

Università degli Studi di Milano-Bicocca

DIPARTIMENTO DI MATEMATICA E APPLICAZIONI

Corso di Laurea Triennale in Matematica



Introduzione alla computazione quantistica

Relatore
Prof. Davide Luigi Ferrario

Candidato
Andrea Panontin
Matricola
813475

Sessione di Laurea di settembre
Anno Accademico 2018–2019

Indice

1	Introduzione	3
1.1	Algebra Lineare Complessa	4
1.1.1	Notazione di Dirac	4
1.1.2	Operatori e prodotto esterno	5
1.1.3	Prodotto Tensore	6
1.1.4	La base computazionale	7
1.2	I postulati della meccanica quantistica	8
1.2.1	La sfera di Bloch	10
1.3	Gate quantistici	11
1.4	Circuiti quantistici	13
1.5	Paragone con la computazione classica	16
2	Complessità computazionale	19
2.1	Macchine di Turing	19
2.1.1	Codifica	21
2.2	Problemi decisionali	21
2.3	Richieste computazionali	22
2.4	Classi di complessità	24
2.4.1	Tempo deterministico e P	24
2.4.2	Classe NP e tempo non deterministico	25
2.4.3	Relazione tra P ed NP	28
2.5	Classi di complessità quantistica	28
2.5.1	Circuiti Booleani e complessità	28
2.5.2	Circuiti quantistici	30
3	Algoritmi quantistici	32
3.1	Fenomeni quantistici	32
3.1.1	Stati di Bell	32
3.1.2	Teletrasporto quantistico	33
3.1.3	Superdense coding	35
3.1.4	Parallelismo quantistico	36
3.1.5	Algoritmo di Deutsch-Josza	38
3.2	Trasformata di Fourier quantistica	40
3.2.1	Stima di fase	44
3.2.2	Individuazione dell'ordine	48
3.2.3	Fattorizzazione degli interi	55
3.2.4	Logaritmo discreto	58

A	Frazioni continue	63
B	Crittografia	68
B.1	Crittografia a chiave pubblica	68
B.1.1	Sistema RSA	68
B.1.2	Bucare RSA	70

Sommario

Negli ultimi decenni si sono registrati importanti sviluppi nel campo dell'informatica quantistica, branca di ricerca nata a partire dall'idea di Feynman di sfruttare sistemi quantistici per eseguire computazioni. Recentemente, grazie alla grande ricerca tecnologica, si è vista la realizzazione dei primi computer quantistici. Parallelamente la ricerca teorica nel campo della computazione quantistica ha portato allo sviluppo di svariati algoritmi con richieste computazionali minori rispetto ai corrispettivi classici, in alcuni casi addirittura in maniera esponenziale.

Alcuni algoritmi quantistici riescono a risolvere in modo efficiente – ovvero con una richiesta di operazioni che dipende in modo polinomiale dalla dimensione del dato in input – problemi considerati “difficili” – ovvero non risolubili in modo efficiente – nell'ambito della computazione classica. Questo, dal punto di vista pratico, potrebbe minare la sicurezza delle più diffuse implementazioni di sistemi di crittografia a chiave pubblica. Dal punto di vista teorico, invece, mette in dubbio la veridicità della tesi forte di Church-Turing, la quale afferma che, per ogni problema risolubile in modo efficiente su un arbitrario sistema computazionale, si possa trovare un algoritmo classico efficiente. Risultati in questa direzione porterebbero ad approfondire le conoscenze necessarie per affrontare il problema del millennio **P** vs **NP**, potenzialmente accentuando l'importanza del paradigma quantistico nella teoria della complessità.

L'elaborato, traendo spunto da [NC11] per la parte propriamente di computazione quantistica e da [San09] e [Pap94] per la parte riguardante la complessità computazionale, ha l'obiettivo di spiegare il funzionamento di alcuni noti algoritmi quantistici, dando gli strumenti necessari a comprenderne l'importanza dal punto di vista della teoria della complessità computazionale.

Inizia, nel primo capitolo, introducendo le basi di algebra lineare e di meccanica quantistica necessarie per poter definire circuiti e algoritmi quantistici. Vengono, inoltre, descritti il funzionamento di un computer quantistico e alcune differenze del nuovo paradigma di computazione da quello classico, in particolare mostrando operazioni non implementabili su un substrato quantistico.

Nel secondo capitolo si introducono gli strumenti alla base della complessità computazionale, che saranno utilizzati per analizzare le richieste degli algoritmi quantistici e i vantaggi rispetto alle controparti classiche. A tal fine viene introdotto il concetto di classe di complessità e ne vengono dati esempi, tra cui le celeberrime **P** ed **NP**, per dare un contesto alla teoria citata.

Nel terzo capitolo si analizzano i primi algoritmi, inizialmente mostrando semplici applicazioni del comportamento quantistico dell'hardware, tra cui il parallelismo e il teletrasporto quantistici, che stanno alla base dei grandi vantaggi computazionali rispetto alla computazione classica. Il lavoro finisce con la descrizione del funzionamento di una lista di algoritmi che sfruttano le peculiarità del nuovo paradigma computazionale per ottenere una riduzione nelle richieste computazionali, a volte esponenziale, rispetto alle più efficienti controparti classiche. Tra questi figurano gli importantissimi algoritmi per l'individuazione dell'ordine, per la fattorizzazione degli interi e per il calcolo del logaritmo discreto, descritti per la prima volta da Peter Shor in [Sho97].

Ringraziamenti

Ringrazio innanzitutto la mia famiglia, che mi ha sempre supportato nel corso degli studi. Dedico un grande ringraziamento alla prof.ssa Maria Elena Jary e al prof. Franco Vergani, le cui lezioni mi hanno fatto intravedere la bellezza della matematica e mi hanno portato a seguire questo bellissimo percorso didattico.

Ringrazio i miei compagni di corso, per avermi sopportato e supportato in questi anni, condividendo la grande passione per la matematica. Un ringraziamento va anche a tutti i professori che, durante tutti gli scorsi tre anni, mi hanno mostrato nuovi ambiti della materia, facendomi appassionare a temi di cui non sarei stato altrimenti in grado di scoprire la bellezza.

Infine desidero ringraziare il mio relatore, prof. Davide Luigi Ferrario, il quale mi ha guidato nella stesura di questo documento, prima manifestazione di indipendenza didattica.

Struttura dell'elaborato

Il seguente elaborato vuole essere un'introduzione all'argomento della computazione quantistica. Per questo motivo presuppone semplicemente conoscenze di algebra lineare e analisi matematica di base, presenti in ogni curriculum di matematica o fisica. Si presuppone, inoltre, che il lettore abbia un'educazione matematica, non necessariamente fisica, quindi si dedica una grande mole di spazio per introdurre concetti basilari di meccanica quantistica, presentati in qualsiasi corso su tale argomento. Questo non è un invito, per il lettore esperto nella materia sopracitata, ad evitare di leggere le parti relative alla meccanica quantistica. Esse, infatti, sono riprese in ottica computazionale, con alcune variazioni rispetto alla usuale trattazione fisica.

L'opera si divide in tre capitoli, un primo il cui fine è di descrivere il funzionamento di un computer quantistico, un secondo che vuole trattare la complessità computazionale e, infine, un terzo in cui sono descritti una famiglia di algoritmi quantistici. È chiara la forte dipendenza logica del terzo capitolo dal primo. Il funzionamento degli algoritmi, invece, non richiede conoscenze di complessità computazionale per essere compreso.

La dipendenza dei capitoli è, quindi, la seguente: Il primo e il secondo capitolo risultano, sostanzialmente, indipendenti (a meno dell'ultima sezione del secondo capitolo, che risulta più chiara in luce di una lettura del primo). Il terzo capitolo, principalmente, ha come scopo quello di descrivere il funzionamento degli algoritmi, obiettivo per il quale è necessario possedere le conoscenze descritte nel primo capitolo. Il secondo capitolo è necessario, nel terzo capitolo, solo per chiarificare la nozione di richieste computazionali, usate per descrivere la velocità di esecuzione dell'algoritmo descritto.

Capitolo 1

Introduzione

La computazione quantistica rappresenta un paradigma computazionale fondamentalmente diverso dall'analogo classico. Innanzitutto si sostituisce il bit con il qubit, un ente più complesso, con maggiori potenzialità, ma anche grandi restrizioni al proprio utilizzo. I gate utilizzabili nella computazione saranno altrettanto diversi dalle controparti classiche, con conseguenti peculiarità nel *modus operandi* del computer quantistico. Tutto ciò è dovuto alle particolarità della meccanica quantistica e dalle sue differenze dalle teorie classiche. Per questo motivo, inoltre, si noterà una computazione intrinsecamente probabilistica, poiché è tale il comportamento delle misure quantistiche, usate per ottenere i risultati della computazione salvati nei qubit.

Queste peculiarità non sono senza pregi. Renderanno possibili fenomeni inimmaginabili dal punto di vista classico. Per citarne alcuni, introdotti nel capitolo 3, si vedrà la possibilità di computare tutte le immagini di una funzione con l'esecuzione di un solo circuito quantistico – indipendentemente dal numero di immagini da calcolare. Si studierà un canale di comunicazione quantistico – realizzato anche in pratica con buone tolleranze di errore – in cui lo scambio di un solo qubit permetterà lo scambio di 2 bit classici. Si vedranno, infine, algoritmi esponenzialmente più veloci rispetto alle migliori controparti classiche esistenti.

Per arrivare a comprendere tutti questi affascinanti fenomeni è necessario possedere un lessico matematico che spesso non viene fornito nei curricula universitari. A questo scopo verrà dedicato questo capitolo, il cui fine è quello di fornire tutti gli strumenti necessari per lavorare con la computazione quantistica.

In particolare l'introduzione che segue è fortemente ispirata alla trattazione di [Gud03], con aggiunte tratte da [NC11]. L'organizzazione di questa sezione rispetta la prossima descrizione. In primis viene introdotta l'algebra lineare complessa, ovvero il lessico usato per descrivere la meccanica e la computazione quantistica. In particolare si usa la notazione di Dirac, standard nella letteratura. Segue una veloce trattazione dei postulati della meccanica quantistica, che governano il comportamento di un computer quantistico. Infine si introduce più propriamente la computazione tramite gate e circuiti quantistici. Il capitolo termina sfruttando questi strumenti per dare una prima visione generale delle peculiarità della computazione quantistica, in particolare tramite il paragone con la controparte classica.

1.1 Algebra Lineare Complessa

Per comprendere il resto della trattazione è necessaria una conoscenza dell'algebra lineare complessa di base, della quale si dà una breve descrizione. Seguendo la convenzione nell'ambito della computazione quantistica, ereditata dalla meccanica quantistica, si userà la notazione di Dirac. La seguente sezione va interpretata come un'introduzione alla notazione e al lessico, sfruttata per ripassare i concetti fondamentali, ai fini della computazione quantistica, della materia.

1.1.1 Notazione di Dirac

Nell'ambito della computazione quantistica si è interessati principalmente allo spazio di Hilbert $V = \mathbb{C}^n$. In particolare gli stati di una computazione saranno vettori unitari in tale spazio. Come in letteratura, i vettori in tale spazio saranno indicati con la notazione *ket* di Dirac:

$$|v\rangle := (v_1, \dots, v_n) \quad \text{con } v_i \in \mathbb{C} \quad \forall i \in \{1, \dots, n\}$$

dove (v_1, \dots, v_n) è da intendersi come un vettore $n \times 1$ in colonna, in cui gli v_i sono le componenti del vettore v rispetto alla base canonica, la cui nomenclatura verrà spiegata nella sezione 1.1.4

$$|1\rangle := (1, 0, \dots, 0), \dots, |n\rangle := (0, \dots, 0, 1).$$

Sempre rispetto alla stessa base, si introduce la notazione *bra*, per il duale

$$\langle v| := |v\rangle^\dagger = [v_1^*, \dots, v_n^*]$$

in cui $[v_1^*, \dots, v_n^*]$ è da intendersi come il vettore riga $1 \times n$, i cui elementi v_i^* sono i complessi coniugati di v_i . Considerati i due stati $|v\rangle = (v_1, \dots, v_n)$ e $|w\rangle = (w_1, \dots, w_n)$, si rappresenta il prodotto interno con la notazione *braket*:

$$\langle v|w\rangle := \langle v|w\rangle = |v\rangle^\dagger |w\rangle = \sum_{i=1}^n v_i^* w_i.$$

È degno di nota il fatto che, a differenza della definizione di prodotto interno in spazi di Hilbert presentata nelle trattazioni matematiche, quella appena introdotta è lineare rispetto alla seconda componente, non rispetto alla prima.

Segue da questa definizione che, data una base $|1\rangle, \dots, |n\rangle$ di V , un qualsiasi vettore $|v\rangle \in V$ ha l'unica rappresentazione:

$$v = \sum_{i=1}^n \langle i|v\rangle \cdot |i\rangle$$

in particolare gli elementi $\langle i|v\rangle$ sono le componenti di $|v\rangle$ nella base $|1\rangle, \dots, |n\rangle$.

In funzione del prodotto interno, inoltre, si introducono i concetti di norma $\|v\| := \sqrt{\langle v|v\rangle}$ e di ortogonalità, per cui $|v\rangle$ e $|w\rangle$ sono ortogonali sse $\langle v|w\rangle = 0$. In particolare si dice che la base $|1\rangle, \dots, |n\rangle$ è *ortonormale* sse $\langle i|j\rangle = \delta_{ij} \quad \forall i, j$, ove δ_{ij} è la delta di Kronecker. D'ora in avanti il termine base sarà usato per indicare una base ortonormale.

1.1.2 Operatori e prodotto esterno

I gate quantistici sono descritti da particolari funzioni sullo spazio degli stati: gli *operatori unitari*, i quali manipolano lo stato della computazione come descritto a seguire. Si definisce *operatore* su V una qualsiasi applicazione $A : V \rightarrow V$ lineare, ovvero tale che, per ogni $a_1, \dots, a_n \in \mathbb{C}$ e $|v_1\rangle, \dots, |v_n\rangle \in V$

$$A \left(\sum_{i=1}^n a_i |v_i\rangle \right) = \sum_{i=1}^n a_i \cdot A |v_i\rangle.$$

Dato un operatore A su V esiste un unico operatore A^\dagger su V tale che

$$\langle v | Aw \rangle = \langle A^\dagger v | w \rangle \quad \forall v, w \in V$$

ove $|Aw\rangle := A|w\rangle$ e $\langle A^\dagger v| = (A^\dagger |v\rangle)^\dagger$. L'operatore A^\dagger viene detto *aggiunto* di A e, se la rappresentazione matriciale di A è A_{ij} , quella di A^\dagger è A_{ji}^* .

Un operatore U si dice *unitario* sse $U^\dagger U = Id$. In tal caso vale

$$\langle Uv | Uw \rangle = \langle v | U^\dagger U w \rangle = \langle v | w \rangle \quad \forall v, w \in V \quad (1.1)$$

ovvero l'operatore preserva il prodotto interno e, di conseguenza, la norma. Segue da ciò che ogni operatore unitario ha autovalori di norma 1. Sempre da (1.1) si desume che un operatore unitario manda basi ortonormali in basi ortonormali, un'osservazione che tornerà utile.

Si definisce, inoltre, *proiezione* un operatore P tale che $P = P^\dagger = P^2$.

Dati due vettori $|v\rangle$ e $|w\rangle$ se ne definisce *prodotto esterno* l'operatore $|v\rangle\langle w|$ su V , definito dalla seguente proprietà:

$$(|v\rangle\langle w|) |v'\rangle := \langle w | v' \rangle |v\rangle \quad \forall v' \in V$$

In particolare, posta $|1\rangle, \dots, |n\rangle$ una base di V , si ha:

$$\left(\sum_{i=0}^n |i\rangle\langle i| \right) |v\rangle = \sum_{i=0}^n \langle i | v \rangle |i\rangle = |v\rangle \quad \forall v \in V$$

da cui segue immediatamente l'*equazione di completezza* $\sum |i\rangle\langle i| = Id$.

Preso un arbitrario operatore A su V , applicando due volte l'*equazione di completezza* si ottiene:

$$A = I_V A I_V = \sum_{i,j=0}^n |j\rangle\langle j| A |i\rangle\langle i| = \sum_{i,j=0}^n \langle j | A | i \rangle |j\rangle\langle i| \quad (1.2)$$

in cui è stata introdotta la notazione per l'elemento di matrice $\langle i | A | v \rangle := \langle i | Av \rangle$. Quella in (1.2) si chiama rappresentazione di A tramite prodotto esterno, in cui gli elementi $\langle j | A | i \rangle$ sono le componenti della rappresentazione matriciale di A nella base $|1\rangle, \dots, |n\rangle$, detti anche *elementi di matrice*. In particolare, fissato un sottospazio W di V e una base $|1\rangle, \dots, |k\rangle$ di W , la proiezione su W si può rappresentare nella forma $P = \sum_{i=1}^k |i\rangle\langle i|$.

1.1.3 Prodotto Tensore

Siano $V = \mathbb{C}^n$ e $W = \mathbb{C}^m$. Siano $|v\rangle = (z_1, \dots, z_n) \in V$ e $|w\rangle = (y_1, \dots, y_m) \in W$. È definita naturalmente una mappa $T : V \times W \rightarrow \mathbb{C}^{mn}$ che opera come segue:

$$T(|v\rangle, |w\rangle) = T((z_1, \dots, z_n), (y_1, \dots, y_m)) = (z_1 y_1, \dots, z_1 y_m, \dots, z_n y_1, \dots, z_n y_m)$$

Si denota con $|v\rangle \otimes |w\rangle := T(|v\rangle, |w\rangle)$ il *prodotto tensore* di $|v\rangle$ e $|w\rangle$. Tale operazione gode delle seguenti proprietà:

1. $a(|v\rangle \otimes |w\rangle) = (a|v\rangle) \otimes |w\rangle = |v\rangle \otimes (a|w\rangle) \quad \forall a \in \mathbb{C}$
2. $(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$
3. $|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$
4. $\langle v_1 \otimes w_1 | v_2 \otimes w_2 \rangle = \langle v_1 | v_2 \rangle \langle w_1 | w_2 \rangle$

Si chiama $V \otimes W$ il più piccolo spazio vettoriale contenente tutti gli elementi $|v\rangle \otimes |w\rangle$ al variare di $|v\rangle \in V$ e di $|w\rangle \in W$. Dalla definizione di T e dalle proprietà 2 e 3 risulta chiaro che, se $|v_1\rangle, \dots, |v_n\rangle$ e $|w_1\rangle, \dots, |w_m\rangle$ sono basi rispettivamente per V e W , allora, scelto un ordinamento (per esempio lessicografico) sulle coppie ordinate (i, j) , l'insieme ordinato $|v_i\rangle \otimes |w_j\rangle$ forma un insieme di generatori linearmente indipendenti per $V \otimes W$, la cui dimensione è, dunque, $\dim V \otimes W = \dim V \dim W$. Inoltre la proprietà 4 garantisce che tale insieme ordinato sia ortonormale, ovvero una base.

Dati due operatori A e B rispettivamente su V e W , essi definiscono l'operatore $A \otimes B$ su $V \otimes W$ tramite la seguente formula:

$$A \otimes B \left(\sum_{i,j} a_{ij} |v_i\rangle \otimes |w_j\rangle \right) := \sum_{i,j} a_{ij} A|v_i\rangle \otimes B|w_j\rangle$$

Si può mostrare che quanto appena definito può essere generalizzato a prodotti tensore di ordine maggiore, ovvero si possono definire spazi $V_1 \otimes \dots \otimes V_n$, descritti come sopra. In particolare se $V_1 = \dots = V_n = V$ si introduce la notazione $V^{\otimes n} := V \otimes \dots \otimes V$. Analogamente si definiscono $|v\rangle^{\otimes n} := |v\rangle \otimes \dots \otimes |v\rangle$ e $A^{\otimes n} := A \otimes \dots \otimes A$, rispettivamente per $|v\rangle \in V$ e A operatore su V .

Un commento riguardo alla scelta del nome *prodotto tensore*: l'applicazione T può essere vista come:

$$T(|v\rangle, |w\rangle) = T((z_1, \dots, z_n), (y_1, \dots, y_m)) = \begin{bmatrix} z_1 y_1 & \dots & z_1 y_m \\ \vdots & \ddots & \vdots \\ z_n y_1 & \dots & z_n y_m \end{bmatrix} = |v\rangle |w\rangle^T$$

In cui l'immagine $Q := T(|v\rangle, |w\rangle)$ è una matrice $n \times m$, che rappresenta un 2-tensore, inteso come applicazione bilineare, che opera come $Q(|u\rangle, |t\rangle) := |u\rangle^T Q |t\rangle$, dove $|u\rangle \in V$ e $|t\rangle \in W$. In altri termini, nelle basi precedenti, il tensore Q può essere espresso come

$$Q = \sum_{i=1}^n \sum_{j=1}^m (z_i y_j) (\mathbf{v}_i \otimes \mathbf{w}_j),$$

dove \mathbf{v}_i e \mathbf{w}_j sono i duali degli elementi delle basi. In quest'ottica il prodotto tensore di due spazi V e W , di dimensione finita, fissata una base, è isomorfo allo spazio dei 2-tensori definiti su $V \times W$ e l'isomorfismo è facilmente ricostruibile sulla base dell'osservazione appena proposta. Questo breve excursus mostra la motivazione dietro alla nomenclatura di *prodotto tensore*.

Entanglement

Alcuni elementi di $V \otimes W$ possono essere scritti nella forma $|v\rangle \otimes |w\rangle$, ove $|v\rangle \in V$ e $|w\rangle \in W$ e verranno chiamati vettori *decomponibili*. In genere, però, gli elementi di $V \otimes W$ non possono essere decomposti in un singolo prodotto tensore, ma devono essere espressi come somma finita di *vettori decomponibili*, ovvero $|x\rangle = \sum_{i=1}^n |v_i\rangle \otimes |w_i\rangle$, con $|v_i\rangle \in V$ e $|w_i\rangle \in W$. Questi stati sono detti *entangled* e ricoprono un importante ruolo nella computazione quantistica. Un semplice esempio, che si mostrerà fondamentale nell'analisi della sezione 3.1 sui fenomeni quantistici, di stato *entangled* è:

$$|\beta_{00}\rangle = \frac{|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle}{\sqrt{2}} \in V^{\otimes 2}, \quad \text{con } V := \mathbb{C}^2.$$

La nomenclatura β_{00} è standard e verrà rivista meglio nella sezione 3.1.1 sugli stati di Bell.

Il fatto che lo stato $|\beta_{00}\rangle$ sia *entangled* si dimostra per assurdo: si assume che esistano $|a\rangle \in V$ e $|b\rangle \in W$ tali che $|\beta_{00}\rangle = |a\rangle \otimes |b\rangle$. Si considera, ora, il prodotto interno di $|\beta_{00}\rangle$ con $|0\rangle \otimes |0\rangle$, $|1\rangle \otimes |1\rangle$ e $|0\rangle \otimes |1\rangle$; per le proprietà 1, 2 e 4 del prodotto tensore si ottiene:

$$\begin{aligned} \langle a|0\rangle \langle b|0\rangle &\stackrel{4}{=} \langle \beta_{00}|0 \otimes 0\rangle \stackrel{1,2}{=} \frac{1}{\sqrt{2}} (\langle 1 \otimes 1|0 \otimes 0\rangle + \langle 0 \otimes 0|0 \otimes 0\rangle) \stackrel{4}{=} \frac{1}{\sqrt{2}} \\ \langle a|1\rangle \langle b|1\rangle &= \langle \beta_{00}|1 \otimes 1\rangle = \frac{1}{\sqrt{2}} (\langle 1 \otimes 1|1 \otimes 1\rangle + \langle 0 \otimes 0|1 \otimes 1\rangle) = \frac{1}{\sqrt{2}} \\ \langle a|0\rangle \langle b|1\rangle &= \langle \beta_{00}|0 \otimes 1\rangle = \frac{1}{\sqrt{2}} (\langle 1 \otimes 1|0 \otimes 1\rangle + \langle 0 \otimes 0|0 \otimes 1\rangle) = 0. \end{aligned}$$

Dalla prima equazione si ricava che $\langle a|0\rangle \neq 0$, dalla seconda che $\langle b|1\rangle \neq 0$, mentre dalla terza che almeno una delle due deve essere 0: una contraddizione.

1.1.4 La base computazionale

Prima di passare a nuovi concetti un alleggerimento notazionale: nel prosieguo della trattazione il prodotto tensore $|v_1\rangle \otimes \dots \otimes |v_n\rangle$ sarà indicato con $|v_1\rangle |v_2\rangle \dots |v_n\rangle$ o, ancora più semplicemente, con $|v_1 v_2 \dots v_n\rangle$. Insieme con la proprietà 4 del prodotto tensore, questa notazione comporta che $|v_1 v_2 \dots v_n\rangle^\dagger = \langle v_1 v_2 \dots v_n| = \langle v_1| \otimes \dots \otimes \langle v_n|$.

Spesso, nell'ambito della computazione quantistica, si ha a che fare con lo spazio $V = \mathbb{C}^2$, rappresentante il singolo *qubit*, che, nel caso di sistemi con n qubit, dà origine allo spazio $V^{\otimes n} = \mathbb{C}^{2^n}$. Il nome qubit vuole richiamare il bit, in quanto il concetto di qubit è il più stretto analogo a quello di bit. È, infatti, il più piccolo sistema su cui si opera in computazione quantistica e, scelta una qualsiasi base ortonormale, si possono associare i due elementi di tale base agli stato 0 e 1 del bit.

In particolare, se si usa la notazione di Dirac per la base canonica di \mathbb{C}^2 : $|0\rangle := (1, 0)$ e $|1\rangle := (0, 1)$, si ottiene la seguente base per $V^{\otimes n}$:

$$|0 \dots 00\rangle, |0 \dots 01\rangle, |0 \dots 10\rangle, |0 \dots 11\rangle, \dots, |1 \dots 11\rangle$$

dove ogni elemento è prodotto tensore di n vettori scelti tra $|0\rangle$ e $|1\rangle$. Questa particolare base viene detta *base computazionale*.

Ciascuno dei vettori di tale base è identificato da un numero binario. Si può, quindi, introdurre un'analogia notazione associando a $|i\rangle$ il vettore della *base computazionale* corrispondente alla rappresentazione binaria di i . Si ottiene, dunque, la seguente forma:

$$|i\rangle \quad \text{per } 0 \leq i \leq 2^n - 1$$

Nel resto della trattazione si assumerà sempre l'uso della base computazionale, dove non altrimenti specificato.

1.2 I postulati della meccanica quantistica

La teoria della meccanica quantistica si occupa di descrivere particelle quantistiche e le loro interazioni. Un sistema fisico consistente di una o più particelle quantistiche si dice *sistema quantistico* e viene rappresentato da uno spazio di Hilbert. Per una completa descrizione di tale sistema è necessario, in genere, uno spazio infinito-dimensionale ma, restringendosi a una rappresentazione parziale del sistema, si può ottenere uno spazio degli stati di dimensione finita. In particolare un sistema, per cui si considera una tale descrizione parziale, si dice *finito-dimensionale*. In quanto ai fini della computazione quantistica è sufficiente limitarsi allo studio di questi spazi, nel prosieguo dell'elaborato tutti i sistemi considerati saranno finito-dimensionali.

Un sistema quantistico, inoltre, si dice *isolato* se non interagisce con altri sistemi fisici. Per la trattazione teorica dell'informatica quantistica anche questo fatto giocherà un ruolo importante, quindi, a meno che diversamente specificato, verrà assunto che i sistemi considerati siano isolati.

