min \left(\sum_i a_i x_i + \sum_i \sum_{j>i} b_{i,j} x_i x_j + c \right)$$

- Possiamo sempre mappare una formulazione QUBO in una **matrice**:

$$x_1 + 2x_2 + 3x_3 + 12x_1x_2 + 13x_1x_3 + 14x_2x_4 \longrightarrow \begin{bmatrix} 1 & 12 & 13 & 0 \\ 0 & 2 & 0 & 14 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Formulazione del problema

- Il **modello di Ising** ha invece la seguente forma, con variabili in $\{-1, +1\}$:

$$\min \left(\sum_i h_i s_i + \sum_i \sum_{j>i} J_{i,j} s_i s_j \right)$$

- Possiamo mappare tale formulazione in un **vettore** e una **matrice**:

$$s_1 + 2s_2 + 3s_3 + 12s_1s_2 + 13s_1s_3 + 14s_2s_4$$

$$J = \begin{bmatrix} 0 & 12 & 13 & 0 \\ 0 & 0 & 0 & 14 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$h = [1 \quad 2 \quad 3 \quad 0]$$

Formulazione del problema

- Le due formulazioni sono **equivalenti**. Possiamo passare indifferentemente dall'una all'altra con dei semplici cambi di variabile:

$$x_i \mapsto \frac{s_i + 1}{2}$$

$$s_i \mapsto 2x_i - 1$$

- Nonostante tali modelli di minimizzazione siano «**non vincolati**» per definizione, è possibile risolvere problemi vincolati includendo il vincolo nella definizione della funzione di costo:

$$\begin{aligned} \min y &= x^t C x \\ Ax &= b, \ x \text{ binary} \end{aligned}$$



$$\begin{aligned} y &= x^t C x + P(Ax - b)^t (Ax - b) \\ &= x^t C x + x^t D x + c \\ &= x^t Q x + c \end{aligned}$$