Postulato 1. *Ad ogni sistema quantistico isolato è associato uno spazio di Hilbert $V = \mathbb{C}^n$, chiamato spazio degli stati del sistema. Inoltre il sistema, in ogni istante, è descritto da uno stato, ovvero da un vettore unitario in V*

In particolare il più semplice sistema quantistico ha come spazio degli stati $V = \mathbb{C}^2$ ed è chiamato *qubit*. Questo sarà l'atomo della computazione quantistica e sarà su sistemi comprendenti uno o più qubit che avverrà la computazione quantistica. Fissata la base computazionale ($|0\rangle, |1\rangle$), il generico stato del qubit assume la forma $|x\rangle = a|0\rangle + b|1\rangle$ in cui a e b sono elementi di \mathbb{C} tali che $|a|^2 + |b|^2 = 1$. Da questo fatto segue una grande differenza di un qubit da un bit: mentre un bit è costretto ad assumere i valori della base (ovvero 0 o 1) il qubit può assumere una loro arbitraria combinazione lineare, a cui spesso ci si riferisce con il termine sovrapposizione, di norma unitaria. Si passa da uno spazio degli stati discreto (con solo due elementi distinti per il bit) ad uno spazio continuo (\mathbb{CP}^{n-1}).

Postulato 2. *L'evoluzione di un sistema quantistico isolato è descritta da un operatore unitario sul suo spazio degli stati; ovvero lo stato $|\psi(t_2)\rangle$, al tempo t_2 , è legato allo stato $|\psi(t_1)\rangle$, al tempo t_1 , da un operatore unitario U_{t_1,t_2} , tale che $|\psi(t_2)\rangle = U_{t_1,t_2} |\psi(t_1)\rangle$*

L'unitarietà dell'operatore garantisce che l'evoluzione temporale preservi i prodotti interni tra i vari stati, inoltre, mantenendo la norma, mappa stati, ovvero vettori unitari, in vettori unitari, quindi stati. Una particolarità di ogni operatore unitario è la reversibilità: è un'operatore biunivoco dallo spazio degli stati in sé stesso. Ciò giocherà un ruolo nel differenziare i computer quantistici da quelli classici.

A priori non tutti gli operatori unitari su uno spazio di Hilbert corrispondono a evoluzioni temporali ammissibili del sistema quantistico associato. Risulta, però, che, nel caso del qubit, tale affermazione è vera, per cui, ignorando la praticità dell'implementazione fisica, ogni operatore unitario rappresenta un valido gate quantistico. Come conseguenza si ha che, anche per un singolo qubit, esistono infiniti gate quantistici teoricamente ammissibili, addirittura un continuum di essi.

In realtà, dal punto di vista dell'implementazione pratica, tale postulato risulta difficile da rispettare. Innanzitutto risulta complesso ottenere un sistema quantistico isolato, ancor più difficile operare su di esso lasciando il sistema isolato. In questa trattazione si ignoreranno tali problematiche. Si rimanda a [NC11], capitolo 3, per approfondire la matematica dei codici a correzione di errori sviluppati per mitigarle.

Postulato 3. *Le misure quantistiche sono descritte da una successione finita di proiettori $\{P_m\}_m$ che agiscono sullo spazio degli stati del sistema e soddisfano l'equazione di completezza $\sum_m P_m = Id$. Se lo stato prima della misura è $|\psi\rangle$ allora la probabilità che l' m -esima misura dia risultato positivo è $p(m) = \langle\psi|P_m|\psi\rangle$ e, in seguito ad essa, il sistema si troverà nello stato*

$$\frac{P_m |\psi\rangle}{\langle\psi|P_m|\psi\rangle^{1/2}} = \frac{P_m |\psi\rangle}{\sqrt{p(m)}}.$$

In particolare lo stato di un sistema non è immediatamente accessibile ad un osservatore esterno. Per ottenerne informazioni, infatti, l'unico modo possibile è di operare con una misura quantistica su di esso, con una conseguente perturbazione del sistema. Inoltre, è importante notare che una misura porta a perdere gran parte dell'informazione contenuta in uno stato arbitrario, soprattutto se in sovrapposizione di svariati elementi della base. Questo è dovuto al fatto che, in seguito al processo di misura, si ottiene solo un risultato, un elemento della base, e si altera lo stato del sistema, allineandolo all'output della misura. Per quanto riguarda la computazione questo fatto si ripercuote sulla difficoltà nell'accedere all'informazione racchiusa da uno stato quantistico. Per arrivare ad essa, infatti, bisogna misurare il sistema, ottenendo solamente un elemento della base come risultato, invece dell'intera combinazione lineare che lo rappresenta. Per questo motivo, in generale, una volta preparato uno stato contenente un grande quantitativo di informazione – ovvero in una sovrapposizione di molti stati della base – è necessario operare su tale stato per ridurre la varietà della combinazione lineare a pochi elementi, di interesse per la computazione. La precedente frase, un po' criptica, verrà esemplificata con l'algoritmo di Deutsch-Josza in sezione 3.1.5.

Una misura quantistica di particolare importanza per la computazione è la *misura nella base computazionale* che, nel caso di $V^{\otimes n} = \mathbb{C}^{2^n}$, assume la forma $\{P_i\}_{i=0}^{2^n-1}$, dove $P_i := |i\rangle\langle i|$.

Prima di introdurre l'ultimo postulato si definisce *sistema composito* un sistema quantistico ottenuto considerando insieme un numero finito di più piccoli sistemi quantistici, detti *componenti*.

Postulato 4. Lo spazio degli stati di un sistema composito è dato dal prodotto tensore degli spazi degli stati dei propri componenti. Se i componenti sono esattamente n , rispettivamente negli stati $|\psi_1\rangle, \dots, |\psi_n\rangle$, lo stato del sistema composito è $|\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle$.

Se si considera il sistema composito di n componenti, ciascuna delle quali è un *qubit*, il sistema si dice *n-qubit*. In genere lo si dota della base computazionale $|x\rangle$, con $x \in \{0, 2^n - 1\}$ con associata la misura nella base computazionale $\{P_x\}_{x=0}^{2^n-1}$.

1.2.1 La sfera di Bloch

Si considerino, per un n -qubit, gli stati $e^{i\theta}|\psi\rangle$ e $|\psi\rangle$, con $\theta \in \mathbb{R}$. Si dice che i due stati differiscono per un fattore di *fase globale* $e^{i\theta}$. Dal punto di vista delle misurazioni due stati che differiscono per una *fase globale* risultano indistinguibili e, per questo motivo, si tende a ignorare tale fattore di fase. Per chiarificare l'affermazione precedente, fissato un arbitrario $i \in \{0, \dots, 2^n - 1\}$ si considera la proiezione associata P_i , elemento della misura nella base computazionale. La probabilità che $|\psi\rangle$ venga misurato nello stato $|i\rangle$ è data da $p(i) = \langle\psi|P_i|\psi\rangle$, mentre per $e^{i\theta}|\psi\rangle$ è $p_\theta(i) = \langle e^{i\theta}\psi|P_i|e^{i\theta}\psi\rangle = \langle\psi|P_i|e^{-i\theta}e^{i\theta}\psi\rangle = \langle\psi|P_i|\psi\rangle = p(i)$. Per l'arbitrarietà di i , si nota che hanno le stesse probabilità di essere misurati in un qualsiasi stato della base computazionale. Il procedimento appena evidenziato è indipendente dalla scelta di base, ma risulta valido anche con altre definizioni di misurazione e per spazi infinito-dimensionali. Per questa universalità, in genere, si considerano due stati che differiscono per una fase globale come fisicamente equivalenti.

Un importante concetto che segue da quanto appena mostrato è che, ai fini computazionali, un qubit non necessita di 4 numeri reali per essere rappresentato, in quanto se ne può scartare uno, che rappresenta la fase globale del qubit. In particolare, per un arbitrario stato $|\psi\rangle = a|0\rangle + b|1\rangle$, si può utilizzare la seguente rappresentazione:

$$|\psi\rangle = e^{i\gamma} \left(\cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle \right) \quad (1.3)$$

nella quale θ, γ, φ sono numeri reali. Questa rappresentazione è sensata in quanto $|a|^2 + |b|^2 = 1$. Per quanto affermato in precedenza si può effettivamente ignorare il fattore $e^{i\gamma}$ di fase globale, per ottenere l'equivalente stato

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\varphi} \sin \frac{\theta}{2} |1\rangle, \quad (1.4)$$

in cui i numeri φ e θ individuano il punto

$$(\cos \varphi \sin \theta, \sin \varphi \sin \theta, \cos \theta) \quad (1.5)$$

sulla sfera unitaria \mathbb{S}^2 . Per l'importanza che ricopre nel creare un'intuizione riguardo allo stato dei qubit questa rappresentazione ha un nome: quello in (1.5) è detto *vettore di Bloch* e rappresenta un punto sulla *sfera di Bloch*, come mostrato in figura 1.1.

Per quanto questa rappresentazione sia intuitiva e comoda per studiare l'effetto dei gate quantistici su singoli qubit, non è facilmente generalizzabile al caso di n qubit.

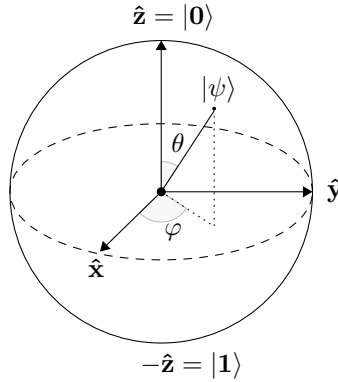


Figura 1.1: Stato di un qubit rappresentato sulla sfera di Bloch

Questo in quanto in quel caso la quantità di numeri reali necessari per descrivere lo stato della computazione è troppo grande per poter essere facilmente rappresentata in 3 dimensioni, rendendo pressoché inutile una rappresentazione alternativa dello stato di un n -qubit, che non gioverebbe all'intuizione.

1.3 Gate quantistici

Per poter costruire i primi circuiti ed algoritmi quantistici è fondamentale introdurre delle prime matrici, e quindi degli operatori, su \mathbb{C}^2 con *base computazionale*. La scelta dei seguenti ricade sulla loro grande presenza nel lavoro che segue e in letteratura.

Gate di Pauli e di Hadamard

Le prime che saranno citate sono le *matrici di Pauli*, di grande utilità, anche nel più vasto campo della meccanica quantistica:

$$\sigma_0 := Id = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \sigma_1 = \sigma_x = X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad (1.6)$$

$$\sigma_2 = \sigma_y = Y := \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_3 = \sigma_z = Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (1.7)$$

Segue la *matrice di Hadamard*, che ricorrerà spesso nella stesura degli algoritmi quantistici:

$$H := \frac{1}{\sqrt{2}} (X + Z) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Tutti questi, chiaramente, sono operatori unitari su $V = \mathbb{C}^2$ e, quindi, gate quantistici per singoli qubit. In particolare, la matrice X scambia gli elementi della base computazionale, mandando $|0\rangle$ in $|1\rangle$ e viceversa. Per questo motivo, in accordo con la computazione classica, si chiama gate NOT quello che agisce come definito da X sulla base computazionale.

Chiaramente si possono introdurre anche gate che agiscono su più di un qubit. Un primo, ed importantissimo, esempio è basato sull'operatore di *Hadamard* H , su V , il

quale può essere generalizzato a $H^{\otimes n}$ su $V^{\otimes n}$, un gate su n -qubit che ritornerà frequentemente nel resto della trattazione. Per ottenerne una scrittura esplicita bisogna, innanzitutto, esprimere l'operatore di *Hadamard* come prodotto esterno:

$$H = \frac{1}{\sqrt{2}} (|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| - |1\rangle\langle 1|) = \frac{1}{\sqrt{2}} \sum_{x,y=0}^1 (-1)^{x \cdot y} |x\rangle\langle y|$$

dove x e y sono in $\{0, 1\}$ e $x \cdot y$ indica l'usuale prodotto. Ciò diventa:

$$H^{\otimes 2} = \frac{1}{\sqrt{2^2}} \sum_{x,y,x',y'} (-1)^{x \cdot y + x' \cdot y'} |xx'\rangle\langle yy'| = \frac{1}{\sqrt{2^2}} \sum_{x'',y''} (-1)^{x'' \cdot y''} |x''\rangle\langle y''|$$

in cui, nell'ultimo membro, la somma è sui numeri binari di due cifre, ovvero $x'', y'' \in \{00, 01, 10, 11\}$, e $x'' \cdot y''$ indica il prodotto bit per bit modulo 2. Per chiarificare, un paio di esempi:

$$\begin{aligned} 10 \cdot 11 &= 1 \cdot 1 + 0 \cdot 1 = 1 + 0 = 1 \pmod{2} \\ 11 \cdot 11 &= 1 \cdot 1 + 1 \cdot 1 = 1 + 1 = 0 \pmod{2}. \end{aligned}$$

Analogamente questa scrittura è equivalente alla seguente:

$$H^{\otimes 2} = \frac{1}{\sqrt{2^2}} \sum_{x,y=0}^3 (-1)^{x \cdot y} |x\rangle\langle y|$$

dove x e y variano tra 0 e 3, mentre $x \cdot y$ indica il prodotto bit per bit modulo 2 della rappresentazione binaria di x e di y . Infine questa forma è immediatamente generalizzata al caso di

$$H^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x,y=0}^{2^n-1} (-1)^{x \cdot y} |x\rangle\langle y| \quad (1.8)$$

in cui $x \cdot y$, come prima, indica il prodotto bit per bit modulo 2 della rappresentazione binaria di x e di y .

Risulta importante notare come agisce il gate $H^{\otimes n}$ sull'elemento $|0\rangle$:

$$|0\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle. \quad (1.9)$$

Esso, infatti, associa al vettore $|0\rangle$ la combinazione lineare uniforme di tutti gli elementi della base computazionale. Tale caratteristica sarà sfruttata spesso nella costruzione di algoritmi quantistici.

Gate controllati

Un altro gate che ricopre un ruolo fondamentale nella teoria della computazione quantistica è il gate NOT controllato, o CNOT. Tale gate opera su 2 qubit nel seguente modo: lascia invariato il primo qubit, usandolo come controllo per operare sul secondo. In particolare, se il primo qubit è nello stato $|0\rangle$, il secondo rimane invariato, altrimenti

ti opera sul secondo qubit negandolo, ovvero applicando il gate corrispondente alla matrice di Pauli X . Più in generale, dato un qualsiasi operatore unitario U si può implementare il gate U controllato, CU , che opera in modo simile al gate CNOT: lascia invariato il primo qubit e opera sul secondo, applicando il gate U , solo se il primo è nello stato $|1\rangle$. Per dimostrare che i vari gate U controllati sono gate quantistici ammissibili bisogna dimostrare che sono unitari. A tal fine si introduce la rappresentazione matriciale

$$CU := \left[\begin{array}{c|c} Id & 0 \\ \hline 0 & U \end{array} \right] \quad (1.10)$$

che rappresenta il gate U controllato nella base computazionale. Per convincersene si ricorda che la prima metà degli elementi della base ha il primo qubit nello stato 0. Chiaramente l'operatore aggiunto CU^\dagger avrà rappresentazione

$$CU^\dagger := \left[\begin{array}{c|c} Id & 0 \\ \hline 0 & U^\dagger \end{array} \right] \quad (1.11)$$

da cui segue immediatamente, per prodotto a blocchi di matrici, che

$$CU \cdot CU^\dagger := \left[\begin{array}{c|c} Id & 0 \\ \hline 0 & UU^\dagger \end{array} \right] = Id \quad (1.12)$$

ovvero che l'operatore U controllato è unitario, per ogni operatore unitario U , per cui è un gate quantistico ammissibile.

Gate Toffoli

L'ultimo operatore introdotto in questa sezione, la cui importanza teorica è notevole e verrà mostrata più approfonditamente in seguito, è il gate *Toffoli*. Tale operatore può essere descritto come un gate NOT doppiamente controllato: opera su 3 qubit lasciando inalterati i primi 2, mentre agisce con un gate NOT solo se i primi due qubit sono nello stato $|11\rangle$. È rappresentato, nella base computazionale, dalla matrice

$$T := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (1.13)$$

la quale è chiaramente unitaria, per cui anche il gate Toffoli è un gate quantistico ammissibile.

1.4 Circuiti quantistici

Un circuito classico consiste in una serie di cavi che congiungono, senza entrare in cicli, delle porte logiche. L'informazione è trasportata dai cavi sotto forma di corrente elettrica ed è trasformata dai gate logici a intervalli regolari marcati dal tempo di clock del processore. In particolare l'operazione di un circuito classico può essere vista come una serie di manipolazioni dei dati di input, che risulta nell'output misurato.

Un computer quantistico opera in maniera analoga: al posto dei cavi si ha l'evoluzione temporale o spaziale di un sistema quantistico e i gate logici sono sostituiti da *gate quantistici*, che corrispondono, per il postulato 2, a operatori unitari sullo spazio degli stati. In particolare un grafo aciclico diretto, i cui nodi rappresentano gate quantistici che operano sullo stato dei vari qubit in input, forma un circuito quantistico. Questa definizione informale verrà rivista e approfondita nel capitolo 2, alla sezione 2.5.2.

Per chiarificare il *modus operandi* del modello computazionale del circuito quantistico, in cui verranno implementati gli algoritmi del capitolo 3, si descrivono nel dettaglio le fasi e le componenti coinvolte nella computazione quantistica.

Innanzitutto nel modello di circuito quantistico si considera, in genere, la presenza di un computer classico a supporto delle operazioni di input e output. Tale supporto, inoltre, può essere usato per computare le routine classiche presenti negli algoritmi quantistici, permettendo di utilizzare il modello propriamente quantistico per le operazioni che più traggono beneficio da esso. Il modello del circuito quantistico prevede la possibilità di preparare gli n -qubit della computazione in uno stato della base computazionale, corrispondente al valore dei dati di input su cui si intende operare. Prevede, inoltre, la possibilità di applicare su tali qubit ogni gate in una famiglia di porte logiche quantistiche. Per lo studio teorico, in genere, si considera una famiglia infinita di gate, costituita da tutti i gate su un qubit ammissibili, accompagnati dai gate su un qubit controllati. L'esecuzione dei gate del circuito avviene seguendo la definizione del circuito come grafo diretto, risultando in uno stato particolare del sistema di n -qubit. A questo punto si opera con una misura nella base computazionale, che restituisce uno stato, il cui valore corrispondente è l'output della computazione. Quest'ultimo passo costituisce una grande differenza dalla computazione classica ed è dovuto all'impossibilità di osservare lo stato di un sistema quantistico senza perturbarlo.

Per rappresentare graficamente un circuito quantistico si è sviluppata una notazione standard, che permette di presentare in modo conciso la struttura di un algoritmo, senza doverlo necessariamente descrivere nel dettaglio a parole. Ogni qubit è rappresentato da una linea orizzontale, che ne indica l'evoluzione temporale (letta da sinistra verso destra), e si assume che lo stato iniziale sia espresso in base computazionale. Un gate agente su uno o più qubit è rappresentato da un riquadro, al cui interno è posto un identificativo del gate. Esso opera sui qubit corrispondenti ai cavi entranti nel riquadro, portandoli a un nuovo stato, che viene trasportato dai cavi uscenti. Un esempio elementare di circuito con un paio di gate su qubit singoli è:

$$\begin{array}{c} |0\rangle \text{ --- } \boxed{H} \text{ --- } \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ |1\rangle \text{ --- } \boxed{X} \text{ --- } |0\rangle \end{array}$$


in cui il gate di Hadamard che agisce sul primo qubit e un gate NOT agisce sul secondo.

I gate controllati si indicano con un pallino sulla linea del bit di controllo, collegato verticalmente al riquadro del gate. Se il pallino è nero il gate viene eseguito quando il qubit di controllo è nello stato $|1\rangle$, mentre se è bianco quando è nello stato $|0\rangle$. In particolare il gate CNOT si indica con un + cerchiato, al posto del riquadro corrispondente al gate NOT, come rappresentato in figura (1.2). Questa notazione è motivata dal fatto che l'operatore CNOT, sugli elementi della base computazionale $|x\rangle, |y\rangle \in \{|0\rangle, |1\rangle\}$,

agisce nel seguente modo

$$|x, y\rangle \mapsto |x, x \oplus y\rangle \quad (1.14)$$

dove il simbolo \oplus indica la somma modulo 2.

Oltre ai gate controllati da un singolo qubit si possono introdurre gate controllati da più qubit, tra cui il gate *Toffoli* (il secondo gate nel circuito a sinistra in figura 1.2), i quali vengono eseguiti sse tutti i bit di controllo rispettano la condizione indicata dal pallino. Si introduce anche il simbolo  per indicare una misurazione, che in genere viene effettuata a fine computazione.

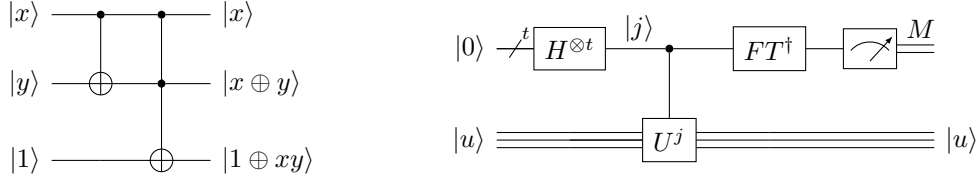


Figura 1.2: Esempi di circuiti quantistici.

Dal simbolo di misurazione si nota che escono due cavi, al posto che uno solo, i quali indicano che viene trasportata informazione classica, non quantistica. Inoltre, a volte, viene indicato con un'etichetta, in questo caso M , l'output della misurazione, se deve essere utilizzato nel resto della computazione. Un'ultima notazione spesso utilizzata nei circuiti è quella evidenziata nel circuito a destra della figura 1.2, in cui il simbolo $/$ sul cavo indica che il filo rappresenta un insieme di svariati qubit, in cui a volte se ne indica anche il numero con la notazione $/^n$. Un'altra rappresentazione, alternativa alla precedente, per rappresentare l'evoluzione di un sistema di più qubit è quella mostrata nello stesso circuito, per lo stato $|u\rangle$, con 3 linee parallele.

Un esempio utile di circuito quantistico è quello di scambio di qubit, la cui notazione è quella mostrata a sinistra della figura 1.3, mentre viene implementato come descritto a destra nella stessa figura. L'effetto che ottiene sullo stato della computazione è perfettamente descritto dal nome e funziona in quanto agisce nel seguente

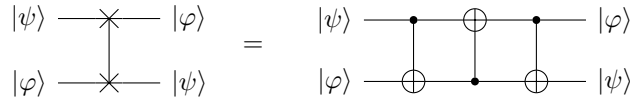


Figura 1.3: Circuito che implementa lo scambio di qubit

modo sugli elementi della base computazionale $|x\rangle, |y\rangle \in \{|0\rangle, |1\rangle\}$:

$$\begin{aligned} |x, y\rangle &\mapsto |x, x \oplus y\rangle \\ &\mapsto |x \oplus (x \oplus y), x \oplus y\rangle = |y, x \oplus y\rangle \\ &\mapsto |y, (x \oplus y) \oplus y\rangle = |y, x\rangle \end{aligned}$$

Estendendolo per linearità ai generici qubit, si osserva che opera come richiesto.

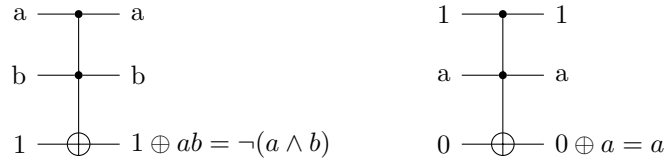


Figura 1.4: Implementazione di gate NAND e FANOUT usando un gate Toffoli e dei bit ausiliari. Nel primo circuito viene implementato un gate NAND, i cui due input sono i due bit di controllo e l'output è il terzo bit. Nel secondo circuito si implementa un gate FANOUT, in cui il primo e l'ultimo bit sono ausiliari, il secondo bit costituisce l'input e l'output si legge nel secondo e terzo bit.

1.5 Paragone con la computazione classica

Il modello della computazione quantistica, oltre a basarsi su fenomeni fisici fondamentalmente differenti da quelli classici, dà origine a un paradigma di programmazione fortemente distinto da quello della computazione classica. In questa sezione saranno mostrate svariate differenze tra i due modelli, anche tramite vantaggi e svantaggi del primo rispetto al secondo. Innanzitutto nella computazione quantistica, per un sistema di 1 qubit, esistono infiniti gate ammissibili differenti (come precedentemente mostrato nella sezione 1.2), contro i solo due gate classici distinti per un singolo bit: l'identità e la negazione. Per quanto questa abbondanza possa far pensare a una maggiore libertà nella scelta delle operazioni eseguibili in computazione quantistica, non è necessariamente così: spesso i gate classici implementati, come per esempio i gate AND e OR, sono irreversibili, ovvero non sono funzioni binarie biunivoche. Ogni gate quantistico, invece, dovendo essere un operatore unitario, è forzato ad essere reversibile, ovvero biunivoco. In genere si considera l'insieme dei gate NAND¹ e FANOUT², o analogamente AND, NOT e FANOUT, come insieme di gate classici universali. Si chiama insieme di gate universali un insieme di porte logiche con le quali è possibile computare ogni funzione $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, e quindi eseguire ogni computazione classica ammissibile. Per una dimostrazione formale dell'affermazione che l'insieme dei soli gate NAND e FANOUT è universale si rimanda a [NC11], sezione 3.1.2. Purtroppo il gate NAND è irreversibile e non può essere simulato su un computer quantistico. Fortunatamente, come è stato studiato da Fredkin – si veda [FT82] – e Toffoli – [Tof80] – esistono delle porte logiche classiche universali e reversibili, implementabili come gate quantistici, che rendono evidente il fatto che la teoria della computazione quantistica sia almeno tanto ricca quanto quella classica, riuscendo a emularla. Un esempio di tale porta logica è il gate *Toffoli*, definito nella sezione 1.4. Tale gate, infatti con l'ausilio di input ausiliari e output superflui, può emulare i gate NAND e FANOUT, come mostrato in figura 1.4, da cui segue la sua universalità.

Risulta, quindi, teoricamente molto interessante un'implementazione di tale gate che sia praticamente realizzabile, in particolare che tramite porte logiche su 1 singolo qubit che siano fisicamente costruibili. I gate CNOT e *Hadamard*, già incontrati, lo sono. Inoltre, un altro operatore ricorrente in letteratura, la cui realizzazione fisica è stata altrettanto studiata, è il gate $\pi/8$, denotato con la lettera T , che agisce nel

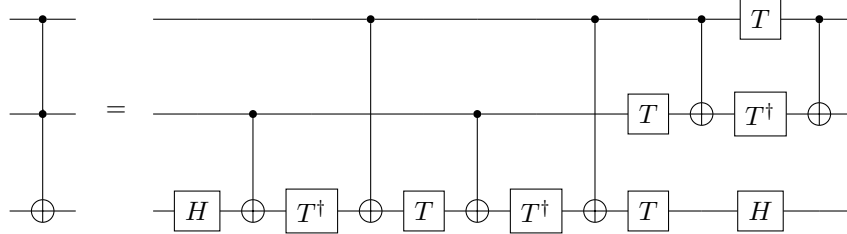
¹NAND opera come un gate AND seguito da un gate NOT

²FANOUT a partire da un singolo bit ne restituisce due copie esatte

seguinte modo

$$T := \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} = e^{i\pi/8} \begin{bmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{bmatrix}. \quad (1.15)$$

Il nome, storico, di tale gate è dovuto al fatto che la sua definizione, a meno di una fase globale, corrisponde con l'operatore sulla cui diagonale compaiono solo $e^{\pm i\pi/8}$. Con queste sole tre porte logiche si può costruire il seguente circuito



ovvero un'implementazione fisica del gate Toffoli.

Questo fatto, di interesse più teorico che pratico, non deve distogliere l'attenzione dalla computazione più propriamente quantistica, la quale mostra delle ulteriori peculiarità che verranno esplorate in modo più approfondito nelle seguenti sezioni. Per evidenziare la differenza del paradigma di programmazione quantistico da quello classico si mostreranno due "limitazioni" della computazione quantistica, lasciando i comportamenti più positivi per il successivo capitolo 3.

Una prima differenza è causata dal postulato 3 della meccanica quantistica: uno stato quantistico non può essere direttamente osservato, ma si possono ottenere informazioni su di esso solamente tramite misure. Oltre al fatto che questo procedimento è fondamentalmente probabilistico e, quindi, introduce incertezza nella computazione, ciò comporta che, in generale, non è possibile ottenere tutte le informazioni immagazzinate nello stato finale di una specifica computazione (di cui un esempio importantissimo sarà lo stato finale della trasformata di Fourier quantistica, descritta nella sezione 3.2). Ciò richiede, per sfruttare a pieno le capacità computazionali dei sistemi quantistici, maggior ingegno nella progettazione degli algoritmi, i quali devono essere in grado di sfruttare le informazioni nascoste nelle sovrapposizioni di stati, per arrivare a output in cui solo i risultati interessanti abbiano grande probabilità di essere misurati.

Un'ulteriore differenza, dovuta al seguente teorema, evidenzia la grande mole di informazioni immagazzinata in un singolo qubit. In particolare questo risultato esclude dagli strumenti utilizzabili nella costruzione di algoritmi quantistici la possibilità, molto comunemente usata in ambito classico, di copiare esattamente un arbitrario qubit.

Teorema 1.5.1 (No-cloning theorem). *Non esiste un gate quantistico che, a partire da un qubit arbitrario, possa crearne una copia identica, senza distruggere il qubit di partenza*

Dimostrazione. Suppongo che esista un operatore unitario U su \mathbb{C}^4 che riesca a clonare un arbitrario stato $|\psi\rangle \in \mathbb{C}^2$, ovvero tale che:

$$\exists |s\rangle \in \mathbb{C}^2 : (|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle. \quad (1.16)$$

Considero un altro stato $|\varphi\rangle \in \mathbb{C}^2$ arbitrario che U riesca a clonare, ovvero:

$$U(|\varphi\rangle \otimes |s\rangle) = |\varphi\rangle \otimes |\varphi\rangle. \quad (1.17)$$

Prendendo i prodotti interni delle due equazioni precedenti si ottiene:

$$\langle\psi|\varphi\rangle = \langle\psi|\varphi\rangle \langle s|s\rangle = \langle\psi \otimes s|\varphi \otimes s\rangle = \quad (1.18)$$

$$\langle U(\psi \otimes s)|U(\varphi \otimes s)\rangle = \langle\psi \otimes \psi|\varphi \otimes \varphi\rangle = (\langle\psi|\varphi\rangle)^2 \quad (1.19)$$

Ma l'equazione $x = x^2$ ammette solo 0 e 1 come soluzioni, da cui segue che $|\psi\rangle = |\varphi\rangle$ o $|\psi\rangle \perp |\varphi\rangle$. Ciò implica che se un gate quantistico riesce a duplicare un particolare stato $|\psi\rangle$ di un qubit, potrà duplicare solamente stati perpendicolari ad esso, ovvero non può esistere un gate quantistico che duplichi un qubit, indipendentemente dallo stato (ignoto) in cui si trova. ■

In seguito a questa breve introduzione si dovrebbe aver chiaro che il mondo della computazione quantistica rappresenta un ambito fondamentalmente distinto da quella classica. Inoltre sono stati introdotti un numero sufficiente di strumenti per poter iniziare a studiare come queste differenze influiscano sulla potenza computazionale di tali macchine.

Capitolo 2

Complessità computazionale

L'obiettivo della complessità computazionale è quello di catalogare i vari problemi in base alla loro complessità intrinseca e di quantificare le risorse computazionali richieste per la loro risoluzione. Per risorse computazionali si intendono il tempo necessario per terminare una computazione, la quantità di gate o di memoria richieste per eseguire un algoritmo o, meno frequentemente, l'energia necessaria per risolvere un problema. Questa volontà è motivata dal desiderio degli uomini di individuare le classi dei problemi facilmente risolvibili, ovvero per cui esiste un algoritmo efficiente, e dei problemi facilmente verificabili, ovvero per cui esiste una configurazione che certifica la risolubilità del problema. Per analizzare le richieste è necessario fissare un sistema computazionale su cui eseguire gli algoritmi. In base ad esse, poi, si definiscono le classi di complessità in cui sono divisi i problemi. Lo scopo delle prossime sezioni sarà di introdurre i due principali modelli computazionali, ai fini della computazione quantistica: la macchina di Turing e i circuiti quantistici. Su di essi si definiranno, poi, le più importanti classi di complessità, tra cui le celeberrime **P** – classe dei problemi risolvibili in modo efficiente – ed **NP** – classe dei problemi facilmente verificabili – e le controparti quantistiche **BQP** e **QMA**.

2.1 Macchine di Turing

Le macchine di Turing sono un modello computazionale classico di enorme rilevanza storica e teorica. Esse sono strumenti semplici, ma estremamente potenti e versatili. Per giustificare l'inclusione in questa trattazione si è scelto di anteporre alla loro definizione una famosa tesi che vuole rendere più chiara la potenza computazionale di tale modello:

Tesi di Church-Turing: La classe di funzioni computabili da una macchina di Turing corrisponde alla classe di funzioni che possono essere calcolate seguendo un algoritmo.

Questa tesi non è un teorema e non può essere dimostrata, in quanto sarebbe necessario definire in modo preciso la classe di funzioni calcolabili tramite un algoritmo. Quello che, invece, vuole affermare è l'uguaglianza tra la precisa nozione della classe di funzioni calcolabili da una macchina di Turing e l'insieme delle funzioni che, intuitivamente, si considerano computabili seguendo un algoritmo. Giustifica, quindi, la scelta della macchina di Turing come modello computazionale di riferimento, capace di computare tutto ciò che può essere calcolato.

Macchina di Turing

Una *macchina di Turing a k nastri* è una macchina astratta dotata delle seguenti caratteristiche:

Spazio di lavoro: i k nastri formano lo spazio di lavoro. Un nastro è una sequenza monodimensionale, infinita a destra, di celle. Ogni cella può contenere un elemento di un alfabeto Γ . Su ogni nastro è presente una testina che può leggere o scrivere sulla cella su cui è posizionata. La computazione avviene in intervalli temporali discreti in cui, in ciascun intervallo, le testine possono muoversi al più di una posizione, a destra o sinistra, sul proprio nastro.

In particolare il primo nastro è il nastro di *input*, contenente i dati del problema. È un nastro di sola lettura e la testina non può scrivere su esso. Gli altri, invece, sono i nastri di lavoro, sui quali le testine possono anche scrivere. Uno di questi viene dedicato all'*output*, ed è il nastro da cui si legge il risultato della computazione quando la macchina di Turing si arresta.

Insieme di regole operative: una macchina di Turing può trovarsi in un qualunque stato di un fissato insieme, detto Q . Tra questi saranno presenti q_{start} , assunto dalla macchina a inizio computazione, e q_{halt} , che segnala il termine delle operazioni. Lo stato in cui si trova determina il comportamento della macchina di Turing. Infatti, essa opera nel seguente modo: (1) le k testine leggono il contenuto della cella su cui si trovano, (2) in base allo stato della macchina le $k - 1$ testine scrivono sulla cella un nuovo simbolo dell'alfabeto, (3) sempre in base allo stato, le varie testine si muovono a destra o sinistra, al più di una posizione, (4) la macchina di Turing cambia stato, scegliendo dall'insieme Q .

Formalmente una macchina di Turing è definita da

Definizione 2.1: Macchina di Turing.

Una macchina di Turing M a k nastri è una tripletta ordinata (Γ, Q, δ) , dove:

- Γ è l'insieme, detto alfabeto di M , con i simboli che possono essere salvati nelle celle dei nastri. Contiene i simboli \square , che indica la cella vuota, e \triangleright , che indica l'inizio del nastro.
- Q è l'insieme degli stati in cui può trovarsi M . Vi appartengono gli stati q_{start} e q_{halt} , che rappresentano l'inizio e la fine della computazione.
- $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ è detta funzione di transizione e regola il comportamento della macchina di Turing.

Il comportamento della macchina di Turing M è determinato da δ nel seguente modo: siano $q \in Q$ lo stato di M e $(\sigma_1, \sigma_2, \dots, \sigma_k)$ i simboli letti sui k nastri. Sia $(q', (\sigma'_2, \dots, \sigma'_k), z) := \delta(q, (\sigma_1, \sigma_2, \dots, \sigma_k))$, dove $z \in \{L, S, R\}^k$ l'immagine di δ . Al prossimo passo della computazione M passerà allo stato q' , le $k - 1$ testine sui nastri di lavoro scriveranno $(\sigma'_2, \dots, \sigma'_k)$ sulla propria cella e, infine, le k testine si muoveranno in base al corrispondente valore di z : se è L a sinistra, R a destra mentre con S non si muovono. Si assume che una testina, se è sulla prima cella e la mappa δ le impone di muoversi a sinistra, non si muoverà.

La computazione di M , fissata una stringa di input x , avviene secondo il seguente schema. Tutti i nastri, a meno di quello di input, sono inizializzati con il simbolo \triangleright sulla prima cella e ogni altra cella vuota, ovvero con il simbolo \square . Il nastro di input è

inizializzato con la prima cella a \triangleright , seguita da una sequenza finita di celle non vuote, che rappresentano l'input x , e il resto con il simbolo \square . A inizio computazione tutte le testine sono sulla cella più a sinistra – con il simbolo \triangleright – e M è nello stato q_{start} . Questa è chiamata configurazione iniziale di M sull'input x .

Da questa configurazione iniziale la computazione segue il procedimento codificato da δ , come descritto in precedenza, fino a che non raggiunge lo stato q_{halt} . In questo stato la funzione di transizione è definita per non modificare ulteriormente la configurazione della macchina M . Si è, quindi, fermata la computazione e si può leggere l'output nel nastro designato.

Non è necessario che la macchina di Turing arrivi mai allo stato q_{halt} . Potrebbe, infatti, non terminare mai la computazione e non restituire mai un output completo.

Si introduce, quindi, la seguente notazione: Fissata una macchina di Turing M e un input x , si definisce:

$$M(x) := \begin{cases} y & \text{se } M \text{ raggiunge lo stato } q_{\text{halt}} \\ \nearrow & \text{altrimenti} \end{cases}, \quad (2.1)$$

dove, nel primo caso, y è la stringa letta sul nastro di output quando la macchina di Turing raggiunge lo stato q_{halt} .

2.1.1 Codifica

Definizione 2.2: stringa binaria.

Si definisce stringa binaria di lunghezza n un elemento di $\{0, 1\}^n$.

L'insieme di tutte le stringhe binarie è $\{0, 1\}^ := \cup_{n \geq 0} \{0, 1\}^n$.*

In genere, nello studio della complessità computazionale, si considerano solamente funzioni il cui dominio e codominio sono stringhe binarie. Si possono rappresentare funzioni su oggetti arbitrari tramite questo tipo di mappe. Per raggiungere questo scopo bisogna definire una *codifica*, ovvero una mappa iniettiva dall'insieme contenente tutti gli oggetti su cui si vuole operare, a valori nell'insieme delle stringhe binarie $\{0, 1\}^*$. L'unica richiesta che si pone è che a oggetti distinti corrispondano stringhe distinte.

Vista questa consuetudine si tendono a considerare macchine di Turing il cui alfabeto è $\Gamma := \{\square, \triangleright, 0, 1\}$. Fissato un problema, poi, si specifica una codifica in cui rappresentare la classe di oggetti su cui si opera. Nel prosieguo della trattazione, infatti, si assumerà sempre l'utilizzo di tale alfabeto e, se necessario, sarà anche introdotta la codifica usata.

2.2 Problemi decisionali

Una particolare classe di problemi studiati nella teoria della complessità computazionale sono i cosiddetti *problemi decisionali*, associati ai *linguaggi*. Essi permettono di costruire un'elegante famiglia di classi di complessità e, per questo motivo, sono spesso usati in letteratura. Per arrivare alla definizione di linguaggio è necessario introdurre il concetto di stringa su alfabeti arbitrari, non solo binaria:

Definizione 2.3: Insieme delle stringhe.

Dato un insieme finito Σ , detto alfabeto, si dice stringa di lunghezza n , sull'alfabeto Σ , un elemento di Σ^n .

L'insieme di tutte le stringhe sull'alfabeto Σ è $\Sigma^ := \cup_{n \geq 0} \Sigma^n$.*

Con questa nozione si introduce il concetto di linguaggio:

Definizione 2.4: Linguaggio.

Dato un'alfabeto Σ si definisce linguaggio un sottoinsieme $L \subset \Sigma^*$ delle stringhe sull'alfabeto Σ .

Il problema decisionale associato chiede di decidere se, dato un arbitrario $x \in \Sigma^*$, si ha $x \in L$ oppure no. Si dice che x è una soluzione al problema decisionale, o verifica il problema, sse $x \in L$.

È importante notare che ogni linguaggio può essere associato a una funzione Booleana – ovvero il cui codominio è $\{0,1\}$ – e viceversa. La prima associazione si ottiene definendo $f : X \rightarrow \{0,1\}$ in modo che $f(x) = 1 \iff x \in L$, ovvero $L = \{x \in X : f(x) = 1\}$. Data una funzione Booleana sull'insieme X , analogamente, si definisce il linguaggio associato $L_f := \{x \in X : f(x) = 1\}$.

Si possono utilizzare le macchine di Turing per *decidere* i linguaggi, ovvero per verificare se, dato un linguaggio L , un determinato elemento x appartiene a L . Per far ciò si introduce un'opportuna codifica c , che mappi L in $\{0,1\}^*$, in modo da ottenere un analogo linguaggio $c(L) \subseteq \{0,1\}^*$. Si dice, ora, che una macchina di Turing M *decide* o *risolve* il linguaggio L sse

$$M(x) = \begin{cases} 1 & \text{se } x \in c(L) \\ 0 & \text{se } x \notin c(L) \end{cases}, \quad (2.2)$$

ovvero se, per ogni stringa x in input, M arriva allo stato q_{halt} in un tempo finito. In particolare la macchina M , in un tempo finito, afferma che il valore in input è, o non è, una soluzione al problema decisionale L .

Una condizione più debole, ma comunque utile, chiede che

$$M(x) = 1 \quad \forall x \in c(L). \quad (2.3)$$

In tal caso si dice che la macchina M *accetta* L . La condizione è più debole in quanto, se $x \notin c(L)$, non si richiede nemmeno che M termini la computazione, potrebbe essere $M(x) = \nearrow$. Questa condizione è più naturale per codifiche di linguaggi, in quanto, dato un linguaggio $L \subseteq X$, in genere esistono stringhe $x \notin c(X)$.

2.3 Richieste computazionali

Per analizzare la complessità di un problema è necessario, innanzitutto, definire una grandezza che rappresenti la richiesta di risorse computazionali di un dato algoritmo o di una data macchina di Turing.

La prima risorsa considerata è il tempo. Nel caso di macchine di Turing può essere facilmente definito il *tempo di computazione* contando le operazioni elementari necessarie a giungere allo stato q_{halt} .

È chiaro che questa quantità varia con l'input considerato. Si può, infatti, ottenere una funzione che associ ad ogni diversa stringa di input il tempo necessario per il completamento della computazione. Risulta più interessante, però, studiare il tempo richiesto in funzione della lunghezza della stringa in input. Fissata una lunghezza, però, non è più univocamente determinato il tempo di computazione di una macchina di Turing. Bisogna, quindi, operare una scelta su che tempo considerare. In genere si tende a scegliere il tempo massimo, che porta a formulare le seguenti definizioni:

Definizione 2.5: Tempo di calcolo e computazione.

Siano $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ e $T : \mathbb{N} \rightarrow \mathbb{N}$ due funzioni. Sia M una macchina di Turing. Si dice che M computa f in tempo $T(n)$ o T sse per ogni $x \in \{0, 1\}^*$ la macchina M , con input x , dopo al più $T(|x|)$ passi, arriva allo stato q_{halt} con $f(x)$ stampato sul nastro di output.

Si dice che M computa f sse esiste una tale funzione T per cui M computa f in tempo T .

Nella definizione precedente si è usata la notazione $|x|$ per indicare la lunghezza della stringa x .

In maniera analoga si può definire il tempo di calcolo per le macchine di Turing o persino per i linguaggi. Si dice, infatti, che una macchina M opera in tempo $T : \mathbb{N} \rightarrow \mathbb{N}$ sse, per ogni input $x \in \{0, 1\}^*$, M raggiunge lo stato q_{halt} in al più $T(|x|)$ passi. Analogamente un linguaggio L è deciso in tempo $T(n)$ sse esiste una macchina di Turing M che decide L e opera in tempo T .

L'esatto comportamento di T , però, risulta spesso superfluo, soprattutto per dati in input di grandi dimensioni. È, infatti, sufficiente conoscerne il termine che cresce più velocemente per poter confrontare algoritmi e problemi distinti.

Inoltre, cambiando leggermente il modello su cui eseguire la computazione, per esempio aggiungendo lettere all'alfabeto di una macchina di Turing, il singolo algoritmo viene eseguito in tempi diversi. Spesso questa differenza è la sola moltiplicazione per una costante, che non influisce sulla scala degli infiniti.

Per tenere conto di queste richieste si introduce la seguente notazione.

Definizione 2.6: notazione asintotica.

Siano $f, g : \mathbb{N} \rightarrow \mathbb{R}$, si introducono le seguenti notazioni:

O grande: Si dice che f è O grande di g , notazione $f = O(g)$, sse

$$\exists n_0 \in \mathbb{N}, c \in \mathbb{R} \text{ t.c. } f(n) \leq cg(n) \quad \forall n \geq n_0. \quad (2.4)$$

Ω grande: Si dice che f è Ω grande di g , notazione $f = \Omega(g)$, sse

$$\exists n_0 \in \mathbb{N}, c \in \mathbb{R} \text{ t.c. } cg(n) \leq f(n) \quad \forall n \geq n_0. \quad (2.5)$$

Θ grande: Si dice che f è Θ grande di g , notazione $f = \Theta(g)$, sse

$$f = O(g) \text{ e } f = \Omega(g) \quad (2.6)$$

Le tre notazioni hanno finalità distinte, ma tutte legate al confronto delle richieste computazionali di algoritmi e problemi distinti.

O grande è usato per confrontare algoritmi differenti. Permette di trovare una semplice, ma sufficientemente accurata, maggiorazione delle richieste computazionali di un dato algoritmo. Essendo una maggiorazione consente di confrontare le richieste massime degli algoritmi, permettendo un'ordinamento semplice di essi.

Ω grande, invece, è utilizzato per studiare interi problemi, indipendentemente dall'algoritmo usato per risolverli, oppure classi di algoritmi. In queste analisi, infatti, è utile individuare minorazioni alle richieste computazionali, per avere una stima della difficoltà del problema, oppure dell'efficienza della classe scelta di algoritmi.

Θ grande, invece, permette di individuare algoritmi che hanno lo stesso comportamento asintotico. Un semplice esempio consiste di implementazioni dello stesso algoritmo in differenti modelli computazionali.

2.4 Classi di complessità

Una classe di complessità è un insieme di problemi decisionali che possono essere decisi, in un dato modello computazionale, entro un fissato limite di risorse. Le prime che saranno definite useranno il modello della macchina di Turing e considereranno limiti sul tempo di computazione.

Si costruisce, innanzitutto, l'importante classe **P** la quale, intuitivamente, rappresenta i problemi che ammettono una soluzione efficiente.

2.4.1 Tempo deterministico e P

Sfruttando la definizione 2.5 per il tempo di calcolo, si introducono le classi di tempo deterministico:

Definizione 2.7: classe DTIME.

Sia $T : \mathbb{N} \rightarrow \mathbb{N}$ una funzione. Si definisce $\mathbf{DTIME}(T)$ la classe di ogni problema decisionale L per il quale esiste una costante $c > 0$ t.c. L è deciso in tempo $c \cdot T$.

La classe $\mathbf{DTIME}(T)$, quindi, rappresenta tutti i linguaggi L per cui esiste una macchina di Turing M che decide L in tempo $t = O(T)$. Si può, quindi, dire che la richiesta di tempo del problema L è $O(T)$.

In base alla precedente definizione si può costruire la classe dei problemi decidibili in modo efficiente, ovvero con richiesta di tempo polinomiale:

Definizione 2.8: classe P.

Si definisce $\mathbf{P} := \cup_{c \geq 1} \mathbf{DTIME}(n^c)$.

Questa classe gioca un ruolo importantissimo nella teoria della complessità computazionale, quindi merita alcune osservazioni. Tramite questa classe si definiscono, infatti, i problemi che ammettono risoluzione efficiente e, a priori, non è ovvio il motivo. Pare strano che un problema la cui richiesta di tempo sia di $O(n^{1000})$ possa essere considerato efficiente. Inoltre le costanti non esplicitate potrebbero rendere un problema in $\mathbf{DTIME}(e^n)$ sostanzialmente più veloce da risolvere, per piccoli input, di un problema in $\mathbf{DTIME}(n^2)$. Nonostante questi criticismi, risulta effettivamente una buona scelta definire la classe dei problemi risolubili in modo efficiente tramite **P**.

Da un punto di vista più pratico si potrebbe, infatti, decidere, di chiamare efficiente un dato algoritmo che sia computabile in tempo, per esempio, n^2 . Sarebbe naturale definire efficienti anche algoritmi che eseguono solamente operazioni efficienti con al più chiamate ad altri algoritmi efficienti. Permettendo questo innestamento si individua esattamente **P** come classe dei problemi che ammettono risoluzioni efficienti.

Si potrebbe anche criticare, a questa definizione di problema efficientemente risolubile, la dipendenza da uno specifico modello computazionale. Nulla garantisce che i problemi in **P** siano efficientemente risolubili in modelli computazionali come i circuiti classici o quantistici, per non citare modelli più esotici. Tuttavia esiste una forma forte della tesi di Church-Turing, che vuole risolvere proprio questo problema.

Tesi forte di Church-Turing: Ogni modello computazionale fisicamente realizzabile può essere simulato da una macchina di Turing con al più un incremento polinomiale nelle richieste computazionali.

La veridicità di questa tesi, infatti, garantirebbe la buona definizione classe **P** come classe dei problemi efficientemente risolubili. Essa risulterebbe, infatti, indipendente dal modello computazionale.

2.4.2 Classe NP e tempo non deterministico

La seguente classe svolge un ruolo strettamente legato ai problemi che ammettono soluzione efficiente. Si tratta, infatti, dei problemi per i quali si può verificare in modo efficiente la veridicità di una soluzione. In termini più precisi: sono i problemi decisionali per i quali, dato un x che verifica il problema – ovvero che è nel linguaggio – si ha una stringa di dimensioni non troppo grandi che certifica in modo efficiente l'appartenenza di x in L . La definizione formale è la seguente:

Definizione 2.9: classe NP.

Si definisce **NP** la classe contenente tutti i linguaggi $L \subseteq \{0,1\}^*$ tali per cui esistono un polinomio $p: \mathbb{N} \rightarrow \mathbb{N}$ e una macchina di Turing M , di tempo polinomiale, tali che, per ogni $x \in \{0,1\}^*$:

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ t.c. } M(x, u) = 1 \quad (2.7)$$

Se $x \in L$, ogni $u \in \{0,1\}^{p(|x|)}$ che soddisfa $M(x, u) = 1$ viene detto certificato per x (rispetto al linguaggio L e alla macchina di Turing M).

Questa è la prima definizione che sarà data di **NP** e risulta più intuitiva più intuitiva della successiva. L'importanza di questa definizione, ai fini pratici, può essere spiegata con l'esempio della crittografia. Si vuole, infatti, che il problema di decifrare un messaggio crittato sia in **NP**. Fissati un messaggio cifrato e in chiaro, la situazione migliore sarebbe che, senza ulteriori informazioni, il problema di decidere se il primo viene decodificato nel secondo sia difficile. Si vuole, però, che il possessore della chiave di cifratura possa verificare l'affermazione precedente in modo efficiente. La chiave di cifratura, dunque, opererebbe da certificato, rendendo facile il controllo.

Il fatto che esistano dei tali problemi, che siano effettivamente difficili da risolvere in mancanza del certificato, ma facili in sua presenza, è ancora un problema irrisolto. Per risolverlo è necessario studiare la relazione insiemistica che lega **P** ed **NP**, argomento approfondito nella prossima sezione.

Si presentano degli ulteriori esempi, basati sulla precedente definizione, per chiarificare il concetto di problema in **NP**:

Somma parziale: data una sequenza di n numeri a_1, \dots, a_n e un valore S , decidere se esiste una sottosequenza di a_1, \dots, a_n la cui somma è S . Il certificato è la sequenza di numeri.

Commesso viaggiatore: i dati sono un insieme di n nodi, $\binom{n}{2}$ numeri $d_{i,j}$, i quali indicano la distanza tra i nodi i e j , e un numero k . Il problema chiede di decidere se esiste un circuito chiuso che visita ogni nodo e che sia lungo al più k . Il certificato è il circuito.

Numero composito: dato un numero N decidere se è un numero composito o primo. Il certificato è la fattorizzazione di N .

Fattorizzazione: dati tre numeri N, L, U decidere se N ammette un fattore M nell'intervallo $[L, U]$. Il certificato è il fattore M di N .

Prima di analizzare le relazioni tra le classi di complessità appena definite è utile introdurre un nuovo modello computazionale e una definizione equivalente di **NP**:

Macchina di Turing non deterministica

Riprendendo la definizione 2.1 di macchina di Turing si introduce la seguente:

Definizione 2.10: Macchina di Turing non deterministica.

Una macchina di Turing non deterministica N a k nastri è una tripletta ordinata (Γ, Q, δ) , dove:

- Γ è l'insieme, detto alfabeto di N , con i simboli che possono essere salvati nelle celle dei nastri. Contiene i simboli \square , che indica la cella vuota, e \triangleright , che indica l'inizio del nastro.
- Q è l'insieme degli stati in cui può trovarsi N . Vi appartengono gli stati q_{start} e q_{halt} , che rappresentano l'inizio e la fine della computazione.
- $\delta \subseteq [Q \times \Gamma^k] \times [Q \times \Gamma^{k-1} \times \{L, S, R\}^k]$ è detta relazione di transizione e regola il comportamento della macchina di Turing.

Come nel caso deterministico si usa il primo nastro come solo input, l'ultimo come nastro di output e gli altri come nastri di lavoro. L'unica differenza dalla definizione 2.1 consiste nel fatto che δ non è più una funzione, ma una relazione. In questo dettaglio risiede il non determinismo del nuovo modello computazionale. Infatti, a partire da una data configurazione di N – stato della macchina, posizione delle testine e contenuti dei nastri – non è determinato in modo univoco la configurazione seguente. Fissata una prima configurazione, infatti, si apre un albero, che rappresenta le possibili evoluzioni della computazione, come rappresentato nella figura 2.1. Tale differenza dal modello deterministico comporta una nozione completamente differente di risoluzione di un problema decisionale. Infatti si dice che la macchina di Turing non deterministica N decide il problema decisionale L sse: (1) per ogni $x \in L$ esiste una successione di scelte non deterministiche che risulta in una computazione in cui N raggiunge lo stato q_{halt} e riporta sul nastro di output 1, (2) per ogni $x \notin L$ non esiste alcuna sequenza di scelte come descritte nel caso (1).

Si ha un comportamento asimmetrico tra (1) e (2), simile alla definizione di accettazione di un linguaggio da parte di una macchina di Turing deterministica M .

Si dice, inoltre, che una macchina di Turing non deterministica N decide il linguaggio L in tempo $T : \mathbb{N} \rightarrow \mathbb{N}$ sse N decide L e, per ogni stringa x in input, ogni possibile scelta non deterministica porta a computazioni che raggiungono lo stato q_{halt} in un numero di passi $k \leq T(|x|)$. Nella rappresentazione grafica $T(|x|)$ è una maggiorazione della profondità dell'albero delle possibili computazioni non deterministiche. È interessante notare che, ad un fissato livello di profondità, il modello di macchina di Turing non deterministica esplora un insieme di possibili configurazioni esponenzialmente più grande rispetto alla macchina di Turing deterministica.

Si possono, ora, introdurre le classi di tempo non deterministico:

Definizione 2.11: classe NTIME.

Sia $T : \mathbb{N} \rightarrow \mathbb{N}$ una funzione. Si definisce $\mathbf{NTIME}(T)$ la classe di ogni problema decisionale L per il quale esistono una costante $c > 0$ e una macchina di Turing non deterministica N t.c. N decide L in tempo $c \cdot T$.

Ovvero la trasposizione non deterministica della definizione 2.7. In queste due definizioni, infatti, **D** indica il determinismo e **N** il non determinismo del modello computazionale utilizzato.

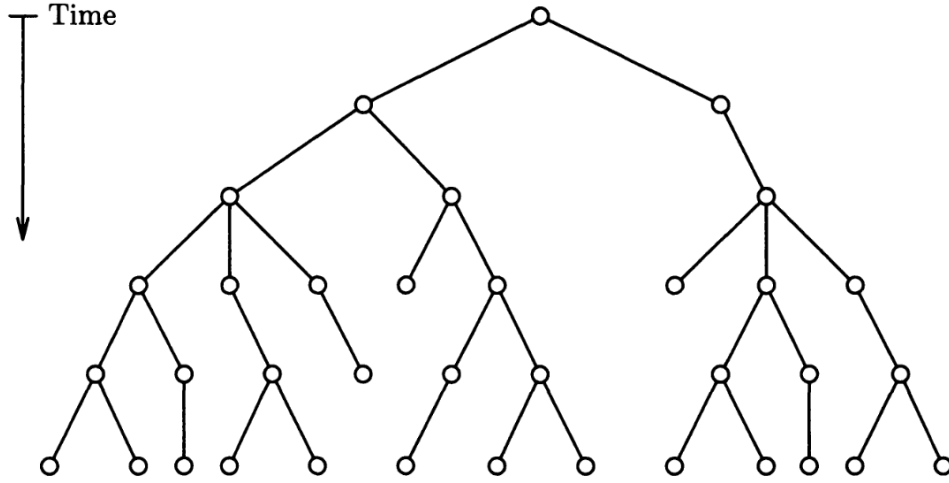


Figura 2.1: Computazione non deterministica.

Si può dare, analogamente al caso di \mathbf{P} , la definizione di problema risolubile in tempo polinomiale da una macchina non deterministica. In questo caso – come storicamente – \mathbf{NP} indica Non deterministic \mathbf{P} .

Definizione 2.12: classe \mathbf{NP}_s .

Si definisce $\mathbf{NP}_s := \bigcup_{c \geq 1} \mathbf{NTIME}(n^c)$.

Si è scelto il pedice s per indicare che questa è la definizione storica della classe \mathbf{NP} . Si hanno, ora, due definizioni indipendenti di \mathbf{NP} , le quali risultano equivalenti:

Teorema 2.4.1 (equivalenza delle definizioni di \mathbf{NP}). $\mathbf{NP}_s = \mathbf{NP}$

Dimostrazione. La dimostrazione si basa sul fatto che la successione di scelte che porta la macchina di Turing non deterministica a verificare l'input x può essere usata come un certificato.

$\mathbf{NP}_s \subseteq \mathbf{NP}$: Sia $L \in \mathbf{NP}_s$, allora esistono un polinomio $q : \mathbb{N} \rightarrow \mathbb{N}$ e una macchina di Turing non deterministica N tali che per ogni input x , in un numero di passi inferiore a $q(|x|)$, N giunge allo stato q_{halt} . In particolare, se $x \in L$, per una particolare sequenza di scelte non deterministiche, N stampa in output il valore 1. Si consideri una codifica binaria della stringa che descrive le scelte. Se ne può individuare una tale che la lunghezza della codifica binaria sia al più $p(|x|) := c \cdot q(|x|)$ per ogni x , con $c \in \mathbb{N}$ una costante.

Questa stringa opera come certificato per la definizione 2.9. Si può ora usare una macchina di Turing M per simulare la macchina N , seguendo le scelte codificate dal certificato. Chiaramente questa simulazione può essere svolta con un solo incremento polinomiale nel tempo q impiegato da N . M , in questo caso, verifica in tempo polinomiale che $x \in L$. Se, inoltre, $x \notin L$ si può prendere una qualsiasi sequenza di scelte non deterministiche e M arriverà allo stato q_{halt} , sempre in tempo polinomiale. La coppia p, M , dunque, soddisfa la definizione citata, per cui $L \in \mathbf{NP}$.

$\text{NP}_s \supseteq \text{NP}$: Sia $L \in \text{NP}$, si considerino il polinomio $p : \mathbb{N} \rightarrow \mathbb{N}$ e la macchina di Turing M come da definizione 2.9. Per ogni $x \in L$ esiste un certificato $u \in \{0, 1\}^{p(|x|)}$ per cui $M(x, u) = 1$. Si definisce la macchina di Turing non deterministica N di $k + 1$ nastri, dove k è il numero di nastri di M , che opera nel seguente modo: Per le prime $p(|x|)$ operazioni scrive un carattere scelto casualmente tra 0 e 1 sul secondo nastro, poi opera deterministicamente. In particolare, nel secondo periodo, opera come la macchina di Turing M , considerando i primi due nastri come nastri di input (come se fossero concatenati). In quanto esiste un certificato u di lunghezza $p(|x|)$ per cui $M(x, u) = 1$, allora esiste una sequenza di scelte non deterministiche per cui N stampa u sul secondo nastro e arriva a stampare 1. In quanto la macchina M è polinomiale segue che anche N lo sarà: esegue esattamente $p(|x|)$ operazioni più di quelle di M . Si è, quindi, verificato che $L \in \text{NP}_s$. ■

2.4.3 Relazione tra P ed NP

In quanto la definizione di macchina di Turing non deterministica è un'estensione di quella deterministica, risulta chiaro che ogni macchina di Turing M è anche non deterministica (semplicemente non è molto interessante come tale). È, quindi, chiaro che, per ogni funzione $T : \mathbb{N} \rightarrow \mathbb{N}$, si ha $\text{DTIME}(T) \subseteq \text{NTIME}(T)$. Da questo segue che $\text{P} \subseteq \text{NP}$.

Purtroppo questo è il più raffinato risultato conosciuto per quanto riguarda la relazione tra P ed NP . Uno dei 7 problemi del millennio, infatti, chiede di stabilire se, effettivamente, $\text{P} \neq \text{NP}$, ma rimane tutt'ora irrisolto.

Anche riprendendo i pochi esempi descritti nella sezione precedente si nota l'incompletezza delle informazioni a nostra disposizione:

Commesso viaggiatore: Si può dimostrare che è un problema di “difficile” soluzione. In particolare si dimostra che è in P sse $\text{P} = \text{NP}$.

Numero composto: È stato recentemente mostrato, in [AKS04], che questo problema è in P .

Fattorizzazione: Tutt'ora non si sa se sia, o meno, in P o se sia “difficile” quanto il problema del commesso viaggiatore. Si suppone, assumendo $\text{P} \neq \text{NP}$, che sia in $\text{NP} \setminus \text{P}$. È per questa ragione che esistono sistemi crittografici basati sul problema della fattorizzazione degli interi.

2.5 Classi di complessità quantistica

Per ora è stata introdotta la sola teoria classica. Per passare allo studio degli algoritmi che seguiranno è necessario formalizzare il modello dei circuiti, in particolare quantistici, e introdurre le definizioni, che riprendano quelle date nelle sezioni precedenti, per richieste computazionali e classi di complessità.

2.5.1 Circuiti Booleani e complessità

Nel primo capitolo si è data una nozione informale e operativa di circuito, per comprendere il funzionamento di un computer quantistico. Segue una definizione più formale dei circuiti classici utilizzati per computare funzioni di Boole, associate ai problemi decisionali.

Definizione 2.13: Circuito booleano.

Si definisce circuito booleano con n bit di input un grafo aciclico diretto nel quale i nodi possono essere:

- nodi di input, di grado entrante 0, etichettati da uno degli n bit dell'input,
- porte logiche, con grado entrante 1 o 2, scelte tra i gate AND, OR e NOT.

Tra questi nodi si sceglie un gate logico come porta di output del circuito.

Ogni nodo con grado entrante maggiore di 1 è detto gate logico o porta logica. Si indica con $C(x)$ l'output del circuito C a partire dall'input x . Si definiscono, inoltre, *dimensione* del circuito il numero di porte logiche che possiede e *profondità* del circuito la massima lunghezza di un cammino da una porta di input alla porta di output.

Risulta chiaro che, ogni circuito booleano con n bit di input, computa una funzione booleana $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Si può, quindi, pensare di utilizzare dei circuiti per decidere linguaggi, o problemi decisionali. In generale, però, un linguaggio – o la sua codifica binaria – contiene elementi di diversa lunghezza, per cui non basta un singolo circuito per decidere un intero linguaggio. Si deve, infatti, introdurre la nozione di *famiglia di circuiti*. Essa è una successione $\{C_n\}_{n \in \mathbb{N}}$ di circuiti booleani, in cui il circuito C_n opera su input di lunghezza binaria n .

Dato un linguaggio L , quindi, esiste sempre una famiglia di circuiti $\{C_n\}_{n \in \mathbb{N}}$ che decide il linguaggio L , ovvero che ne computa la funzione booleana associata. Si dice che questa famiglia è di *dimensione minimale* (rispettivamente di *profondità minimale*) se non esistono famiglie, che decidono lo stesso linguaggio, di dimensione (rispettivamente profondità) minore.

A partire da questi concetti, quindi, si dà la seguente definizione:

Definizione 2.14: Complessità della dimensione dei circuiti.

Dato un linguaggio formale L si definisce complessità della dimensione dei circuiti la funzione $T : \mathbb{N} \rightarrow \mathbb{N}$, che associa ad ogni n la dimensione di C_n , ovvero dell' n -esimo circuito della famiglia di dimensione minimale che decide L .

Sostituendo profondità a dimensione si ottiene l'analoga definizione per la *complessità di profondità* dei circuiti.

Il fatto che input di lunghezza diversa siano analizzati da circuiti distinti rende questo modello di computazione non uniforme. A priori, infatti, in una famiglia di circuiti, non è richiesta alcuna relazione tra i vari elementi della famiglia. Per studiare le classi di complessità originate da questo modello computazionale si introduce il concetto di uniformità. Intuitivamente una famiglia è uniforme se esiste un algoritmo che descriva, in funzione della sola dimensione n dell'input, come costruire il corrispondente circuito C_n . Inoltre si distinguono vari tipi di uniformità in funzione delle risorse impiegate dall'algoritmo richiesto.

In particolare saranno utili le seguenti famiglie uniformi di circuiti (non solo booleani):

Definizione 2.15: Famiglia uniforme in tempo polinomiale.

Una famiglia di circuiti $\{C_n\}_{n \in \mathbb{N}}$ si dice uniforme in tempo polinomiale sse esiste una macchina di Turing deterministica M tale che:

- M opera in tempo polinomiale,
- per ogni $n \in \mathbb{N}$ la macchina M produce una descrizione del circuito C_n sull'input della stringa contenente esattamente n simboli 1, denotata con 1^n .

2.5.2 Circuiti quantistici

In analogia ai circuiti booleani si definiscono i circuiti quantistici:

Definizione 2.16: Circuito quantistico.

Si definisce circuito quantistico con n qubit di input un grafo aciclico diretto nel quale i nodi possono essere:

- nodi di input, di grado entrante 0, etichettati da uno degli n qubit dell'input,
- porte logiche, con grado entrante uguale a quello uscente, scelte in una fissata famiglia di gate.

Di questi nodi si scelgono m nodi con grado entrante maggiore di zero, ai quali si associa l'output del circuito.

Dai nodi di output si vuole ottenere dell'informazione classica. È necessario, quindi, far seguire la porta logica di output da un'opportuna misura nella base computazionale, nell'implementazione del circuito. Per un circuito QC si indica l'output con $QC(x)$, dove x è lo stato quantistico di input. La famiglia di gate tra cui scegliere le porte logiche potrebbe essere finita o infinita. Nella realizzazione fisica si è più portati a scegliere famiglie finite di porte logiche, per ovvie questioni pratiche. A tal fine è importante lo studio di quelle che sono chiamate famiglie di gate universali, svolto in [NC11]. A differenza dei gate universali classici, che possono computare ogni funzione a valori in stringhe binarie, le famiglie universali quantistiche possono *approssimare*, con arbitraria precisione, qualsiasi gate realizzabile.

Per scopi più teorici, invece, è utile considerare la famiglia infinita contenente tutti i gate su un singolo qubit e il gate CNOT. Essa, infatti, si dimostra essere sufficiente a simulare il comportamento di qualsiasi altro gate quantistico. Per approfondire questo argomento si rimanda a [NC11].

Classi di complessità

Su questo modello computazionale si ripropongono, senza alterazioni, le nozioni di complessità della dimensione e della profondità dei circuiti. Si riporta, per chiarezza, la prima:

Definizione 2.17: Complessità della dimensione dei circuiti.

Dato un linguaggio formale L si definisce complessità della dimensione dei circuiti la funzione $T : \mathbb{N} \rightarrow \mathbb{N}$, che associa ad ogni n la dimensione di QC_n , ovvero dell' n -esimo circuito della famiglia di dimensione minimale che decide L .

Questa stima sarà spesso utilizzata nel capitolo seguente, per analizzare l'efficienza degli algoritmi sviluppati. In particolare si userà la notazione asintotica descritta nella definizione 2.6. Si dirà, dunque, che un algoritmo ha un costo computazionale di $O(T)$ gate per affermare che la corrispondente famiglia di circuiti ha complessità di dimensione $F = O(T)$.

Come per la complessità, anche i concetti di famiglia uniforme e, in particolare, di famiglia uniforme in tempo polinomiale, definizione 2.15, si traspongono senza alterazione alcuna.

Si possono, ora, definire le classi di complessità quantistiche equivalenti a **P** ed **NP**. La differenza principale con le precedenti, a meno del differente modello computazionale, risiede nella natura aleatoria della computazione quantistica. Si sarà, infatti, costretti a richiedere una minima probabilità di successo degli algoritmi.

Definizione 2.18: classe BQP.

La classe **BQP** contiene tutti i problemi decisionali L che ammettono una famiglia uniforme in tempo polinomiale di circuiti quantistici $\{QC_n\}_{n \in \mathbb{N}}$ tale che:

- Per ogni n il circuito QC_n prende in input n qubit e restituisce in output un singolo bit,
- Per ogni $x \in L$ vale $\mathbb{P}(QC_{|x|}(x) = 1) \geq 2/3$
- Per ogni $x \notin L$ vale $\mathbb{P}(QC_{|x|}(x) = 0) \geq 2/3$

La sigla **BQP** rappresenta: “Bounded-error Quantum Polynomial time”. Si richiede, infatti, l’esistenza di una famiglia di circuiti, di cui si può ottenere una descrizione in tempo polinomiale tramite una macchina di Turing e che ha una probabilità di errore limitata. Si noti che il valore $2/3$ è, fondamentalmente, simbolico. La ripetizione, di un algoritmo che rispetti tale condizione, un numero costante di volte diminuisce esponenzialmente la probabilità di errore – ogni ripetizione è indipendente – rispettando comunque le condizioni della definizione.

Analogamente alla classe **P** si considerano i problemi in **BQP** come efficientemente risolubili da una computer quantistico. Tutti gli algoritmi che verranno studiati nel prossimo capitolo soddisfano le condizioni appena esposte, ovvero risolvono problemi in questa classe di complessità. Intuitivamente ciò è dovuto al fatto che si riesce a dare una descrizione esplicita dei circuiti che li implementano, indipendentemente dalla lunghezza degli input, in questa breve trattazione.

Mentre, come è reso evidente dai risultati trovati da Fredkin e Toffoli è noto che $\mathbf{P} \subseteq \mathbf{BQP}$, attualmente non è conosciuta alcuna relazione tra le classi **NP** e **BQP**. Ci sono, però, evidenze che $\mathbf{NP} \not\subseteq \mathbf{BQP}$. Inoltre, per quanto un computer quantistico possa essere fondamentalmente più veloce di un computer classico, è stato dimostrato che non sarà esponenzialmente più veloce di esso.

Si passa, ora, all’analogo quantistico di **NP**:

Definizione 2.19: classe QMA.

Si definisce **QMA** la classe contenente tutti i linguaggi $L \subseteq \{0, 1\}^*$ tali per cui esistono un polinomio $p : \mathbb{N} \rightarrow \mathbb{N}$ e una famiglia uniforme in tempo polinomiale di circuiti quantistici $\{QC_n\}_{n \in \mathbb{N}}$ tali che:

$$\forall x \in L \exists |\psi\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ t.c. } \mathbb{P}(QC_{|x|}(|x\rangle, |\psi\rangle) = 1) \geq \frac{2}{3} \quad (2.8)$$

$$\forall x \notin L, \forall |\psi\rangle \in (\mathbb{C}^2)^{\otimes p(|x|)} \text{ si ha } \mathbb{P}(QC_{|x|}(|x\rangle, |\psi\rangle) = 1) \leq \frac{1}{3} \quad (2.9)$$

Tale definizione riprende la definizione deterministica di problemi per i quali esiste un certificato efficientemente verificabile. Come nel caso di **BQP** le differenze principali dalla definizione classica sono le seguenti. Innanzitutto si ammette una probabilità di errore per la natura probabilistica della computazione quantistica. Inoltre si richiede che il certificato sia di natura quantistica, ovvero un $p(|x|)$ -qubit.

Capitolo 3

Algoritmi quantistici

Con gli strumenti sviluppati fin'ora si possono, finalmente, studiare i primi algoritmi quantistici. Innanzitutto verranno presentati degli algoritmi che sfruttano il comportamento quantistico del modello computazionale esibendo comportamenti impossibili in quello classico. In seguito verranno trattati degli algoritmi per la risoluzione di importanti problemi dal punto di vista pratico. Tali algoritmi si baseranno sulla *trasformata di Fourier quantistica*, una riscrittura in termini quantistici della *trasformata di Fourier discreta*, alla base di molti algoritmi efficienti in termini classici. È notevole osservare che tutti questi algoritmi mostreranno richieste computazionali decisamente minori, a volte anche in modo esponenziale, rispetto alle migliori controparti classiche.

3.1 Fenomeni quantistici

Come introdotto nella sezione 1.5 il modello di computazione quantistica gode di svariate peculiarità rispetto alla computazione classica. La sezione citata si concentra sulle mancanze della prima rispetto all'ultima, in questa, invece, si mostreranno alcuni fenomeni che non sono replicabili classicamente. In particolare si tratteranno alcuni possibili vantaggi ottenibili con il nuovo paradigma di computazione.

3.1.1 Stati di Bell

In	Out
$ 00\rangle$	$(00\rangle + 11\rangle) / \sqrt{2} =: \beta_{00}\rangle$
$ 01\rangle$	$(01\rangle + 10\rangle) / \sqrt{2} =: \beta_{01}\rangle$
$ 10\rangle$	$(00\rangle - 11\rangle) / \sqrt{2} =: \beta_{10}\rangle$
$ 11\rangle$	$(01\rangle - 10\rangle) / \sqrt{2} =: \beta_{11}\rangle$

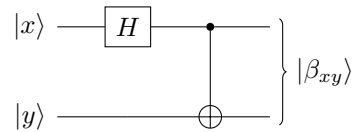


Figura 3.1: Circuito che costruisce gli stati di Bell a partire dagli elementi della base computazionale. Nella tabella sono elencati gli output del circuito, al variare dei valori in input.

A partire dagli elementi della base computazionale il circuito rappresentato in figura 3.1 produce i seguenti stati:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (3.1)$$

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \quad (3.2)$$

$$|\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \quad (3.3)$$

$$|\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \quad (3.4)$$

in cui la notazione $|\beta_{xy}\rangle$, oltre a rappresentare gli stati di partenza su cui opera il circuito 3.1, può essere usata per ottenere l'esplicita espressione del corrispondente stato di Bell tramite la seguente espressione

$$|\beta_{xy}\rangle = \frac{|0y\rangle + (-1)^x |1\bar{y}\rangle}{\sqrt{2}}, \quad (3.5)$$

dove $\bar{y} := \neg y$ è la negazione di y .

Questi stati sono noti con il nome di *stati di Bell* o *stati EPR* in onore di Bell, Einstein, Podolsky e Rosen, i quali per primi hanno studiato le proprietà di questi speciali stati. Essi, infatti, sono stati entangled, come è stato mostrato per $|\beta_{00}\rangle$ nella sezione 1.1.3, e formano una base per \mathbb{C}^4 . Quest'ultima affermazione, in particolare, è evidente dal fatto che sono immagini tramite un circuito quantistico, ovvero un operatore unitario, degli elementi della base computazionale.

Questi stati, inoltre, sono legati dalle seguenti relazioni, che implicitamente giocheranno un ruolo molto importante nelle prossime sezioni:

$$Id \otimes Id |\beta_{00}\rangle = Id \otimes Id |\beta_{00}\rangle = |\beta_{00}\rangle; X \otimes Id |\beta_{00}\rangle = Id \otimes X |\beta_{00}\rangle = |\beta_{01}\rangle; \quad (3.6)$$

$$Z \otimes Id |\beta_{00}\rangle = Id \otimes Z |\beta_{00}\rangle = |\beta_{10}\rangle; iY \otimes Id |\beta_{00}\rangle = Id \otimes iY |\beta_{00}\rangle = |\beta_{11}\rangle \quad (3.7)$$

dove Id, X, Y e Z sono le matrici di Pauli, per cui

$$Id = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}; \quad iY = ZX = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \quad (3.8)$$

3.1.2 Teletrasporto quantistico

Si sfrutteranno, ora, le peculiarità degli stati di Bell per ottenere un risultato sorprendente e di grande utilità per la comprensione del funzionamento dei computer quantistici. L'obiettivo del seguente algoritmo, detto di *teletrasporto quantistico*, è di trasmettere lo stato $|\psi\rangle$ di un qubit tra due computer quantistici, rispettivamente in mano ad Alice e Bob, scambiando solamente 2 bit di informazione classica. Senza un precedente scambio di informazioni tale processo è chiaramente impossibile, in quanto, come evidenziato nelle sezioni precedenti, lo stato di un qubit è descritto da 3 valori reali, che richiederebbero una quantità infinita di bit classici per essere descritti.

Per sorvolare questo problema si crea uno stato entangled di Bell, per semplicità $|\beta_{00}\rangle$, e si dà il primo qubit di questa coppia ad Alice e il secondo a Bob. Per quanto i due computer quantistici possano essere separati da una grande distanza spaziale, si

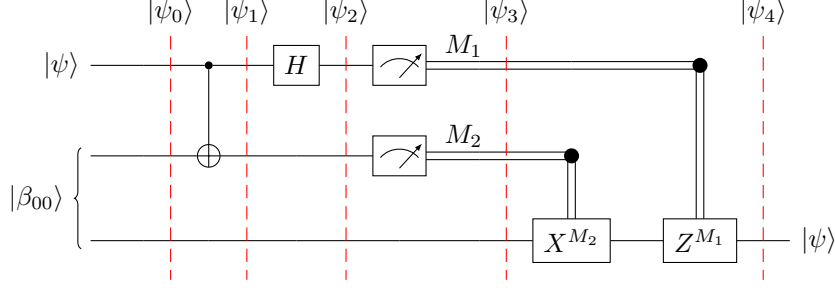


Figura 3.2: Circuito per l'algoritmo di teletrasporto quantistico. Lo stato $|\psi\rangle$ passa dal primo all'ultimo registro, tramite sola comunicazione classica.

descriverà il sistema con un unico stato dei 3-qubit, in cui, inizialmente, il primo (da trasmettere) sarà nello stato $|\psi\rangle = a|0\rangle + b|1\rangle$ e gli altri due in quello entangled $|\beta_{00}\rangle$. Di questo stato i valori a e b sono arbitrari e, a priori, sconosciuti. Si assume, inoltre, che i primi due qubit siano in possesso di Alice e solo il terzo di Bob. Lo stato iniziale, quindi, sarà

$$|\psi_0\rangle := |\psi\rangle |\beta_{00}\rangle = \frac{1}{\sqrt{2}} \{a|0\rangle(|00\rangle + |11\rangle) + b|1\rangle(|00\rangle + |11\rangle)\}, \quad (3.9)$$

come indicato in figura 3.2. Inizialmente Alice agisce sulla propria coppia di qubit, prima con un gate CNOT, ad ottenere lo stato

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \{a|0\rangle(|00\rangle + |11\rangle) + b|1\rangle(|10\rangle + |01\rangle)\} \quad (3.10)$$

e, poi, con un gate Hadamard sul primo qubit, ad ottenere

$$|\psi_2\rangle = \frac{1}{2} \{a(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + b(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)\}. \quad (3.11)$$

Questo stato, in particolare, può essere riscritto raggruppando i primi 2 qubit, in possesso di Alice, come segue

$$\begin{aligned} |\psi_2\rangle = \frac{1}{2} \{ & |00\rangle (a|0\rangle + b|1\rangle) + |01\rangle (a|1\rangle + b|0\rangle) \\ & + |10\rangle (a|0\rangle - b|1\rangle) + |11\rangle (a|1\rangle - b|0\rangle) \}. \end{aligned} \quad (3.12)$$

In questo stato, Alice, misurando la propria coppia di qubit, forzerà, a causa dell'entanglement della coppia di Bell, il bit di Bob ad assumere uno dei 4 stati appena elencati. In particolare, in funzione della misurazione $M_1 M_2$ di Alice, lo stato del qubit di Bob sarà $|\psi_3(M_1 M_2)\rangle$, più esplicitamente:

$$M_1 M_2 = 00 \implies |\psi_3(00)\rangle = (a|0\rangle + b|1\rangle) \quad (3.13)$$

$$M_1 M_2 = 01 \implies |\psi_3(01)\rangle = (a|1\rangle + b|0\rangle) \quad (3.14)$$

$$M_1 M_2 = 10 \implies |\psi_3(10)\rangle = (a|0\rangle - b|1\rangle) \quad (3.15)$$

$$M_1 M_2 = 11 \implies |\psi_3(11)\rangle = (a|1\rangle - b|0\rangle). \quad (3.16)$$

A questo punto, in seguito alla comunicazione della coppia di bit dalla misurazione di Alice dei propri qubit, Bob può provvedere ad operare sul qubit in suo possesso per ottenere $|\psi\rangle$. In particolare dovrà agire con le seguenti operazioni:

$$M_1 M_2 = 00 \implies Id |\psi_3(00)\rangle = (a|0\rangle + b|1\rangle) = |\psi\rangle \quad (3.17)$$

$$M_1 M_2 = 01 \implies X |\psi_3(01)\rangle = (a|0\rangle + b|1\rangle) = |\psi\rangle \quad (3.18)$$

$$M_1 M_2 = 10 \implies Z |\psi_3(10)\rangle = (a|0\rangle + b|1\rangle) = |\psi\rangle \quad (3.19)$$

$$M_1 M_2 = 11 \implies ZX |\psi_3(11)\rangle = (a|0\rangle + b|1\rangle) = |\psi\rangle. \quad (3.20)$$

In particolare, nell'ultima riga si opera con la trasformazione ZX , che corrisponde ad applicare prima il gate X e poi il gate Z , come rappresentato nel circuito 3.2, in cui le operazioni si susseguono da sinistra a destra. Inoltre risulta chiaro dal prospetto precedente che le operazioni corrette per ottenere lo stato $|\psi\rangle$, data la misurazione $M_1 M_2$, sono $Z^{M_1} X^{M_2}$, ovvero quanto rappresentato nel circuito, in cui lo stato finale, correttamente, è proprio $|\psi\rangle$.

Questo interessante fenomeno merita due ulteriori commenti prima di essere abbandonato. Innanzitutto potrebbe parre che, trasportando lo stato del qubit $|\psi\rangle$ da Alice a Bob, se ne stia creando una copia, in violazione del teorema 1.5.1 (no cloning theorem). Ciò non è vero, in quanto, in seguito alla misurazione della coppia di qubit di Alice, il primo registro, in cui era immagazzinato lo stato $|\psi\rangle$, si ritrova in uno dei due stati della base computazionale: $|0\rangle$ o $|1\rangle$. Si ottiene, quindi, una sola copia dello stato $|\psi\rangle$, senza alcuna contraddizione del suddetto teorema.

Un'ulteriore commento, di natura più fisica, riguarda la parvenza che questo circuito permetta, in contraddizione con la teoria della relatività generale, la trasmissione di informazione a velocità superluminale. Questa idea potrebbe essere scatenata dal fatto che la misurazione dei qubit di Alice porta all'immediato "collasso" del qubit di Bob a uno degli stati in (3.13)-(3.16). Anche questa affermazione è solo apparentemente vera: per ottenere lo stato del primo qubit Bob deve attendere l'informazione classica della misurazione effettuata da Alice, la quale non può essere trasmessa a velocità superluminali. In realtà, con strumenti matematici più potenti rispetto a quelli introdotti in questa breve trattazione, si può mostrare che, senza lo scambio dei bit classici, il solo fenomeno di entanglement non trasferisce alcuna informazione – si veda [NC11] alla sezione 2.4.3.

3.1.3 Superdense coding

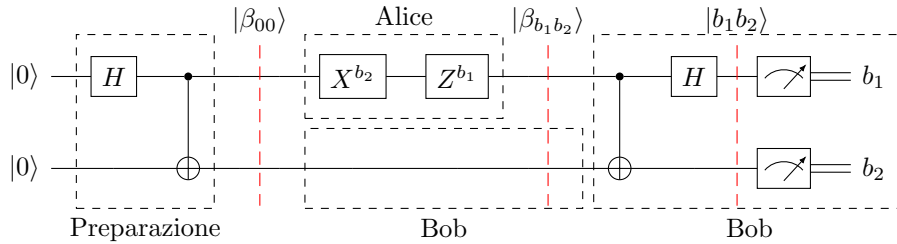


Figura 3.3: Circuito per l'algoritmo di superdense coding. Permette la trasmissione di 2 bit classici tramite lo scambio di 1 solo qubit.

L'algoritmo precedente ha una controparte, che sfrutta sempre gli stati entangled di Bell, per ottenere l'effetto opposto: inviare due bit classici b_1 e b_2 , tramite lo scambio di un singolo qubit da Alice a Bob. Tale algoritmo è chiamato *superdense coding* ed è rappresentato nel circuito in figura 3.3. Per comprenderne appieno il funzionamento si commentano i vari stadi del circuito. Nel riquadro "Preparazione", a partire da $|00\rangle$, viene preparato lo stato $|\beta_{00}\rangle$, come descritto in sezione 3.1.1. Il primo qubit di questo stato viene dato ad Alice, mentre il secondo a Bob. Come descritto dalle identità in (3.6) Alice trasforma il qubit $|\beta_{00}\rangle$ in $|\beta_{b_1 b_2}\rangle$, dove b_1 e b_2 sono i bit classici che vuole trasmettere a Bob. A questo punto Alice invia anche il proprio qubit a Bob, il quale è, ora, in possesso di entrambi i componenti dello stato entangled. Per reversibilità degli operatori quantistici Bob, eseguendo al contrario il circuito quantistico in figura 3.1 ottiene, a partire da $|\beta_{b_1 b_2}\rangle$, lo stato della base computazionale $|b_1 b_2\rangle$. A Bob non resta che misurare lo stato in proprio possesso nella base computazionale per ottenere, con certezza, la coppia di bit $b_1 b_2$ che Alice doveva comunicargli, come ci si era prefissati.

Gli ultimi due algoritmi mostrano come, fondamentalmente, la capacità di memoria di un sistema di n -qubit sia 2^n bit classici, dando alla computazione quantistica, su questo fronte, un vantaggio esponenziale rispetto a quella classica.

3.1.4 Parallelismo quantistico

Una fondamentale caratteristica dei computer quantistici è la possibilità di valutare svariati valori di una funzione binaria contemporaneamente. Questa proprietà è chiamata *parallelismo quantistico* ed è alla base di tutta la sezione 3.2, in cui la trasformata di Fourier quantistica ne farà uso.

Per comprendere appieno le potenzialità, ma soprattutto le limitazioni, di questo peculiare comportamento quantistico è necessario osservarne il funzionamento nel dettaglio. Per chiarezza sarà trattato per primo il caso di una funzione $f : \{0, 1\} \rightarrow \{0, 1\}$ arbitraria. Questa funzione non è necessariamente unitaria (o invertibile), quindi non è immediatamente implementabile quantisticamente. Si può, però, decidere di tenere in memoria sia il valore in entrata alla funzione, sia l'immagine della funzione. Si crea, quindi, il gate U_f che manda il generico stato $|x, y\rangle$ in $|x, y \oplus f(x)\rangle$, dove \oplus indica la somma modulo 2. Per funzioni binarie come f , inoltre, tale trasformazione è unitaria.

In particolare, se $y = 0$, lo stato $|x, 0\rangle$ verrà mandato da U_f in $|x, f(x)\rangle$, dando esplicitamente il valore della funzione. La parte interessante del procedimento per ottenere il parallelismo quantistico, però, si incontra analizzando x . Infatti, ponendo $|x\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ – facilmente preparabile applicando un gate H a un qubit nello stato $|0\rangle$ – U_f agirà nel seguente modo:

$$|x, 0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} |0\rangle \xrightarrow{U_f} \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}. \quad (3.21)$$

In questo stato sono presenti entrambe le immagini della funzione f e queste sono state computate simultaneamente, da un singolo circuito.

Analogamente, data una funzione $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, si definisce un gate U_f , che opera come

$$|x, y\rangle \xrightarrow{U_f} |x, y + f(x) \bmod 2^m\rangle. \quad (3.22)$$

Scrivendone la rappresentazione come matrice risulta chiaro che è anch'esso unitario. Si può persino mostrare che, se esiste un algoritmo classico per computare $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$, allora l'algoritmo quantistico per computare U_f è di paragonabile efficienza. Per approfondire si rimanda a [NC11], sezione 3.2.5.

In particolare, a partire dallo stato

$$|x\rangle |y\rangle := \left(\frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle \right) |0\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle |0\rangle, \quad (3.23)$$

il gate U_f produrrà lo stato

$$\frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle |f(j)\rangle, \quad (3.24)$$

il quale, come nel caso precedente, contiene tutte le immagini di f . In particolare lo stato $|x\rangle$ iniziale può essere facilmente preparato applicando un gate Hadamard $H^{\otimes n}$ allo stato $|0\rangle \otimes n$.

A seguito di quest'analisi risulta chiaro che il circuito quantistico alla base del parallelismo quantistico corrisponda con quanto rappresentato in figura 3.4. Inoltre, per lo scopo attuale, non è importante conoscere la realizzazione del gate U_f , ma solo il fatto che possa essere realizzato. Per l'applicabilità del processo di parallelismo può essere semplicemente considerato come una scatola nera.

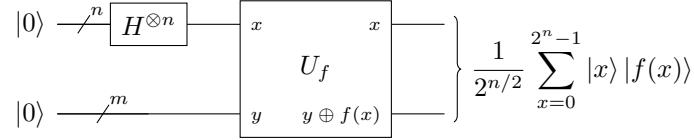


Figura 3.4: Circuito per l'algoritmo di parallelismo quantistico. Crea uno stato finale contenente la valutazione della funzione f su tutti gli elementi del proprio dominio.

Ora che è chiaro il funzionamento dell'algoritmo si possono analizzare più nel dettaglio le sue proprietà. Innanzitutto, indipendentemente dalla grandezza del dominio della funzione f , supposto di aver abbastanza memoria per poter rappresentare tutti gli elementi del dominio, basta un singolo circuito quantistico per operare con il parallelismo. In ambito classico, per valutare la funzione su tutti gli elementi del dominio in contemporanea sarebbe necessario un circuito per ciascun elemento, con una richiesta di risorse fisiche esponenzialmente più grande.

Nonostante questo grandissimo vantaggio, purtroppo, lo stato finale di questo circuito non è immediatamente accessibile: per osservarlo sarebbe necessario misurarlo, facendolo collassare su un solo elemento della sovrapposizione. Ciò restituirebbe come risultato l'immagine di un singolo elemento del dominio, scelto casualmente, cancellando il resto della computazione. Questo fatto comporta la necessità di sfruttare in modo meno diretto la grande quantità di informazioni contenuta nella sovrapposizione. È necessario, infatti, in base al problema da affrontare, operare ulteriormente sullo stato preparato dal parallelismo, per ottenere, con grande probabilità, un risultato corretto. Per mostrare come ciò possa essere messo in atto, nella prossima sezione, verrà analizzato un problema la cui risoluzione quantistica si basa su un ingegnoso uso del parallelismo.

3.1.5 Algoritmo di Deutsch-Josza

L'algoritmo di *Deutsch-Josza* si pone un problema di poca utilità pratica, ma grande utilità didattica. Servirà per comprendere come il parallelismo quantistico possa essere utilizzato per risolvere problemi in cui è necessario valutare una funzione più volte. Innanzitutto è necessario spiegare il problema da risolvere. Sia $f : \{0, 1\}^n \rightarrow \{0, 1\}$ una funzione binaria. Tale funzione deve, inoltre, essere *costante*, ovvero assumere lo stesso valore per ogni elemento del dominio, oppure *bilanciata*, ovvero assumere il valore 1 esattamente la metà delle volte e 0 l'altra metà. L'algoritmo di *Deutsch-Josza*, con una sola esecuzione della scatola nera U_f vuole individuare con certezza se la corrispondente funzione f è bilanciata o costante. Per ottenere tale risultato si opera, come descritto nel circuito della figura 3.5, applicando un circuito di *parallelismo quantistico* e manipolando lo stato ottenuto in modo tale da far interferire tra loro gli elementi della sovrapposizione, fino a poter misurare, con certezza, una risposta al problema.

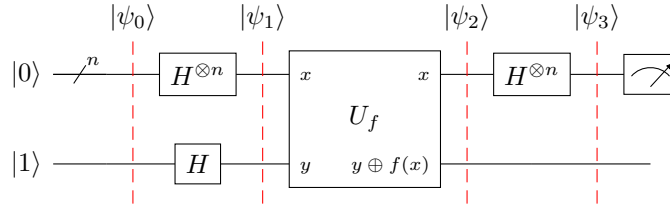


Figura 3.5: Circuito per l'algoritmo di Deutsch-Josza. Riesce a distinguere tra funzioni binarie costanti ed equilibrate.

Per comprendere la breve spiegazione precedente è necessario descrivere passo passo il comportamento del circuito. Lo stato iniziale della computazione è

$$|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle, \quad (3.25)$$

su cui operano i gate Hadamard per creare la sovrapposizione

$$|\psi_1\rangle = \sum_{x \in \{0,1\}^n} \frac{|x\rangle}{2^{n/2}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \quad (3.26)$$

Questo stato è interessante in quanto i primi n qubit, come nell'algoritmo di parallelismo quantistico, sono nella una sovrapposizione uniforme di tutti gli elementi del dominio. L'ultimo bit, invece, è in una combinazione delle immagini possibili, non solo nello stato $|0\rangle$, come ci si aspetterebbe per operare con il parallelismo. Questo fa intuire che si voglia sfruttare una particolare sovrapposizione per ottenere una risposta al problema.

Fissato $x \in \{0, 1\}^n$, posto $y := (|0\rangle - |1\rangle) / \sqrt{2}$, l'azione di U_f su $|x, y\rangle$ diventa

$$|x, y\rangle \xrightarrow{U_f} |x\rangle \left[\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right] = (-1)^{f(x)} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \quad (3.27)$$

Segue che, dopo l'applicazione del gate U_f , lo stato del circuito diventa

$$|\psi_2\rangle = \sum_{x \in \{0,1\}^n} \frac{(-1)^{f(x)} |x\rangle}{2^{n/2}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \quad (3.28)$$

In questo stato l'informazione dell'immagine di f non è accessibile solo dall'ultimo qubit, ma anche dai primi n . Si può, quindi, operare per fare interferire i termini della combinazione lineare dei primi n qubit per ottenere una risposta al problema. Per raggiungere questo obiettivo, nel circuito rappresentato, si opera con un gate di Hadamard, arrivando allo stato $|\psi_3\rangle$. Al fine di calcolarne la forma esplicita si ricorda l'espressione per $H^{\otimes n}$ individuata in (1.8), per cui, dato $z \in \{0,1\}^n$,

$$H^{\otimes n} |z\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot z} |x\rangle, \quad (3.29)$$

dove $x \cdot z$ rappresenta il prodotto bit per bit, modulo 2, tra numeri binari. Questa rappresentazione si presta molto bene alla scrittura esplicita di $|\psi_3\rangle$, il quale risulta essere

$$|\psi_3\rangle = \sum_{z \in \{0,1\}^n} \sum_{x \in \{0,1\}^n} \frac{(-1)^{x \cdot z + f(x)} |z\rangle}{2^n} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]. \quad (3.30)$$

In cui, in particolare, il coefficiente dello stato $|0\rangle^{\otimes n}$ del primo registro è $(\sum_x (-1)^{f(x)}) / 2^n$. In base al fatto che f sia costante o bilanciata questa espressione assumerà due possibili valori. Se f è costante, allora il termine $|0\rangle^{\otimes n}$ avrà coefficiente $+1$ o -1 , in base al valore di f . Segue che ogni altro termine nella combinazione lineare dello stato $|\psi_3\rangle$ sarà 0. Se, invece, f è bilanciata allora lo stato $|0\rangle^{\otimes n}$ avrà coefficiente 0, in quanto i termini positivi e negativi sono in egual numero. In particolare, eseguendo una misura nella base computazionale dei primi n qubit, se f è costante si otterrà con certezza 0, altrimenti, con certezza, si otterrà un risultato diverso da 0, permettendo di distinguere tra i due casi possibili. Si è riusciti, quindi, a far interferire i termini della sovrapposizione in modo tale da ottenere facilmente una risposta a un problema richiedente la valutazione di una funzione su tutti gli elementi del dominio. Per riassumere, a meno dei commenti, segue una descrizione dell'algoritmo di Deutsch-Josza.

Algoritmo: Deutsch-Josza.

Input: (1) una scatola nera per eseguire il gate U_f , che opera su $x \in \{0,1\}^n$ e $y \in \{0,1\}$ come $|x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$. Nel gate precedente la funzione è $f : \{0,1\}^n \rightarrow \{0,1\}$ ed è bilanciata o costante. (2) n qubit inizializzati nello stato $|0\rangle$ ed un qubit nello stato $|1\rangle$.

Output: 0 se la funzione f è costante, un valore diverso da 0 se è bilanciata.

Runtime: una singola chiamata alla scatola nera per l'esecuzione del gate U_f . Ritorna sempre un output corretto.

Procedura:

1. $|0\rangle^{\otimes n} |1\rangle$ stato iniziale
2. $\xrightarrow{H^{\otimes n+1}} \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$ sovrapposizione iniziale
3. $\xrightarrow{U_f} \sum_{x=0}^{2^n-1} \frac{(-1)^{f(x)} |x\rangle}{2^{n/2}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$ risultato scatola nera
4. $\xrightarrow{H^{\otimes n}} \sum_{z=0}^{2^n-1} \sum_{x=0}^{2^n-1} \frac{(-1)^{x \cdot z + f(x)} |z\rangle}{2^n} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right]$ sovrapposizione finale
5. $\mapsto z$ misura primo registro

3.2 Trasformata di Fourier quantistica

Classicamente si definisce la *trasformata di Fourier discreta* l'applicazione che associa al vettore (x_0, \dots, x_{N-1}) di lunghezza N il vettore (y_0, \dots, y_{N-1}) , le cui componenti sono determinate dalla seguente espressione:

$$y_k := \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k / N}. \quad (3.31)$$

Analogamente la *trasformata di Fourier quantistica* o QFT (Quantum Fourier Transform) è l'unico operatore lineare così definito sulla base ortonormale $(|0\rangle, \dots, |N-1\rangle)$:

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |k\rangle. \quad (3.32)$$

Da cui si ottiene l'azione della QFT sul generico stato:

$$\sum_{j=0}^{N-1} x_j |j\rangle \mapsto \sum_{k=0}^{N-1} y_k |k\rangle, \quad (3.33)$$

dove le y_k sono le immagini tramite la *trasformata di Fourier discreta* del vettore (x_0, \dots, x_{N-1}) , definite in (3.31).

Non è ovvio dalla definizione, ma tale operatore è unitario:

Lemma 1. *L'operatore trasformata di Fourier quantistica è unitario.*

Dimostrazione. Dalla definizione della *trasformata di Fourier quantistica* risulta chiaro che, nella base $(|0\rangle, \dots, |N-1\rangle)$, ha rappresentazione matriciale data da:

$$F := \frac{1}{\sqrt{N}} \left[e^{2\pi i j k / N} \right]_{j,k} \quad (3.34)$$

che, per commutatività del prodotto, è chiaramente una matrice simmetrica. In quanto, inoltre, $(e^{2\pi i j k / N})^* = e^{-2\pi i j k / N}$, si trova che la matrice associata all'operatore aggiunto di F è:

$$F^\dagger = \frac{1}{\sqrt{N}} \left[e^{-2\pi i j k / N} \right]_{j,k}. \quad (3.35)$$

Prendendo il prodotto di (3.34) e (3.35) si ottiene:

$$[F \cdot F^\dagger]_{l,m} = \frac{1}{N} \sum_{j=0}^{N-1} e^{\frac{2\pi i}{N} j \cdot (l-m)}. \quad (3.36)$$

Se $l = m$, l'ultimo termine di (3.36) diventa

$$\frac{1}{N} \sum_{j=0}^{N-1} 1 = \frac{1}{N} \cdot N = 1, \quad (3.37)$$

altrimenti $l - m \neq 0$, per cui, al variare di $j \in \{0, \dots, N-1\}$, i termini $e^{\frac{2\pi i}{N} j \cdot (l-m)}$ scorrono, magari più di una volta, tutte le radici R -esime dell'unità, ove $R := N/\text{MCD}(N, l-m)$. Inoltre, per ogni $R \in \mathbb{N}$, vale la scomposizione:

$$x^R - 1 = (x-1)(x^{R-1} + x^{R-2} + \dots + 1). \quad (3.38)$$

Se z è una radice R -esima primitiva dell'unità, allora

$$z^R - 1 = 0 = (z-1)(z^{R-1} + z^{R-2} + \dots + 1) \quad (3.39)$$

da cui, per la legge di annullamento del prodotto, segue che, se $l \neq m$,

$$[F \cdot F^\dagger]_{l,m} = \frac{1}{N} \sum_{j=0}^{N-1} e^{\frac{2\pi i}{N} j \cdot (l-m)} = 0. \quad (3.40)$$

Da (3.37) e (3.40) si può concludere (3.36) con

$$[F \cdot F^\dagger]_{l,m} = \frac{1}{N} \sum_{j=0}^{N-1} e^{\frac{2\pi i}{N} j \cdot (l-m)} = \delta_{l,m}. \quad (3.41)$$

In particolare vale (3.41) $\iff F \cdot F^\dagger = Id \iff F$ è un operatore unitario. \blacksquare

Questo lemma ha come conseguenza il fatto che la trasformata di Fourier quantistica sia un gate quantistico ammissibile su n -qubit. In particolare, adesso cercheremo di individuare esplicitamente un circuito quantistico per il calcolo della QFT

Circuito quantistico

Si consideri $N = 2^n$, con $n \in \mathbb{N}$, e un sistema di n -qubit sul cui spazio degli stati si prende la base computazionale $|0 \dots 0\rangle, \dots, |1 \dots 1\rangle$. Usando la rappresentazione binaria per gli elementi della base $j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0 =: j_1 j_2 \dots j_n$ e per le frazioni binarie $0.j_1 j_{l+1} \dots j_m := j_l/2 + j_{l+1}/2^2 + \dots + j_m/2^{m-l+1}$, si ottiene una più agevole forma della QFT:

Lemma 2 (rappresentazione prodotto). *La trasformata di Fourier quantistica ha la seguente rappresentazione euivalente:*

$$|j_1 j_2 \dots j_n\rangle \mapsto \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle) (|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}} \quad (3.42)$$

Dimostrazione. Per definizione (3.32) la trasformata di Fourier quantistica agisce sugli elementi della base computazionale come:

$$\begin{aligned}
|j\rangle &\mapsto \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j k / 2^n} |k\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \sum_{k_2=0}^1 \dots \sum_{k_n=0}^1 e^{2\pi i j (\sum_{l=1}^n k_l 2^{-l})} |k_1 k_2 \dots k_n\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \sum_{k_2=0}^1 \dots \sum_{k_n=0}^1 \bigotimes_{l=1}^n e^{2\pi i j k_l 2^{-l}} |k_l\rangle \\
&= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[\sum_{k_l=0}^1 e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\
&= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \\
&= \frac{(|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}
\end{aligned}$$

■

Per arrivare alla costruzione di un circuito quantistico per la QFT si definisce il gate R_k :

$$R_k := \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{bmatrix}, \quad (3.43)$$

il quale, insieme con il gate di Hadamard è alla base del circuito rappresentato in figura 3.6, che è il cuore dell'implementazione della *trasformata di Fourier quantistica*.

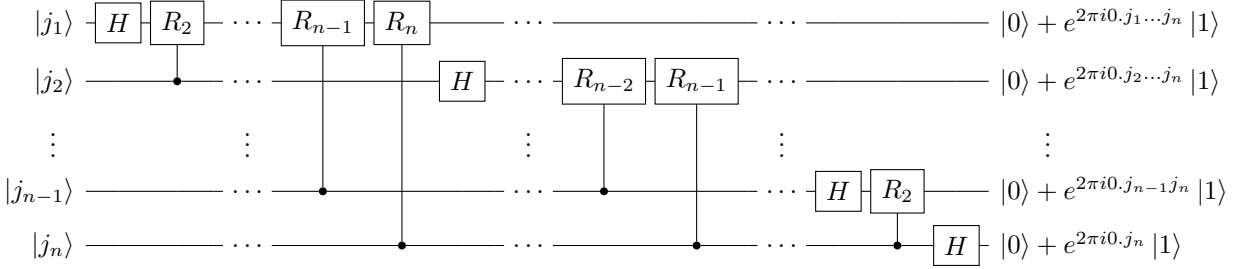


Figura 3.6: Circuito che implementa la trasformata di Fourier

Per comprendere come opera il circuito appena rappresentato lo seguiremo in dettaglio, a partire dallo stato iniziale $|j_1 j_2 \dots j_n\rangle$. Applicando il gate di Hadamard al primo qubit si ottiene lo stato

$$\frac{1}{2^{1/2}} (|0\rangle + e^{2\pi i 0 \cdot j_1} |1\rangle) |j_2 \dots j_n\rangle, \quad (3.44)$$

in quanto $e^{2\pi i 0 \cdot j_1} = -1$ se $j_1 = 1$, mentre vale $+1$ se $j_1 = 0$. In seguito all'azione del gate R_2 -controllato si arriva allo stato

$$\frac{1}{2^{1/2}} (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2} |1\rangle) |j_2 \dots j_n\rangle \quad (3.45)$$

e, continuando ad applicare i gate R_k , con $k \in \{3, \dots, n\}$, al primo qubit si giunge a

$$\frac{1}{2^{1/2}} (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle) |j_2 \dots j_n\rangle. \quad (3.46)$$

Analogamente si opera con un gate Hadamard sul secondo qubit, ottenendo

$$\frac{1}{2^{2/2}} (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot j_2} |1\rangle) |j_3 \dots j_n\rangle \quad (3.47)$$

che, in seguito all'azione dei gate R_k -controllati diventa

$$\frac{1}{2^{2/2}} (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot j_2 \dots j_n} |1\rangle) |j_3 \dots j_n\rangle. \quad (3.48)$$

Operando in modo analogo per i rimanenti qubit si ottiene lo stato finale

$$\frac{(|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot j_2 \dots j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle)}{2^{n/2}}. \quad (3.49)$$

Questa non è ancora un'implementazione della QFT. Operando con opportuni scambi di qubit, di cui il circuito (1.3) mostra una possibile realizzazione, si può ottenere un circuito il cui stato finale è

$$\frac{(|0\rangle + e^{2\pi i 0 \cdot j_n} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot j_{n-1} j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}. \quad (3.50)$$

che, come definito in (3.42), è l'immagine cercata. Si ha un circuito che implementa la *trasformata di Fourier quantistica*.

È interessante notare che, per k molto grandi, i gate R_j , con j piccolo, operano con un piccolo cambiamento di fase. Fisicamente risultano molto difficili da realizzare in modo accurato, per cui sono stati studiati circuiti quantistici che implementino la trasformazione in modo approssimato, evitando i gate incriminati.

Costo computazionale

Il circuito che implementa la *trasformata di Fourier quantistica*, basato su quello rappresentato in (3.6), richiede il seguente numero di gate per essere implementato:

1. In (3.6) sono utilizzati 1 gate Hadamard ed $n - 1$ gate R_k per il primo qubit, $n - 1$ gate per il secondo qubit e così via fino all' n -esimo qubit con 1 gate, per un totale di $n + (n - 1) + \dots + 1 = \frac{n(n+1)}{2}$ gate.
2. Gli scambi finali sono al più $n/2$, ciascuno dei quali è implementato da 3 gate CNOT.

Ovvero questo circuito richiede di $\Theta(n^2)$ gate ed è, dunque, l'implementazione di un algoritmo $\Theta(n^2)$ per la *trasformata di Fourier quantistica*.

In contrasto i migliori algoritmi classici per la computazione della trasformata di Fourier discreta (come, per esempio, FFT) su 2^n elementi richiedono $\Theta(n 2^n)$ gate per

la propria implementazione, ovvero esponenzialmente più gate della QFT. Superficialmente questo dato sembra miracoloso, vista l'importanza della trasformata di Fourier discreta nelle applicazioni pratiche. Purtroppo l'algoritmo quantistico non può sostituire direttamente l'algoritmo classico, in quanto non è possibile ottenere direttamente i coefficienti delle immagini, se non perturbando il sistema. Per sfruttare, quindi, le potenzialità di questo algoritmo bisogna agire in modo più sottile.

3.2.1 Stima di fase

La trasformata di Fourier è alla base dell'algoritmo per la risoluzione del seguente problema: dato un operatore unitario U su \mathbb{C}^m e un suo autovettore $|u\rangle$ a cui è associato l'autovalore $e^{2\pi i\varphi}$, individuare φ , la fase di $|u\rangle$. Un algoritmo che risolve questo problema è detto algoritmo di *stima di fase*. Nell'esecuzione dell'algoritmo saranno chiamati varie volte dei gate U^{2^j} -controllati, ovvero i gate controllati che eseguono 2^j volte consecutive l'operatore U . Ai fini dell'algoritmo di *stima di fase* non è importante la realizzazione del gate U , che sarà considerato una *scatola nera* o un *oracolo* e non influenzerà il conto per la complessità computazionale dell'algoritmo.

Circuito quantistico

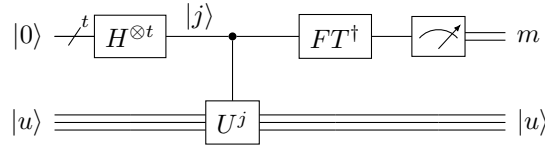


Figura 3.7: Circuito che implementa l'algoritmo di stima di fase.

L'algoritmo che verrà descritto in questa sezione fa uso di 2 registri: un primo, di t qubit, inizializzato allo stato $|0\rangle$, dove verrà salvata la stima della fase φ , e un secondo, inizializzato nello stato $|u\rangle$, abbastanza capiente da rappresentare l'autovettore di U . La scelta di t , lunghezza del primo registro, si mostrerà dipendere dall'accuratezza con cui si vuole ottenere la stima della fase φ e dalla probabilità di successo dell'algoritmo.

L'algoritmo di *stima di fase* è rappresentato nel circuito in figura 3.7, la cui spiegazione nel dettaglio è a seguire: Innanzitutto si opera con un gate di Hadamard su tutti i t qubit del primo registro, che porta allo stato

$$|j\rangle \otimes |u\rangle = \frac{(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) \dots (|0\rangle + |1\rangle)}{2^{t/2}} \otimes |u\rangle. \quad (3.51)$$

A questo punto, come descritto più in dettaglio nel circuito rappresentato in figura 3.8, si opera con il gate controllato U^j , ove j è lo stato del qubit nel primo registro. Dal diagramma si desume che il gate controllato U^j è implementato da successivi gate controllati U^{2^k} , i quali agiscono sempre sul qubit nello stato $|u\rangle$. In particolare il gate controllato U agisce sul 2-qubit nello stato $(|0\rangle + |1\rangle)|u\rangle$ nel seguente modo

$$CU(|0\rangle + |1\rangle)|u\rangle = |0\rangle|u\rangle + |1\rangle(e^{2\pi i\varphi}|u\rangle) = (|0\rangle + e^{2\pi i\varphi}|1\rangle)|u\rangle \quad (3.52)$$

Analogamente agiscono i gate U^{2^j} controllati, per cui, in seguito al circuito 3.8, il primo registro si trova nello stato

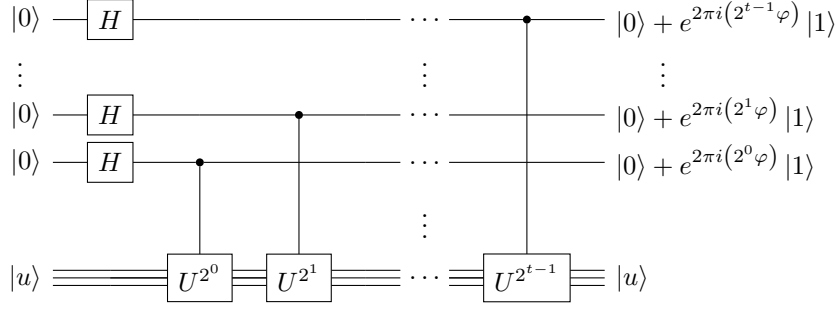


Figura 3.8: Prima parte del circuito 3.7 in dettaglio

$$\frac{(|0\rangle + e^{2\pi i 2^{t-1} \varphi} |1\rangle) \cdots (|0\rangle + e^{2\pi i 2^0 \varphi} |1\rangle)}{2^{t/2}} = \frac{1}{2^{t/2}} \sum_{k=0}^{2^t-1} e^{2\pi i k \varphi} |k\rangle, \quad (3.53)$$

mentre il secondo per tutta la computazione è restato, e resterà, nello stato $|u\rangle$.

La seconda parte della computazione consiste nell'applicare al primo registro la trasformata di Fourier inversa, per la quale un circuito può essere l'inverso del circuito 3.6 per la trasformata di Fourier, descritto nella sezione precedente. Successivamente si misura tale registro nella base computazionale.

Per comprendere il funzionamento dell'algoritmo è utile supporre, inizialmente, che φ sia un numero razionale, la cui scrittura $0.\varphi_1\varphi_2\ldots\varphi_m$ abbia $m \leq t$ cifre e, quindi, sia rappresentabile nel primo registro. Chiamati $\varphi_{m+1} = \cdots = \varphi_t = 0$, si ottiene che lo stato (3.53) può essere riscritto come

$$\frac{(|0\rangle + e^{2\pi i 0.\varphi_t} |1\rangle) (|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t} |1\rangle) \cdots (|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2\ldots\varphi_t} |1\rangle)}{2^{t/2}}. \quad (3.54)$$

Confrontato con la rappresentazione prodotto della trasformata di Fourier (3.42), rende evidente il fatto che, in seguito alla trasformata di Fourier inversa, il primo registro si troverà nello stato $|\varphi_1\varphi_2\ldots\varphi_t\rangle$. Da questo stato la misurazione nella base computazionale restituirà sempre il valore esatto di φ .

Accuratezza e probabilità di successo

Nel caso generale il processo è più complesso e non restituisce sempre il risultato richiesto, per cui richiede un'analisi più attenta. Si chiami $0 \leq b \leq 2^t - 1$ il massimo numero tale che $b/2^t = 0.b_1b_2\ldots b_t \leq \varphi$, ovvero tale che $b/2^t$ è la miglior approssimazione per difetto di φ , lunga t bit. Si definisce, inoltre, $\delta := \varphi - b/2^t$, il quale, chiaramente, soddisfa $0 \leq \delta \leq 2^{-t}$. Lo stato (3.53), in seguito alla trasformata di Fourier inversa, diventa

$$\frac{1}{2^t} \sum_{k,l=0}^{2^t-1} e^{-\frac{2\pi i kl}{2^t}} e^{2\pi i \varphi k} |l\rangle, \quad (3.55)$$

i cui coefficienti α_l dei termini $|(b+l) \bmod 2^t\rangle$ sono

$$\alpha_l = \frac{1}{2^t} \sum_{k=0}^{2^t-1} \left(e^{2\pi i(\varphi-(b+l)/2^t)} \right)^k. \quad (3.56)$$

Prendendo la somma parziale della serie geometrica possono essere riscritti come

$$\alpha_l = \frac{1}{2^t} \left(\frac{1 - e^{2\pi i 2^t(\varphi-(b+l)/2^t)}}{1 - e^{2\pi i(\varphi-(b+l)/2^t)}} \right) = \frac{1}{2^t} \left(\frac{1 - e^{2\pi i(2^t\delta-l)}}{1 - e^{2\pi i(\delta-l/2^t)}} \right). \quad (3.57)$$

Per studiare l'errore commesso dal metodo si introduce un valore e , intero positivo, che rappresenta la tolleranza sul calcolo di b che si è disposti ad accettare. Tale valore e rappresenta un errore di $e/2^t$ nel calcolo di $b/2^t$, che diventa di $e/2^t + \delta \leq (e+1)/2^t$, in quanto $0 \leq \delta \leq 2^{-t}$, per φ . Vogliamo studiare, e limitare, la probabilità di ottenere, in seguito alla misurazione, un valore m che non rispetti questa tolleranza, ovvero per cui $|m-b| > e$. La probabilità di osservare un tale m è

$$\mathbb{P}(|m-b| > e) = \sum_{-2^{t-1} < l \leq -(e+1)} |\alpha_l|^2 + \sum_{e+1 \leq l \leq 2^{t-1}} |\alpha_l|^2 \quad (3.58)$$

Inoltre per ogni θ reale $1 - e^{i\theta} \leq 2$, da cui

$$|\alpha_l| \leq \frac{2}{2^t |1 - e^{2\pi i(\delta-l/2^t)}|}. \quad (3.59)$$

Si dimostra, inoltre, che, se $-\pi \leq \theta \leq \pi$, allora $|1 - e^{i\theta}| \geq 2|\theta|/\pi$. In particolare, se $-2^{t-1} < l \leq 2^{t-1}$, si ha $-\pi \leq 2\pi(\delta - l/2^t) \leq \pi$, da cui

$$|\alpha_l| \leq \frac{2}{2^{t+1}(\delta - l/2^t)}. \quad (3.60)$$

Sostituendo (3.60) in (3.58), si ottiene

$$\mathbb{P}(|m-b| > e) \leq \frac{1}{4} \left[\sum_{l=-2^{t-1}+1}^{-(e+1)} \frac{1}{(l-2^t\delta)^2} + \sum_{l=e+1}^{2^{t-1}} \frac{1}{(l-2^t\delta)^2} \right], \quad (3.61)$$

che, in quanto $0 \leq 2^t\delta \leq 1$, diventa

$$\mathbb{P}(|m-b| > e) \leq \frac{1}{4} \left[\sum_{l=-2^{t-1}+1}^{-(e+1)} \frac{1}{l^2} + \sum_{l=e+1}^{2^{t-1}} \frac{1}{(l-1)^2} \right] \quad (3.62)$$

$$\leq \frac{1}{2} \sum_{l=e}^{2^{t-1}-1} \frac{1}{l^2} \quad (3.63)$$

$$\leq \int_{e-1}^{2^{t-1}-1} \frac{1}{l^2} dl \quad (3.64)$$

$$\leq \frac{1}{2(e-1)}. \quad (3.65)$$

Per ottenere un'approssimazione di φ accurata a 2^{-n} si sceglie $e = 2^{t-n} - 1$, per quanto visto quando si è introdotta la tolleranza e . Inoltre si può scegliere t arbitrariamente grande per ottenere con grande probabilità una stima che soddisfa tale requisito di accuratezza. In particolare da (3.65) si ricava che la probabilità di ottenere un risultato entro i requisiti è $1 - \frac{1}{2(2^p-2)}$, posto $t = n + p$. Da questo risultato si desume che, per ottenere un avere corretto con probabilità $1 - \epsilon$, si deve scegliere

$$t = n + \left\lceil \log_2 \left(2 + \frac{1}{2\epsilon} \right) \right\rceil. \quad (3.66)$$

Un ultimo commento sull'algoritmo: nella descrizione precedente si è richiesto di preparare il secondo registro nello stato $|u\rangle$, autovettore dell'operatore U . Non sempre tale preparazione è possibile, ma ciò non preclude l'utilizzo di questo algoritmo. È possibile, infatti, preparare una sovrapposizione di autovettori $|\psi\rangle = \sum_u c_u |u\rangle$. Applicando l'algoritmo ai nuovi registri $|0\rangle |\psi\rangle$, per linearità delle operazioni coinvolte, restituirà come risultato $\sum_u c_u |\widetilde{\varphi}_u\rangle |u\rangle$, dove, con grande probabilità, $\widetilde{\varphi}_u$ è una buona approssimazione di φ_u . Misurando il primo registro, quindi, si ottiene con probabilità $|c_u|^2$ una stima della fase di $|u\rangle$, autovettore di U . Si è riusciti lo stesso a ottenere il risultato richiesto, al costo di introdurre una maggiore aleatorietà nell'output del programma.

Algoritmo: Stima di fase quantistica.

Input: (1) una scatola nera per eseguire il gate U^j controllato, dove j è un intero arbitrario, (2) un autovettore $|u\rangle$ di U , con autovalore $e^{2\pi i \varphi_u}$ (o una combinazione lineare di tali autovettori), (3) $t = n + \lceil \log_2 (2 + \frac{1}{2\epsilon}) \rceil$ qubit inizializzati nello stato $|0\rangle$.

Output: $\widetilde{\varphi}_u$, un'approssimazione di n bit di φ_u .

Runtime: $O(t^2)$ operazioni (dovute alla trasformata di Fourier inversa) e una chiamata alla scatola nera per l'esecuzione del gate U^j controllato. Ritorna un output corretto con probabilità almeno $1 - \epsilon$.

Procedura:

1. $|0\rangle |u\rangle$ stato iniziale
2. $\xrightarrow{H^{\otimes t}} \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |u\rangle$ sovrapposizione iniziale
3. $\xrightarrow{CU^j} \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle U^j |u\rangle$ scatola nera
 $= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i j \varphi_u} |j\rangle |u\rangle$ risultato scatola nera
4. $\xrightarrow{FT^\dagger} |\widetilde{\varphi}_u\rangle |u\rangle$ risultato trasformata Fourier inversa
5. $\mapsto \widetilde{\varphi}_u$ misura primo registro

3.2.2 Individuazione dell'ordine

Le prossime due sezioni sono strettamente correlate, tanto da risultare problemi equivalenti. Per una questione di chiarezza nell'esposizione si è preferito esporre i due argomenti in modo distinto, in particolare prima il problema di individuazione dell'ordine, per poi spiegarne l'applicazione per la fattorizzazione dei numeri interi.

La scelta di trattare questi algoritmi, inoltre, è dovuta al fatto che entrambi godono di un forte interesse sia teorico che applicativo. Dal punto di vista pratico possono essere usati per attaccare, in modo efficiente, la crittografia a chiave pubblica RSA, come descritto in appendice B. Da un punto di vista più teorico, invece, sono una prima evidenza della possibilità che i computer quantistici siano effettivamente più potenti di quelli classici. Potrebbero, quindi, essere un controesempio alla tesi forte di Church-Turing, di grande importanza nell'impostazione della teoria della complessità computazionale.

Il problema da risolvere è il seguente. Sia $\mathbb{Z}/n\mathbb{Z}$ l'anello delle classi di resto modulo n – l'anello quoziente di \mathbb{Z} sull'ideale $n\mathbb{Z}$. Si indichi con $(\mathbb{Z}/n\mathbb{Z})^*$ l'insieme degli elementi invertibili dell'anello $\mathbb{Z}/n\mathbb{Z}$, considerato con la naturale struttura di gruppo moltiplicativo. Il problema di ricerca dell'ordine consiste, dato un arbitrario elemento x del gruppo $(\mathbb{Z}/n\mathbb{Z})^*$, nel trovarne l'ordine. In altre parole si chiede di trovare il minimo $r \in \mathbb{N}$ tale che $x^r = 1 \pmod{N}$.

Circuito quantistico

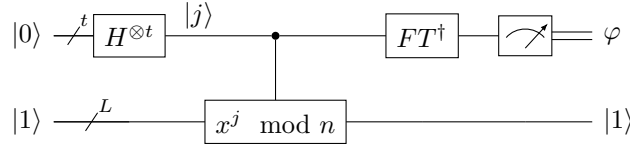


Figura 3.9: Circuito che implementa l'algoritmo di individuazione dell'ordine, si noti l'estrema somiglianza al circuito 3.7 per la stima di fase.

Per illustrare l'algoritmo di ricerca dell'ordine è necessario introdurre il seguente operatore unitario $U_{x,N}$:

$$U_{x,N} |y\rangle := |xy \pmod{N}\rangle, \quad (3.67)$$

dove $y \in \{0,1\}^L$, con $L := \lceil \log_2(N) \rceil$, ovvero il minimo $L \in \mathbb{N}$ t.c. $N \leq 2^L$. Si assume, inoltre, nella definizione di $U_{x,N}$ che, se $N \leq y \leq 2^L - 1$, allora $xy \pmod{N}$ sia ancora y , ovvero $U_{x,N}$ agisce in modo non banale solo sugli $0 \leq y \leq N$. Definito questo operatore, e mostrato che è unitario, l'algoritmo di individuazione dell'ordine si ridurrà ad una stima di fase per l'operatore $U_{x,N}$.

Innanzitutto si dimostra che è un operatore unitario:

Lemma 3. $U_{x,N} : |y\rangle \mapsto |xy \pmod{N}\rangle$ è un operatore unitario

Dimostrazione. x è un elemento del gruppo degli invertibili di $\mathbb{Z}/n\mathbb{Z}$, quindi la sua azione per moltiplicazione a sinistra su $(\mathbb{Z}/n\mathbb{Z})^*$ può essere vista, per il teorema di Cayley, come una permutazione σ sull'insieme finito $(\mathbb{Z}/n\mathbb{Z})^*$. Tale permutazione può essere rappresentata sotto forma di matrice come

$$X_{ij} := \delta_{\sigma(j),j}, \quad (3.68)$$

dove δ è la delta di Kronecker. Questa matrice è chiaramente a coefficienti reali, per cui la sua aggiunta corrisponderà con la trasposta. In particolare σ^{-1} , l'inversa di σ sarà la permutazione per cui $\sigma^{-1}(\sigma(j)) = j$, ovvero $\sigma^{-1} : \sigma(j) \mapsto j$. Ne segue che la rappresentazione matriciale di σ^{-1} sarà

$$Y_{i,j} := \delta_{j,\sigma(j)}. \quad (3.69)$$

È chiaro che Y è la trasposta di X , ovvero $X = Y$ e di conseguenza $U_{x,N}$ è unitaria. ■

Per riuscire ad applicare l'algoritmo di ricerca di fase, inoltre, è necessario individuare degli autovettori. Un esempio sono, al variare di $0 \leq s \leq r-1$, gli stati definiti come

$$|u_s\rangle := \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(-\frac{2\pi i}{r} sk\right) |x^k \bmod N\rangle. \quad (3.70)$$

Su di essi, infatti, $U_{x,N}$ agisce, per linearità, nel seguente modo

$$U_{x,N} |u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(-\frac{2\pi i}{r} sk\right) |x^{k+1} \bmod N\rangle \quad (3.71)$$

$$= \exp\left(\frac{2\pi i}{r} s\right) |u_s\rangle. \quad (3.72)$$

Da questi autovettori si può ottenere, con grande probabilità e accuratezza, una stima di $\exp\left(\frac{2\pi i s}{r}\right)$, dalla quale, con buona probabilità – se s ed r sono coprimi – si può ricavare r .

Purtroppo per preparare lo stato $|u_s\rangle$, per un qualsiasi s , è necessario conoscere l'ordine r di x , ovvero il risultato cercato. Per ovviare questo problema si sfrutta il risultato ottenuto in (3.39) riguardo alla somma delle radici R -esime dell'unità. Esso, infatti, garantisce che $\forall k \in \{1, \dots, r-1\}$

$$\sum_{s=0}^{r-1} \exp\left(-\frac{2\pi i}{r} sk\right) = 0. \quad (3.73)$$

Mentre, per $k = 0$,

$$\sum_{s=0}^{r-1} \exp\left(-\frac{2\pi i}{r} s0\right) = r. \quad (3.74)$$

Queste osservazioni comportano che, sommando su s in (3.70), rimane solo la somma con $k = 0$, ovvero:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = \frac{1}{r} \sum_{s=0}^{r-1} \exp\left(-\frac{2\pi i}{r} s0\right) |1\rangle = |1\rangle. \quad (3.75)$$

Da tutto ciò si desume che, applicando l'algoritmo di stima di fase per $U_{x,N}$, usando $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ qubit nel primo registro e preparando il secondo registro nello stato $|1\rangle$ – di facile preparazione – si otterrà una stima φ della fase s/r , accurata a $2L + 1$ bit, dove s è scelto uniformemente in $0, \dots, r-1$.

A questo punto della computazione termina la fase prettamente quantistica e inizia un processo di elaborazione di φ che sfrutta algoritmi classici e può, quindi, essere

svolta su un modello di computazione classico. Per questo motivo il circuito in figura 3.9 termina con la misurazione di φ e non rappresenta le prossime operazioni.

In particolare, fissato s , si è a conoscenza del numero razionale φ , stima di s/r accurata a $2L + 1$ bit. Per ottenere r da φ si sfruttano l'algoritmo delle frazioni continue, che fornisce approssimazioni sempre più accurate di φ , descritto a seguire, e il teorema 3.2.1, la cui dimostrazione, per non appesantire troppo questa spiegazione, è stata relegata all'appendice A.

Algoritmo: Espansione in frazione continua.

Ogni numero razionale positivo può essere rappresentato come frazione continua finita “semplice”, ovvero con un'espressione come

$$[a_0; a_1, a_2, \dots, a_n] := a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}}, \quad (3.76)$$

in cui $a_0 \in \mathbb{N}$ e $a_1, \dots, a_n \in \mathbb{N}^+ := \{1, 2, \dots\}$. In questa notazione si chiama convergente j -esimo, con $0 \leq j \leq n$, l'espansione $[a_0; a_1, a_2, \dots, a_j]$, corrispondente a una frazione p_j/q_j . Questa espansione si può trovare operando con il seguente

Algoritmo: Sia $x = p_0/p_1$ un numero razionale. Si ricavano, iterativamente, a_j operando nel seguente modo. $a_0 = \lfloor x \rfloor$ è la parte intera di x , per cui

$$x = a_0 + \frac{p_2}{p_1} = a_0 + \frac{1}{\frac{p_1}{p_2}}, \quad (3.77)$$

in cui, nell'ultimo passaggio, si è invertita la frazione. In particolare, in quanto $a_0 = \lfloor x \rfloor$ segue che $p_2/p_1 < 1$, ovvero $p_2 < p_1$. Chiamato $x_1 := p_1/p_2$ si ripete su x_1 il processo appena svolto, ad ottenere a_1 e $x_2 := p_2/p_3$, con $p_3 < p_2$. Iterando si calcolano a_n, x_{n+1} e p_{n+2} a partire da x_n , costruendo un'espansione in frazione continua per x . Tale algoritmo si arresta quando $x_n = \lfloor x_n \rfloor$ è intero, ovvero quando $p_{n+1} = 0$. In particolare, per ogni x razionale, p_j è una successione a coefficienti in \mathbb{N}^+ e strettamente decrescente. Ne consegue che, in un numero finito di passi $p_j = 0$, ovvero l'algoritmo si ferma.

In appendice A, nel teorema A.0.1, si definisce un algoritmo per il calcolo dei convergenti. Questo algoritmo verrà usato, ma è importante citare alcune sue proprietà prima di usarlo. Come mostrato nel lemma 5, dimostrato in appendice, le frazioni costruite da tale algoritmo sono ridotte ai minimi termini. Inoltre, come conseguenza dello stesso teorema A.0.1, la successione dei denominatori delle frazioni corrispondenti ai convergenti è strettamente crescente.

Infine, prima di continuare, è necessario citare il seguente risultato:

Teorema 3.2.1. *Siano φ ed s/r razionali tali che*

$$\left| \frac{s}{r} - \varphi \right| \leq \frac{1}{2r^2} \quad (3.78)$$

Allora s/r è un convergente della frazione continua di φ

Nel caso analizzato φ è un'approssimazione di s/r precisa a $2L + 1$ bit, per cui $|s/r - \varphi| \leq 2^{-2L-1} \leq \frac{1}{2r^2}$, in quanto $r < N \leq 2^L$. Si può, quindi, applicare il teorema. In quanto il valore r è ignoto, mentre N è noto e vale la più forte disuguaglianza $|s/r - \varphi| \leq \frac{1}{2N^2}$ è utile concentrarsi su questa. In quanto $2N^2 > 2r$ si sa che in $(\varphi - \frac{1}{4N^2}, \varphi + \frac{1}{4N^2})$ esiste un solo numero del tipo a/r , ovvero s/r . Si consideri, ora, $m < r$, allora per ogni frazione l/m si avrà, necessariamente

$$\left| \frac{l}{m} - \frac{s}{r} \right| = \left| \frac{lr - sm}{rm} \right| > \frac{1}{rm} > \frac{1}{N^2}. \quad (3.79)$$

Da cui l/m non può distare meno di $1/2N^2$ da φ . Ovvero, per ogni $m < r$, non esiste alcuna frazione con m a denominatore nell'intervallo $(\varphi - \frac{1}{4N^2}, \varphi + \frac{1}{4N^2})$. La frazione con il più piccolo denominatore che vi appartiene è, quindi, s/r . Nell'esecuzione dell'algoritmo per i convergenti della frazione continua, dunque, s/r si può individuare come la prima frazione l/m che soddisfa la condizione

$$\left| \frac{l}{m} - \varphi \right| < \frac{1}{2N^2}. \quad (3.80)$$

È, però, da notare che l'algoritmo delle frazioni continue genera denominatori e numeratori coprimi tra loro. Nel caso in cui, casualmente, $\text{MCD}(s, r) \neq 1$, allora, il denominatore ottenuto non sarà r , ma $r' \mid r$. Nella prossima sezione si analizzerà la probabilità di successo dell'algoritmo e verranno proposte delle strategie operative per mitigare il problema dovuto alla non coprimialità di s ed r .

Probabilità di successo

Per finalizzare l'algoritmo è interessante studiare quali sono le cause di insuccesso di tale algoritmo e quali strategie possono essere implementate per mitigarle. Si introdurranno queste soluzioni prima di passare allo studio del costo computazionale dell'algoritmo, per poter dare un risultato più accurato.

Il primo motivo per cui questo algoritmo potrebbe fallire è che, con probabilità minore di ϵ l'algoritmo di ricerca dell'ordine potrebbe restituire una stima non sufficientemente accurata di s/r . Questa probabilità di errore può essere resa piccola a piacere con un piccolo incremento delle dimensioni del circuito – aumentando t . Il secondo, e più importante, motivo di errore è dato, come accennato a fine della sezione precedente, dalla possibilità che s , scelto uniformemente tra 0 ed $r - 1$, abbia fattori comuni con r . In tal caso il risultato dell'algoritmo di espansione in frazione continua restituirà come output $r' \mid r$, ma non r . Si espongono, ora, 3 possibili strategie per mitigare il problema.

La prima consiste, semplicemente, nel notare che la probabilità che s ed r siano coprimi tra loro è, in realtà abbastanza elevata. Vale, infatti il seguente risultato, la cui dimostrazione è fuori dallo scopo di questa trattazione:

Teorema 3.2.2. *Sia $\pi : \mathbb{N} \rightarrow \mathbb{N}$ la mappa che associa ad $n \in \mathbb{N}$ il numero di numeri primi minori o uguali di n . Vale la seguente disuguaglianza:*

$$\pi(2n) \geq \frac{n}{\log(2n)}. \quad (3.81)$$

Segue che almeno $r/r \log(r)$ dei naturali minori di r sono primi. La probabilità che s sia primo – e quindi coprimo con r – è di $1/2 \log(r) > 1/r \log(N)$, da cui, ripetendo l'algoritmo $2 \log(N)$ volte si ha una grande probabilità di successo.

Un secondo metodo, più interessante, richiede di notare che, se $s \neq 0$, $r' \neq r$ è un fattore proprio di r . L'ordine di $x' := x^{r'} \bmod N$ sarà, quindi, r/r' e si potrà operare con lo stesso algoritmo su x' . È possibile che anche in questo caso s' ed r' non siano coprimi, richiedendo di procedere ulteriormente. Ad ogni passo, però, l'ordine $r^{(n)}$ di $x^{(n)}$ viene diviso per un fattore strettamente maggiore di 2. È necessario, allora, operare al più $\log_2(r) = O(L)$ volte per trovare l'ordine dell'ultima iterazione. A quel punto, moltiplicando tra loro tutti i risultati ottenuti, si ottiene l'ordine di x .

L'ultimo metodo, invece, richiede solamente un numero finito di tentativi, a differenza dei primi due. Operando 2 volte si ottengono i risultati r_1, s_1 ed r_2, s_2 . Da questi, se s_1 ed s_2 non hanno fattori comuni, si può ricavare r semplicemente prendendo il minimo comune multiplo di r_1 ed r_2 . È necessario, quindi, stimare la probabilità che s_1 ed s_2 non abbiano fattori comuni. Per fare ciò si calcola la probabilità che ne abbiano, ovvero la probabilità dell'evento complementare. Questa può essere espressa come la probabilità che esista almeno un primo p che divide entrambi. La probabilità da stimare è, quindi

$$P := 1 - \mathbb{P}(\exists p \text{ primo} : p \mid s_1 \text{ e } p \mid s_2). \quad (3.82)$$

Per subadditività della probabilità, inoltre, si ottiene la maggiorazione

$$\mathbb{P}(\exists p \text{ primo} : p \mid s_1 \text{ e } p \mid s_2) \leq \sum_p \mathbb{P}(p \mid s_1 \text{ e } p \mid s_2), \quad (3.83)$$

dove la somma è su tutti i numeri primi p . In particolare i numeri s_1 ed s_2 sono scelti in modo indipendente l'uno dall'altro, per cui vale anche

$$\mathbb{P}(p \mid s_1 \text{ e } p \mid s_2) = \mathbb{P}(p \mid s_1) \mathbb{P}(p \mid s_2). \quad (3.84)$$

Si deve, inoltre, notare che s_1 – ma la stessa osservazione vale anche per s_2 – è scelto uniformemente in $0, \dots, r-1$. La probabilità che sia divisibile per p è, quindi, data da

$$\frac{|p\mathbb{N} \cap \{0, \dots, r-1\}|}{|\{0, \dots, r-1\}|} = \frac{\lfloor r/p \rfloor + 1}{r} \leq \frac{r/p + 1}{r} = \frac{1}{p} + \frac{1}{r} < \frac{1}{p}. \quad (3.85)$$

Sostituendo queste maggiorazioni in (3.82) si ottiene la stima

$$P = 1 - \mathbb{P}(\exists p \text{ primo} : p \mid s_1 \text{ e } p \mid s_2) > 1 - \sum_p \frac{1}{p^2}, \quad (3.86)$$

dove, come prima, la somma è sui numeri primi p . Rimane, ora, solamente da stimare

$$\sum_p \frac{1}{p^2} < \sum_{n=2}^{\infty} \frac{1}{n^2}. \quad (3.87)$$

A tal fine si sfrutta il seguente risultato

$$\int_x^{x+1} \frac{1}{y^2} dy \geq \frac{2}{3x^2} \quad \forall x \geq 2. \quad (3.88)$$

Ciò è vero in quanto $\int_x^{x+1} 1/y^2 \, dy = 1/(x+x^2)$ e, inoltre, vale la seguente

$$\frac{1}{x+x^2} \geq \frac{2}{3x^2} \iff 3x^2 \geq 2x^2 + 2 \quad (3.89)$$

$$\iff x^2 - 2x \geq 0. \quad (3.90)$$

L'ultima disuguaglianza è chiaramente verificata per ogni $x \geq 2$. Sfruttando il risultato di (3.88) in (3.87) si arriva a

$$\sum_{n=2}^{\infty} \frac{1}{n^2} \leq \frac{3}{2} \int_2^{\infty} \frac{1}{y^2} \, dy = \frac{3}{4}. \quad (3.91)$$

Sostituito in (3.86) si ottiene la stima cercata, ovvero:

$$P > 1 - \sum_p \frac{1}{p^2} > \frac{1}{4}. \quad (3.92)$$

Ripetendo l'algoritmo 4 volte si ha, quindi, una grande probabilità di ottenere il risultato corretto. La cosa importante da notare di questa strategia è che la probabilità di ottenere il risultato giusto non dipende da N . Per N grandi risulta, quindi, la scelta migliore, richiedendo solo un numero costante di ripetizioni.

Costo computazionale

Per eseguire il circuito 3.9, ed analizzarne il costo computazionale, è necessario implementare un circuito che operi la seguente trasformazione

$$|z\rangle |y\rangle \mapsto |z\rangle |x^z y \mod N\rangle \quad (3.93)$$

A tal fine si userà, ed analizzerà, l'algoritmo di *esponenziazione modulare*. Non verrà, però, trattato con estrema profondità, per la quale si rimanda a [Sho97].

Algoritmo: esponenziazione modulare.

L'algoritmo di *esponenziazione modulare* opera nel seguente modo su una coppia di registri, il primo di lunghezza $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ e il secondo di lunghezza $L = \lceil \log_2(N) \rceil$:

$$|z\rangle |y\rangle \mapsto |z\rangle |x^z y \mod N\rangle. \quad (3.94)$$

Espresso z in base binaria $z = z_t \dots z_1$, dove z_1, \dots, z_t sono le cifre binarie di z , l'azione dell'algoritmo è analoga a:

$$|z\rangle |y\rangle \mapsto |z\rangle \left| x^{z_t 2^{t-1}} \dots x^{z_1 2^0} y \mod N \right\rangle \quad (3.95)$$

$$= |z\rangle U^{z_t 2^{t-1}} \dots U^{z_1 2^0} |y\rangle. \quad (3.96)$$

Nell'ultima riga si è espressa l'applicazione di una sequenza di gate $U_{x,N}^{2^j}$ controllati. Ciascuno di questi gate può essere implementato usando un registro ausiliario in cui calcolare in modo reversibile la funzione $x^z \mod N$. Si moltiplica, poi, sempre in modo reversibile il secondo registro, nello stato $|y\rangle$ per il contenuto del terzo registro. Il gate termina cancellando il contenuto del terzo registro eseguendo le operazioni usate per calcolare $x^z \mod N$ in ordine inverso. Come mostrato da [Sho97], nella

sezione 3, tale algoritmo, se implementato con l'usale algoritmo di prodotto di numeri interi (quello insegnato alle elementari) ha un costo di $O(L^2)$ gate.

L'algoritmo di esponenziazione modulare, quindi, si implementa nelle seguenti due fasi. In primo luogo si calcolano, usando la procedura appena delineata, i valori $x^{2^j} \bmod N$, per $0 \leq j \leq t-1$, elevando iterativamente al quadrato il valore precedente. Ciascun elevamento al quadrato ha un costo di $O(L^2)$ gate, da ripetere un totale di $t-1 = 2L + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil = O(L)$ di volte. Questa fase, quindi, ha un costo complessivo di $O(L^3)$.

La seconda fase, invece, richiede di calcolare, nel secondo registro,

$$x^z \bmod N = \left(x^{z_t 2^{t-1}} \bmod N \right) \cdots \left(x^{z_1 2^0} \bmod N \right), \quad (3.97)$$

ovvero di eseguite i $t-1$ gate controllati $U_{x,N}^{2^j}$. Ciascuna di queste moltiplicazioni, come mostrato prima, richiede $O(L^2)$ operazioni per essere eseguita. Anche la seconda fase ha, quindi, un costo computazionale di $O(L^3)$ gate.

Seguendo questa descrizione si può implementare un gate quantistico che opera la seguente trasformazione

$$|z\rangle |y\rangle \mapsto |z\rangle |x^z y \bmod N\rangle \quad (3.98)$$

a un costo di $O(L^3)$ gate. Questo non è il risultato (asintotico) più efficiente conosciuto, per approfondire si può consultare l'articolo precedentemente citato.

Per quanto riguarda l'algoritmo di espansione in frazione continua la richiesta computazionale è analoga. Sia, infatti, $x = p/q$ un numero razionale maggiore di 1, dove p e q sono coprimi. Siano, inoltre, a_0, \dots, a_N i primi N termini dell'espansione in frazione continua semplice di x . Prese p_n e q_n come definite nel teorema A.0.1 si ha che $p_n \geq 2p_{n-2}$ e $q_n \geq 2q_{n-2}$, come mostrato in (A.19). Segue che $p_n, q_n \geq 2^{\lfloor n/2 \rfloor}$, per cui $2^{\lfloor N/2 \rfloor} \leq q \leq p$. Da ciò si ottiene che il numero di termini dell'espansione in frazione continua è $n = O(\log_2(p))$. In particolare, se p e q sono numeri rappresentabili con L bit, allora $n = O(L)$. L'algoritmo per le frazioni continue, quindi, opera con $O(L)$ inversioni e sottrazioni, ciascuna di esse eseguibile con $O(L^2)$ operazioni. Il suo costo computazionale complessivo è, come promesso, $O(L^3)$ gate.

Gli altri due algoritmi che compaiono nel circuito 3.9 sono il gate di Hadamard, con una richiesta di $O(L)$ operazioni e la trasformata di Fourier inversa, che ne richiede $O(L^2)$.

Infine, supponendo di adottare il terzo metodo per mitigare la possibilità di errore dovuta alla non coprimalità di s ed r , si deve ripetere un numero costante di volte l'algoritmo. Combinando tutti questi risultati si ottiene che il costo computazionale complessivo è di $O(L^3)$.

Algoritmo: Individuazione dell'ordine quantistico.

Input: (1) $x, N \in \mathbb{N}$ t.c. $\text{MCD}(x, N) \neq 1$, (2) $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ qubit inizializzati nello stato $|0\rangle$, (3) L qubit inizializzati nello stato $|1\rangle$.

Output: il minimo $r \in \mathbb{N}$ tale che $x^r = 1 \bmod N$.

Runtime: $O(L^3)$ operazioni (per il calcolo dell'esponenziazione modulare). Ritorna un output corretto con probabilità $O(1)$.

Procedura:

1. $|0\rangle |1\rangle$ stato iniziale
2. $\xrightarrow{H^{\otimes t}} \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |1\rangle$ sovrapposizione iniziale
3. $\xrightarrow{U_{x,N}} \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |x^j \bmod N\rangle$ risultato $U_{x,N}$
 $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j / r} |j\rangle |u_s\rangle$
4. $\xrightarrow{FT^\dagger} \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \widetilde{|s/r\rangle} |u_s\rangle$ risultato trasformata Fourier inversa
5. $\mapsto \widetilde{s/r}$ misura primo registro
6. $\mapsto s$ algoritmo frazioni continue

3.2.3 Fattorizzazione degli interi

Nelle due sezioni a seguire saranno descritti due problemi di teoria dei numeri di grande importanza teorica. Questi sono il problema di fattorizzazione e il problema del logaritmo discreto, per la risoluzione dei quali non si dispone di un algoritmo classico efficiente. In realtà si crede che questi problemi non ammettano una risoluzione classica efficiente e, per questo motivo, sono stati implementati vari protocolli crittografici basati su essi.

Si introdurranno, però, algoritmi quantistici efficienti per la risoluzione dei problemi di fattorizzazione e di calcolo del logaritmo discreto. I motivi di interesse in questi algoritmi sono sostanzialmente due. Innanzitutto, se si dimostrasse che questi problemi, classicamente, non ammettono risoluzione efficiente, si avrebbe una conferma che la computazione quantistica rappresenta un modello più potente di quello classico: un controesempio alla tesi forte di Church-Turing. Inoltre le applicazioni pratiche, nel rendere inefficaci i sistemi crittografici citati – come mostrato più in dettaglio in appendice B – creano un forte interesse nell'utilizzo di questi algoritmi.

In questa sezione si ottiene una riduzione del problema di ricerca di un fattore primo al problema di individuazione dell'ordine, appena studiato. Ottenuto questo risultato si ricava, di conseguenza, un algoritmo efficiente per la fattorizzazione dei numeri interi, come preannunciato.

La seguente riduzione assume che il numero N di cui cercare la fattorizzazione sia dispari e non sia una potenza di un primo. Questa restrizione non è molto pesante. Il primo caso, infatti, può essere scartato dividendo ripetutamente per 2 e, in notazione binaria, riconoscere un numero pari non pone un problema. Il secondo caso, invece, richiede di operare con il seguente algoritmo classico:

1. Con un algoritmo veloce, per esempio quello descritto in [Ber98], si controlla se il numero N è una potenza perfetta, ovvero la b -esima potenza di un altro numero naturale.
2. Si controlla con un algoritmo veloce, si veda [AKS04], che tale b -esima radice di N sia un numero primo.

Per terminare la riduzione del problema di fattorizzazione alla ricerca dell'ordine sono, infine, necessari i seguenti risultati:

Teorema 3.2.3. *Sia N un numero di L bit composto, ovvero non primo. Sia x una soluzione non banale, ovvero $1 < x < N - 1$, dell'equazione $x^2 = 1 \pmod{N}$. Allora almeno uno tra $\text{MCD}(x - 1, N)$ e $\text{MCD}(x + 1, N)$ è un fattore proprio, non banale, di N . In particolare la richiesta computazionale per il suo calcolo è di $O(L^3)$ operazioni.*

Dimostrazione. $x^2 = 1 \pmod{N}$ implica che N divide $x^2 - 1 = (x + 1)(x - 1)$, ovvero N ha almeno un fattore comune con $x - 1$ o con $x + 1$. Per ipotesi $1 < x < N - 1$ comporta che $x - 1 < x + 1 < N$, ovvero il fattore comune non può essere N stesso ed è, quindi, un fattore proprio di N . Per finire la dimostrazione basta notare che $\text{MCD}(x - 1, N)$ e $\text{MCD}(x + 1, N)$ si possono calcolare operando con l'algoritmo di Euclide, con una spesa di $O(L^3)$ operazioni – come mostrato in [Knu97]. ■

Il teorema appena dimostrato afferma che, individuato un particolare elemento di $\mathbb{Z}/N\mathbb{Z}^*$, da esso si può trovare efficientemente un fattore proprio di N . Il problema si riconduce, ora, alla ricerca di tali elementi in $\mathbb{Z}/N\mathbb{Z}^*$. Per trovarli è sufficiente individuare degli elementi $y \in \mathbb{Z}/N\mathbb{Z}^*$, di ordine r pari, per cui $y^{r/2} \neq \pm 1 \pmod{N}$. Posto $x = y^{r/2}$ si ha una soluzione non banale di $x^2 = 1 \pmod{N}$. A priori potrebbe essere molto raro individuare tali elementi, fortunatamente non è il caso. L'obiettivo dei seguenti teoremi è di calcolare la probabilità che, scelto uniformemente un elemento in $(\mathbb{Z}/N\mathbb{Z}) \setminus \{0\}$, soddisfi le proprietà appena enunciate.

Lemma 4. *Siano p un primo dispari ed α un numero naturale. Sia 2^d la più grande potenza di 2 che divide $\varphi(p^\alpha)$. Allora, scelto $x \in \mathbb{Z}/p^\alpha\mathbb{Z}^*$ uniformemente, 2^d divide l'ordine di x con probabilità esattamente $1/2$.*

Dimostrazione. In quanto p è dispari $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ è pari, quindi $d \geq 1$. Si può mostrare – si veda [Jac12], Teorema 4.19, pagina 274 – che $\mathbb{Z}/p^\alpha\mathbb{Z}^*$ è un gruppo ciclico. Sia g un suo generatore e si consideri il generico elemento g^k , con $1 \leq k \leq \varphi(p^\alpha)$. Chiamato r l'ordine di g^k si considerano i due possibili casi per k . Se k è dispari, in quanto $kr = \varphi(p^\alpha)$ e $2^d \mid \varphi(p^\alpha)$, allora $2^d \mid r$. Altrimenti k è pari, quindi

$$g^{k\varphi(p^\alpha)/2} = \left(g^{\varphi(p^\alpha)}\right)^{k/2} = 1^{k/2} = 1_{\mathbb{Z}/p^\alpha\mathbb{Z}^*}. \quad (3.99)$$

Ovvero r , ordine di g^k , divide $\varphi(p^\alpha)/2$, per cui 2^d non divide r .

Ricapitolando, $\mathbb{Z}/p^\alpha\mathbb{Z}^*$ si divide in 2 insiemi di uguale cardinalità: gli elementi che possono essere scritti come g^k , con k dispari, e quelli con k pari. Per i primi 2^d divide r , per gli ultimi no. Scegliendo uniformemente x in $\mathbb{Z}/p^\alpha\mathbb{Z}^*$ si ha, quindi, esattamente probabilità $1/2$ di trovare un elemento a cui 2^d divide l'ordine. ■

Prima di calcolare esplicitamente la probabilità richiesta, si ricorda il seguente teorema. Non verrà data una dimostrazione, che si può trovare in [NC11].

Teorema 3.2.4 (cinese dei resti). *Siano m_1, \dots, m_n interi positivi, a due a due coprimi. Il sistema di equazioni*

$$\begin{cases} x = a_1 & \text{mod } m_1 \\ x = a_2 & \text{mod } m_2 \\ \vdots \\ x = a_n & \text{mod } m_n \end{cases} \quad (3.100)$$

ammette soluzioni. In particolare tutte le soluzioni di tale sistema coincidono, modulo $M := m_1 m_2 \dots m_n$

Teorema 3.2.5. Sia $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$ la fattorizzazione di un numero composito dispari. Sia x un numero scelto uniformemente in $\{1, \dots, N-1\}$ coprimo con N . Sia r l'ordine di x modulo N . Allora

$$\mathbb{P}\left(2 \mid r \text{ e } x^{r/2} \neq -1 \pmod{N}\right) \geq 1 - \frac{1}{2^{m-1}}. \quad (3.101)$$

Dimostrazione. Per dimostrare la tesi verrà mostrato l'equivalente:

$$\mathbb{P}\left(2 \nmid r \text{ o } x^{r/2} \neq -1 \pmod{N}\right) \leq \frac{1}{2^{m-1}}. \quad (3.102)$$

Per il teorema cinese dei resti scegliere x uniformemente in $\mathbb{Z}/N\mathbb{Z}^*$ è equivalente a scegliere uniformemente x_j indipendentemente in $\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z}^*$ e a richiedere che $x = x_j \pmod{p_j^{\alpha_j}}$ per ogni j . Si chiami r_j l'ordine di x_j in $\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z}^*$. Siano 2^{d_j} la più grande potenza di 2 che divide r_j e 2^d la più grande che divide r . Se r è dispari, in quanto, per ogni j , $r_j \mid r$, si ha che $d_j = 0$ per ogni j , in particolare tutti i d_j coincidono. Se, invece, $x^{r/2} = -1 \pmod{N}$, allora $x^{r/2} = -1 \pmod{p_j^{\alpha_j}}$ per ogni j . Segue che non esiste alcun j per cui $r_j \mid (r/2)$. Si sa, per costruzione, che $r_j \mid r$, per cui $d_j = d$ per ogni j , in particolare tutti i d_j coincidono. Quest'analisi mostra che vale la seguente inclusione di eventi

$$\left\{2 \nmid r \text{ o } x^{r/2} \neq -1 \pmod{N}\right\} \subseteq \{\exists d \in \mathbb{N} : d_j = d \ \forall 1 \leq j \leq m\}, \quad (3.103)$$

che, per monotonia della probabilità, comporta che

$$\mathbb{P}\left(2 \nmid r \text{ o } x^{r/2} \neq -1 \pmod{N}\right) \leq \mathbb{P}(\exists d \in \mathbb{N} : d_j = d \ \forall 1 \leq j \leq m). \quad (3.104)$$

Per il lemma 4 si sa che, fissato j , la probabilità che $d_j = d$ è al più $1/2$ – nel caso in cui d è la massima potenza di 2 che divide $\varphi(p_j^{\alpha_j})$. Si nota, inoltre, che, per l'arbitrarietà di d , si può imporre $d := d_1$ e verificare la condizione in (3.103) solo per $j \geq 2$. In quanto gli x_j sono scelti indipendentemente, infine, la probabilità si fattorizza, ovvero:

$$\mathbb{P}(d_j = d_1 \ \forall 2 \leq j \leq m) = \prod_{j=2}^m \mathbb{P}(d_j = d_1) \leq \prod_{j=2}^m \frac{1}{2} = \frac{1}{2^{m-1}}. \quad (3.105)$$

■

Operando come descritto in precedenza, quindi, si ha una grande probabilità di calcolare un fattore proprio di N numero composito dispari. La probabilità di successo non dipende direttamente da N , ma dal numero di fattori in cui si decompone, ovvero non è del tipo $O(f(N))$ per una funzione di N , ma è $O(1)$ e, in particolare, è sempre maggiore di $3/4$.

Riguardo all'efficienza è interessante notare come tutti i passaggi di questo algoritmo, a meno della ricerca dell'ordine, possano essere eseguiti efficientemente anche su un computer classico. Questa riduzione è, quindi, una riduzione alla sola ricerca dell'ordine, in quanto l'efficienza dell'algoritmo di fattorizzazione dipende principalmente dalla velocità di quest'ultimo.

È, ora, possibile dare una descrizione sintetica della riduzione:

Algoritmo: Riduzione della fattorizzazione alla ricerca dell'ordine.

Input: un numero N .

Output: un fattore non banale di N .

Runtime: $O((\log_2 N)^3)$ operazioni. Ritorna un output corretto con probabilità $O(1)$.

Procedura:

1. Se N è pari ritorna il fattore 2.
2. Se $N = a^b$ per $a \geq 1$ e $b \geq 2$ – individuato con l'algoritmo classico descritto in [Ber98] – restituisce a .
3. Sceglie uniformemente $1 \leq x \leq N - 1$. Se $\text{MCD}(x, N) > 1$ ritorna il fattore non banale $\text{MCD}(x, N)$.
4. Trova, con l'algoritmo di individuazione dell'ordine, l'ordine r di x . Se r non è pari l'algoritmo si ferma senza restituire un risultato.
5. Se r è pari e $x^{r/2} \not\equiv \pm 1 \pmod N$ calcola $\text{MCD}(x^{r/2} - 1, N)$ e $\text{MCD}(x^{r/2} + 1, N)$. Ritorna il fattore proprio di N individuato tra i numeri appena calcolati. Se $x^{r/2} \equiv \pm 1 \pmod N$ l'algoritmo si ferma senza dare un risultato.

Per concludere propriamente la fattorizzazione è necessario iterare la ricerca di fattori propri, fino a che non si ha una completa fattorizzazione. In particolare se, al punto 3. si sceglie un elemento x il cui ordine è dispari o tale che $x^{r/2} \equiv \pm 1 \pmod N$ si ripete il punto 3., scegliendo un elemento diverso. Se, invece, restituisce un fattore proprio D si ripete l'algoritmo sul fattore D e sul rapporto N/D . Se gli elementi appena descritti sono numeri primi il punto 2. lo individua e si può fermare l'iterazione.

3.2.4 Logaritmo discreto

Il problema del calcolo del logaritmo discreto è il seguente. Dato $p \in \mathbb{N}$ primo si sa che $(\mathbb{Z}/p\mathbb{Z})^*$ è un gruppo moltiplicativo ciclico. Sia a un generatore di tale gruppo e b un qualsiasi altro elemento. Il problema del logaritmo discreto chiede di individuare la più piccola potenza a^s che soddisfi l'equazione $a^s = b \pmod p$.

Per eseguire questo calcolo si descrive un algoritmo che usa tre registri. Il terzo è di lunghezza $L := \lceil \log_2 p \rceil$, ovvero la minima potenza di 2 necessaria per rappresentare tutti gli elementi di $(\mathbb{Z}/p\mathbb{Z})^*$. I primi due, invece, sono di lunghezza $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$, valore su cui dipende la probabilità di successo dell'algoritmo.

Infine, per rendere più chiara la generalizzazione di questo problema a gruppi arbitrari, si chiamerà r l'ordine di a , invece di riferirsi ad esso con il suo valore esplicito: $p - 1$.

Circuito Quantistico

Per iniziare si definisce la funzione $f(x, y) := a^x b^y = a^{x+sy} \pmod p$. A tale funzione, come descritto nella sezione 3.1.4 riguardo al parallelismo quantistico, si associa il gate quantistico U_f , che agisce come:

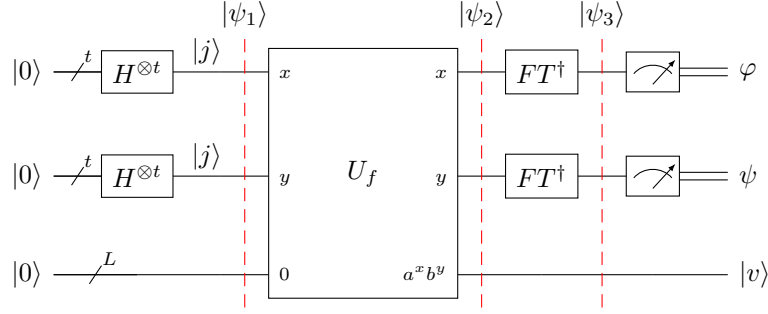


Figura 3.10: Circuito che implementa l'algoritmo di calcolo del logaritmo discreto.

$$|x\rangle |y\rangle |z\rangle \xrightarrow{U_f} |x\rangle |y\rangle |z + f(x, y) \bmod 2^L\rangle. \quad (3.106)$$

Il circuito 3.10 inizia creando, con i gate Hadamard, la sovrapposizione uniforme sui primi due qubit, ovvero lo stato

$$|\psi_1\rangle = \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x\rangle |y\rangle |0\rangle. \quad (3.107)$$

Operando con il gate U_f su $|\psi_1\rangle$, in quanto il terzo qubit è nello stato $|0\rangle$, si arriva a

$$|\psi_2\rangle = \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x\rangle |y\rangle |a^x b^y \bmod p\rangle = \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x, y, a^{x+sy} \bmod p\rangle. \quad (3.108)$$

Si definiscono, ora, in modo analogo alla sezione 3.2.2 per l'individuazione dell'ordine, gli stati u_k

$$|u_k\rangle := \frac{1}{\sqrt{r}} \sum_{l=0}^{r-1} \exp\left(-\frac{2\pi i}{r} kl\right) |a^k \bmod p\rangle. \quad (3.109)$$

Si definisce, inoltre, per come opera sugli stati $|j\rangle$ della base computazionale, l'operatore U_a unitario, come mostrato nel lemma 3,

$$U_a(|j\rangle) := |aj \bmod p\rangle. \quad (3.110)$$

Svolgendo i calcoli, in quanto a ha ordine r , risulta chiaro che

$$U_a(|u_k\rangle) = \exp\left(\frac{2\pi i}{r} k\right) |u_k\rangle, \quad (3.111)$$

ovvero che i k elementi $|u_k\rangle$ sono autovettori dell'operatore U_a , con autovalori $\exp\left(-\frac{2\pi i}{r} k\right)$. Operando, ora, come nella sezione 3.2.2, sull'algoritmo di individuazione dell'ordine, si dimostra che

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_k\rangle = |1\rangle. \quad (3.112)$$

In particolare si può osservare che, applicando $x + sy$ volte l'operatore U_a , si manda

$|1\rangle$ in $|a^{x+sy} \bmod p\rangle$. Inoltre lo stato $|1\rangle$ è descrivibile dalla somma in (3.112), ovvero dalla somma di autovettori di U_a . A partire da queste indicazioni si può facilmente desumere l'uguaglianza

$$|x, y, a^{x+sy} \bmod p\rangle = |x, y\rangle \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(\frac{2\pi i}{r} k(x+sy)\right) |u_k\rangle. \quad (3.113)$$

Segue che lo stato $|\psi_2\rangle$ può essere riscritto come

$$|\psi_2\rangle = \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x, y\rangle \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left(\frac{2\pi i}{r} k(x+sy)\right) |u_k\rangle. \quad (3.114)$$

Applicando la trasformata di Fourier inversa, infine, si arriva allo stato

$$|\psi_3\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left(\sum_{x',y'=0}^{2^t-1} \frac{1}{2^t} \left(\sum_{x=0}^{2^t-1} \omega_{2^t}^{-xx'} \omega_r^{kx} \right) |x'\rangle \frac{1}{2^t} \left(\sum_{y=0}^{2^t-1} \omega_{2^t}^{-yy'} \omega_r^{ksy} \right) |y'\rangle \right) |u_k\rangle, \quad (3.115)$$

in cui $\omega_r := \exp\left(\frac{2\pi i}{r}\right)$. Inoltre i coefficienti di $|x'\rangle$ sono:

$$\alpha_{x'} := \sum_{x=0}^{2^t-1} \omega_{2^t}^{-xx'} \omega_r^{kx} = \sum_{x=0}^{2^t-1} (\omega_{2^t}^{-x'} \omega_r^k)^x = \sum_{x=0}^{2^t-1} (e^{2\pi i(k/r - x'/2^t)})^x. \quad (3.116)$$

Calcolando la somma parziale della serie geometrica, come in (3.57), si ottiene

$$\alpha_{x'} = \sum_{x=0}^{2^t-1} \omega_{2^t}^{-xx'} \omega_r^{kx} = \frac{1 - e^{2\pi i 2^t(k/r - x'/2^t)}}{1 - e^{2\pi i(k/r - x'/2^t)}}. \quad (3.117)$$

Si può svolgere un calcolo analogo per i coefficienti di $|y'\rangle$ ed ottenere

$$\alpha_{y'} := \sum_{y=0}^{2^t-1} \omega_{2^t}^{-yy'} \omega_r^{ksy} = \frac{1 - e^{2\pi i 2^t(ks/r - y'/2^t)}}{1 - e^{2\pi i(ks/r - y'/2^t)}}. \quad (3.118)$$

Sia, ora, k fissato. Per studiare il funzionamento dell'algoritmo si riprendono le stime ricavate nello studio dell'algoritmo per la stima di fase, alla sezione 3.2.1 Si introducono due valori c e d , i quali rappresentano le migliori stime che possono essere misurate, rispettivamente per $2^t k/r$ e $2^t ks/r$. In particolare si pongono $c := \lfloor 2^t k/r \rfloor$ e $d := \lfloor 2^t ks/r \rfloor$. Si introduce anche una tolleranza e su tali stime, che si ripercuote in una tolleranza $2^{-t}e$ sulla stima di k/r e ks/r . Chiamate φ e ψ le misure, rispettivamente del primo e del secondo registro, si ottengono le seguenti stime, ottenute come mostrato fino a (3.65):

$$\mathbb{P}(|\varphi - c| > e) \leq \frac{1}{2(e-1)} \quad \text{e} \quad (3.119)$$

$$\mathbb{P}(|\psi - d| > e) \leq \frac{1}{2(e-1)}. \quad (3.120)$$

Seguendo sempre l'analisi per l'algoritmo di stima di fase si ottiene un'espressione per la lunghezza t dei primi due registri. Si vuole ottenere una stima di k/r e ks/r

accurata a $2L + 1$ bit, con probabilità almeno $1 - \epsilon$. Si deve, quindi, scegliere $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$.

Inoltre, lo stato (3.115) è, nel terzo registro, in una sovrapposizione uniforme degli autovettori $|u_k\rangle$ di U_a . La probabilità di misurare una stima accurata, per un fissato k , di k/r e ks/r sarà di $(1 - \epsilon)/k$.

La scelta di prendere t maggiore di $2L + 1$ è giustificata dalla necessità di operare, come per l'individuazione dell'ordine, descritto in sezione 3.2.2, con l'algoritmo dell'espansione in frazione continua. Tale scelta, infatti, permette di rispettare le ipotesi del teorema 3.2.1, il quale garantisce che k/r è un convergente della frazione continua di φ e ks/r di ψ .

Operando con l'algoritmo per le frazioni continue, quindi, si ottengono una frazione ridotta ai minimi termini per k/r e una per ks/r . Moltiplicando i due risultati per r , che è un valore noto, si ottengono sempre k e ks . Dividendo la seconda stima per la prima si ottiene esattamente s , come richiesto. La probabilità che tale procedura abbia successo dipende solamente dall'accuratezza della stima di k/r e di ks/r . Esse hanno, indipendentemente, probabilità $1 - \epsilon$ di rientrare nei criteri stabiliti, ovvero l'algoritmo produce un risultato corretto con probabilità almeno $(1 - \epsilon)^2$.

Si può, inoltre, generalizzare questo algoritmo a ogni gruppo moltiplicativo. Per far ciò a può essere scelto in modo arbitrario, tra gli elementi di ordine maggiore di 1. b , invece, deve essere una potenza di a , per cui deve essere scelto in $\langle a \rangle$. Grazie all'algoritmo di ricerca dell'ordine si può individuare l'ordine r di a e operare con quel valore, al posto che $p - 1$, implicitamente considerato in questa sezione.

Costo computazionale

Percorrendo il circuito in ordine di esecuzione si incontrano: Due gate di Hadamard, con un costo computazionale di $O(L)$ ciascuno, per un totale di $O(L)$. Il gate U_f , la cui implementazione può essere data da due algoritmi per l'esponenziazione modulare in serie. Ciascuno di essi ha un costo di $O(L^3)$ operazioni, come mostrato in 3.2.2, nella sezione per l'algoritmo di ricerca dell'ordine. Il gate U_f ha, quindi, un costo complessivo di $O(L^3)$. Compagnano, infine, due trasformate di Fourier inverse, che richiedono $O(L^2)$ operazioni elementari ciascuna.

Segue a questo l'esecuzione dell'algoritmo per le frazioni continue il quale ha un costo di $O(L^3)$ operazioni. In realtà questo algoritmo può essere eseguito da un computer classico, lasciando libero il computer quantistico di operare con algoritmi più consoni alle proprie caratteristiche.

Combinando tutti questi risultati si ottiene che la complessità della dimensione del circuito per la risoluzione del problema del logaritmo discreto è di $O(L^3)$ porte logiche elementari.

Algoritmo: Logaritmo discreto quantistico.

Input: (1) $a, b \in (\mathbb{Z}/p\mathbb{Z})^*$ t.c. $(\mathbb{Z}/p\mathbb{Z})^* = \langle a \rangle$, (2) 2 registri contenenti $t = 2L + 1 + \lceil \log_2(2 + \frac{1}{2\epsilon}) \rceil$ qubit inizializzati nello stato $|0\rangle$, (3) L qubit inizializzati nello stato $|0\rangle$.

Output: il minimo $s \in \mathbb{N}$ tale che $a^s = b \pmod{p}$.

Runtime: $O(L^3)$ operazioni. Ritorna un output corretto con probabilità $(1 - \epsilon)^2$.

Procedura:

1. $|0\rangle |0\rangle |0\rangle$ stato iniziale
2. $\xrightarrow{H^{\otimes 2t}} \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x\rangle |y\rangle |0\rangle$ sovrapposizione iniziale
3. $\xrightarrow{U_a} \frac{1}{2^t} \sum_{x,y=0}^{2^t-1} |x, y, a^{x+sy} \bmod p\rangle$ risultato U_a

$$= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left(\sum_{x',y'=0}^{2^t-1} \frac{1}{2^t} \left(\sum_{x=0}^{2^t-1} \omega_{2^t}^{-xx'} \omega_r^{kx} \right) |x'\rangle \frac{1}{2^t} \left(\sum_{y=0}^{2^t-1} \omega_{2^t}^{-yy'} \omega_r^{ksy} \right) |y'\rangle \right) |u_k\rangle$$
4. $\xrightarrow{FT^\dagger} \frac{1}{\sqrt{r}} \sum_{s=0}^{p-2} \widetilde{|k/r\rangle} \widetilde{|ks/r\rangle} |u_k\rangle$ trasformata Fourier inversa
5. $\mapsto \left(\widetilde{k/r}, \widetilde{ks/r} \right)$ misura primi due registri
6. $\mapsto (k, ks)$ algoritmo frazioni continue
7. $\mapsto s$

Appendice A

Frazioni continue

Definizione A.1: frazione continua finita.

Si definisce frazione continua finita la funzione

$$[a_0; a_1, a_2, \dots, a_n] := a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}} \quad (\text{A.1})$$

delle $n + 1$ variabili $a_0, a_1, \dots, a_n \in \mathbb{R}$.

Si chiama j -esimo convergente della frazione continua il termine

$$[a_0; a_1, a_2, \dots, a_j] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_j}}}}, \quad (\text{A.2})$$

dove $0 \leq j < n$.

Teorema A.0.1. *Sia a_0, \dots, a_N una successione di reali positivi. Si ha che*

$$[a_0; a_1, \dots, a_n] = \frac{p_n}{q_n} \quad (\text{A.3})$$

dove p_n e q_n sono numeri reali definiti iterativamente da:

(1) $p_0 := a_0$ e $q_0 := 1$

(2) $p_1 := 1 + a_0 a_1$ e $q_1 := a_1$

(3) per $2 \leq n \leq N$

$$p_n := a_n p_{n-1} + p_{n-2} \quad (\text{A.4})$$

$$q_n := a_n q_{n-1} + q_{n-2} \quad (\text{A.5})$$

Se, inoltre, $a_j \in \mathbb{N}^+$ per ogni j , segue che anche $p_j, q_j \in \mathbb{N}^+$ per ogni j .

Dimostrazione. Nel caso di $n = 0$ è ovvio, mentre nei casi $n = 1, 2$ è una semplice verifica. Si opera, poi, per induzione su n , con caso base $n = 2$, già affrontato.

Sia, ora $n \geq 3$, e valga l'ipotesi induttiva. Risulta chiaro, per definizione di frazione continua, che vale la seguente

$$[a_0; \dots, a_{n-1}, a_n] = [a_0; \dots, a_{n-2}, a_{n-1} + \frac{1}{a_n}] \quad (\text{A.6})$$

Siano \tilde{p}_j e \tilde{q}_j le successioni, definite come nell'enunciato del teorema, associate alla frazione continua che compare nel membro a destra di (A.6). Chiaramente $\tilde{p}_{n-3} = p_{n-3}$, $\tilde{p}_{n-2} = p_{n-2}$ e $\tilde{q}_{n-3} = q_{n-3}$, $\tilde{q}_{n-2} = q_{n-2}$, dove p_j e q_j sono le successioni associate alla frazione continua a sinistra in (A.6).

Per ipotesi induttiva vale la seguente uguaglianza:

$$[a_0; \dots, a_{n-2}, a_{n-1} + \frac{1}{a_n}] = \frac{\tilde{p}_{n-1}}{\tilde{q}_{n-1}}. \quad (\text{A.7})$$

Dove, per definizione di \tilde{p}_{n-1} e \tilde{q}_{n-1} , la frazione a destra può essere riscritta come

$$\frac{\tilde{p}_{n-1}}{\tilde{q}_{n-1}} = \frac{(a_{n-1} + 1/a_n)p_{n-2} + p_{n-3}}{(a_{n-1} + 1/a_n)q_{n-2} + q_{n-3}} \quad (\text{A.8})$$

$$= \frac{p_{n-1} + p_{n-2}/a_n}{q_{n-1} + q_{n-2}/a_n}, \quad (\text{A.9})$$

in cui, l'ultima uguaglianza è dovuta alla definizione di p_{n-1} e q_{n-1} . Moltiplicando l'ultimo termine per $1 = a_n/a_n$ si ottiene

$$\frac{\tilde{p}_{n-1}}{\tilde{q}_{n-1}} = \frac{a_n p_{n-1} + p_{n-2}}{a_n q_{n-1} + q_{n-2}} = \frac{p_n}{q_n} \quad (\text{A.10})$$

Combinando le equazioni (A.6), (A.7) e (A.10) si ottiene

$$[a_0; \dots, a_{n-1}, a_n] = \frac{p_n}{q_n} \quad (\text{A.11})$$

ovvero la tesi. ■

Si ha, quindi, un algoritmo per trovare le frazioni associate ai successivi convergenti di una data frazione continua. Inoltre vale il seguente lemma:

Lemma 5. $q_n p_{n-1} - p_n q_{n-1} = (-1)^n$ per ogni $n \geq 1$

Dimostrazione. Si opera per induzione su n . Sia $n = 1$, allora

$$q_1 p_0 - p_1 q_0 = a_1 a_0 - (1 + a_0 a_1) = -1 = (-1)^n. \quad (\text{A.12})$$

Sia $n > 1$. Per definizione iterativa di p_n e q_n vale

$$q_n p_{n-1} - p_n q_{n-1} = (a_n q_{n-1} + q_{n-2})p_{n-1} - (a_n p_{n-1} + p_{n-2})q_{n-1} \quad (\text{A.13})$$

$$= q_{n-2} p_{n-1} - p_{n-2} q_{n-1} = (-1)(q_{n-1} p_{n-2} - p_{n-1} q_{n-2}) \quad (\text{A.14})$$

Per ipotesi induttiva $q_{n-1} p_{n-2} - p_{n-1} q_{n-2} = (-1)^{n-1}$, da cui si ottiene

$$q_n p_{n-1} - p_n q_{n-1} = (-1)(-1)^{n-1} = (-1)^n \quad (\text{A.15})$$

■

Non era necessario restringere ulteriormente la definizione di *frazione continua* per ottenere i risultati precedenti. Il concetto con cui opereremo, però, è il più specializzato concetto di *frazione continua semplice*, di cui segue la definizione.

Definizione A.2: frazione continua finita semplice.

Si definisce frazione continua finita semplice una frazione continua finita

$$[a_0; a_1, a_2, \dots, a_n] := a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}}, \quad (\text{A.16})$$

in cui $a_0, \dots, a_n \in \mathbb{N}^+ := \{1, 2, \dots\}$.

Risulta chiaro che ogni frazione continua finita semplice rappresenta un numero razionale, però, per i numeri razionali maggiori di 1, vale anche il viceversa. È utile introdurre un potente ed efficiente algoritmo per il calcolo delle frazioni continue per mostrare tale risultato:

Algoritmo: Espansione in frazione continua.

Ogni numero razionale maggiore di 1 ha una rappresentazione in frazione continua finita semplice, ovvero con un'espressione come

$$[a_0; a_1, a_2, \dots, a_n] := a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}}, \quad (\text{A.17})$$

in cui $a_0, \dots, a_n \in \mathbb{N}^+$. Questa espansione si può trovare operando con il seguente

Algoritmo: Sia $x = p_0/p_1$ un numero razionale. Si ricavano, iterativamente, a_j operando nel seguente modo. $a_0 = \lfloor x \rfloor$ è la parte intera di x , per cui

$$x = a_0 + \frac{p_2}{p_1} = a_0 + \frac{1}{\frac{p_1}{p_2}}, \quad (\text{A.18})$$

in cui, nell'ultimo passaggio, si è invertita la frazione. In particolare, in quanto $a_0 = \lfloor x \rfloor$ segue che $p_2/p_1 < 1$, ovvero $p_2 < p_1$. Chiamato $x_1 := p_1/p_2$ si ripete su x_1 il processo appena svolto, ad ottenere a_1 e $x_2 := p_2/p_3$, con $p_3 < p_2$. Iterando si calcolano a_n, x_{n+1} e p_{n+2} a partire da x_n , costruendo un'espansione in frazione continua per x . Tale algoritmo si arresta quando $x_n = \lfloor x_n \rfloor$ è intero, ovvero quando $p_{n+1} = 0$. In particolare, per ogni x razionale, p_j è una successione a coefficienti in \mathbb{N}^+ e strettamente decrescente. Ne consegue che, in un numero finito di passi $p_j = 0$, ovvero l'algoritmo si ferma.

In realtà, oltre all'espansione in frazione continua semplice trovata da questo algoritmo, per ogni numero razionale si può trovare un'espansione equivalente. Per fare ciò, all'ultimo passo, invece che scegliere $a_n = a_n$, si può esprimere $a_n = (a_n - 1) + 1/1$, ottenendo un'espansione con $\tilde{a}_n = a_n - 1$ e $\tilde{a}_{n+1} = 1$. Quest'ambiguità tornerà utile,

in quanto permette di esprimere un numero razionale con una frazione continua semplice con numero pari o dispari di coefficienti. Inoltre tale algoritmo, se si concede $a_0 = 0$, può individuare espansioni in frazioni continue “semplici” per numeri razionali positivi, anche minori di 1.

Prima di passare al prossimo risultato è utile evidenziare un paio di osservazioni. Innanzitutto, nel caso di frazioni continue semplici, il lemma 5 ha come conseguenza che, per ogni n , p_n e q_n sono coprimi tra loro. Questo segue dall’identità di Bezout, la quale afferma che, per $m, n \in \mathbb{N}$, esistono $a, b \in \mathbb{Z}$ tali che $am + bn = \text{MCD}(a, b)$. Se $q_n p_{n-1} - p_n q_{n-1} = (-1)^n$ segue che $\text{MCD}(q_n, p_n) \mid 1$, ovvero la coprimialità. Si ha, quindi, che, nel calcolo dei convergenti, l’algoritmo dato dal teorema A.0.1 restituisce solo frazioni ridotte ai minimi termini.

Un’ulteriore conseguenza utile del teorema A.0.1 segue dalla definizione iterativa di p_n e di q_n . Nel caso di una frazione continua semplice $a_0, \dots, a_n \in \mathbb{N}^+$, per cui $\#j : a_j = 0$, implica che

$$p_n = a_n p_{n-1} + p_{n-2} = a_n (a_{n-1} p_{n-2} + p_{n-3}) + p_{n-2} > 2p_{n-2}. \quad (\text{A.19})$$

Per concludere questa sezione, infine, è necessario dimostrare un risultato fondamentale nell’implementazione dell’algoritmo per l’individuazione dell’ordine, ovvero il seguente teorema:

Teorema A.0.2. *Siano x ed p/q razionali tali che*

$$\left| \frac{p}{q} - x \right| \leq \frac{1}{2q^2} \quad (\text{A.20})$$

Allora p/q è un convergente della frazione continua di x

Dimostrazione. Se, per caso $x = p/q$, l’affermazione è ovvia. Altrimenti sia $p/q = [a_0; \dots, a_n]$ l’espansione in frazione continua semplice di p/q . Siano p_j e q_j definiti come specificato nel teorema A.0.1, in modo tale che $p_n/q_n = p/q$. Si definisce, inoltre, δ in modo che

$$x = \frac{p_n}{q_n} + \frac{\delta}{2q_n^2}. \quad (\text{A.21})$$

Da (A.20) e dal fatto che $q_n \mid q$ si ha che $0 < |\delta| < 1$. Si definisce, infine, λ come dalla seguente equazione

$$\lambda := 2 \left(\frac{q_n p_{n-1} - p_n q_{n-1}}{\delta} \right) - \frac{q_{n-1}}{q_n}. \quad (\text{A.22})$$

Questa espressione è scelta affinché λ soddisfi l’equazione

$$x = \frac{\lambda p_n + p_{n-1}}{\lambda q_n + q_{n-1}}, \quad (\text{A.23})$$

da cui una frazione continua per x è $[a_0; \dots, a_n, \lambda]$. Si sceglie, ora, n pari se $\delta > 0$ e n dispari se $\delta < 0$, come concesso dall’osservazione che segue l’algoritmo per l’espansione in frazione continua. Per il lemma 5, $q_n p_{n-1} - p_n q_{n-1} = (-1)^n$. Segue che

$$2 \left(\frac{q_n p_{n-1} - p_n q_{n-1}}{\delta} \right) = \frac{2}{|\delta|} > 0, \quad (\text{A.24})$$

e, in particolare, λ diventa

$$\lambda = \frac{2}{\delta} - \frac{q_{n-1}}{q_n}. \quad (\text{A.25})$$

In quanto $|\delta| < 1$ e la successione q_j è strettamente crescente vale che

$$\lambda = \frac{2}{\delta} - \frac{q_{n-1}}{q_n} > 2 - 1 > 1 \quad (\text{A.26})$$

ovvero λ è un numero razionale e maggiore di 1. Per questo motivo ammette un'espansione in frazione continua finita e semplice $\lambda = [b_0, \dots, b_m]$. Segue da ciò che l'espansione *semplice* per x è $[a_0, \dots, a_n, b_0, \dots, b_m]$, di cui p/q è un convergente. ■

Appendice B

Crittografia

La crittografia è lo strumento utilizzato per offuscare i messaggi in modo tale che siano accessibili solo a mittente e destinatario, risultando indecifrabili a terze parti. Le motivazioni per utilizzare tali strumenti sono molteplici e hanno spinto a un forte sviluppo nelle tecniche impiegate. L'idea alla base di queste è che certe operazioni sono più veloci, o semplici, da eseguire rispetto ad altre, come studiato nel capitolo 2, concernente la complessità computazionale. Si ricercano, infatti, operazioni che possano essere svolte in modo efficiente, ma per le quali l'inversa non ammetta algoritmi veloci.

La situazione ideale, quindi, sarebbe di un problema in \mathbf{P} , con inverso in \mathbf{NP} , assumendo che $\mathbf{P} \neq \mathbf{NP}$. Purtroppo non sempre è questo il caso, per cui alcuni sistemi crittografici utilizzati sfruttano la mancanza di conoscenze attuali come base per la propria sicurezza. Un tale esempio, il cui comportamento verrà spiegato a breve, è il sistema di crittografia a chiave pubblica RSA, sviluppato da Rivest, Shamir e Adleman. Dopo aver spiegato il funzionamento di tale sistema si sfrutteranno gli algoritmi sviluppati nel capitolo 3 per mostrare l'inefficienza di questo sistema nei confronti dei computer quantistici.

B.1 Crittografia a chiave pubblica

Il concetto dietro al paradigma della crittografia a chiave pubblica è quello alla base di una casella della posta: si creano due chiavi distinte, una privata e una pubblica. Si distribuisce la chiave pubblica affinché chiunque sia in grado di offuscare, crittare, i propri messaggi, analogamente a mettere il messaggio nella casella della posta. Tale chiave, però, non deve essere in grado, né deve rendere facile, decrittare messaggi già crittati, così come da una casella della posta non si possono togliere messaggi già inseriti. È solo la chiave privata che può compiere questa azione, dando accesso al possessore a tutti i messaggi crittati, così come la chiave della casella postale permette l'accesso a tutti i messaggi in essa contenuta.

B.1.1 Sistema RSA

Il crittosistema RSA si basa su svariate proprietà dei campi finiti e sull'apparente complessità del problema di fattorizzazione dei numeri interi.

Le chiavi sono coppie di numeri interi e si generano nel seguente modo:

- Si scelgono due numeri naturali p e q primi molto grandi.
- Si prende il loro prodotto $n := pq$, chiamato *modulo* e si calcola la funzione $\varphi(n) = (p-1)(q-1)$.
- Si sceglie un numero e , chiamato *esponente pubblico*, coprimo con $\varphi(n)$ e minore di esso.
- Si calcola d , detto *esponente privato*, un numero tale che $ed = 1 \pmod{\varphi(n)}$.

La chiave pubblica è $P = (n, e)$, mentre la privata $S = (n, d)$.

Per crittografare un messaggio M si assume che sia codificato in meno di $\lfloor \log_2(n) \rfloor$. Altrimenti è sufficiente dividere il messaggio in blocchi di lunghezza al più $\lfloor \log_2(n) \rfloor$ e cifrare ciascuno di essi indipendentemente. Il messaggio M , tramite la chiave pubblica, viene crittato in

$$E(M) := M^e \pmod{n}, \quad (\text{B.1})$$

con l'utilizzo dell'algoritmo di esponenziazione modulare.

Il processo di decrittazione opera con la seguente operazione:

$$D(E(M)) := E(M)^d = M^{ed} \pmod{n}. \quad (\text{B.2})$$

Anche questo calcolo può essere effettuato in modo efficiente con l'algoritmo di esponenziazione modulare. Bisogna mostrare che questa operazione, indipendentemente dal messaggio M cifrato, permetta di decodificarlo.

Lemma 6. *Per d, e, n ed M come sopra, vale:*

$$M^{ed} = M \pmod{n} \quad (\text{B.3})$$

Dimostrazione. Innanzitutto si nota che, per costruzione, $ed = 1 \pmod{n}$, ovvero esiste $k \in \mathbb{N}$ tale che $ed = 1 + k\varphi(n)$. Per eseguire la dimostrazione si dividono i due casi:

M coprimo con n : Essendo coprimo con n , M è un elemento invertibile di $\mathbb{Z}/n\mathbb{Z}$. Si può, quindi, applicare il piccolo teorema di Fermat:

$$M^{ed} = M^{1+k\varphi(n)} = M \cdot M^{k\varphi(n)} = M \pmod{n}. \quad (\text{B.4})$$

M non coprimo con n : M non è coprimo con n implica che almeno uno tra p e q divide M . Si suppone, per iniziare, che p divida M , ma q non lo faccia. In questo caso $M = 0 \pmod{p}$, per cui $M^{ed} = 0 \pmod{p}$. In quanto q non divide M si ha $M \in \mathbb{Z}/q\mathbb{Z}^*$. Segue che M ha ordine moltiplicativo un divisore di $q-1 = |\mathbb{Z}/q\mathbb{Z}|$, ovvero $M^{q-1} = 1 \pmod{q}$. In quanto $\varphi(n) = (q-1)(p-1)$ si ha anche che $M^{\varphi(n)} = 1 \pmod{q}$. Come osservato in precedenza $ed = 1 + k\varphi(n)$, da cui $M^{ed} = M \pmod{q}$. Per il teorema cinese dei resti, teorema 3.2.4, segue che $M^{ed} = M \pmod{n}$.

Se, invece, q , ma non p , divide M è sufficiente ripercorrere i passaggi precedenti scambiando i ruoli di p e q . Se, infine, sia q che p dividono M allora $M = 0 \pmod{n}$, per cui $M^{ed} = 0 = M \pmod{n}$. ■

Sfruttando questo lemma si ottiene che $D(E(M)) = M$ per ogni M di lunghezza appropriata. Per cui il processo di decrittazione riesce sempre a ricavare il messaggio cifrato.

Richieste computazionali

Nell'implementazione si devono considerare in maniera indipendente due problemi: la creazione delle chiavi e il processo usato per crittare e decrittare il messaggio. Affinché un sistema crittografico sia utile a livello pratico entrambi questi problemi devono essere facilmente risolvibili, in presenza delle dovute chiavi. È per questo motivo che si studia la loro richiesta computazionale.

Per quanto riguarda il primo problema la difficoltà principale è posta dall'individuazione di due numeri primi sufficientemente grandi, in quanto il resto delle operazioni è efficientemente eseguibile. Il migliore metodo individuato è quello di creare numeri casuali e testare la primalità di tali numeri. Per il problema del test di primalità, come descritto in [AKS04], sono noti algoritmi efficienti, che impegnano meno di $O(L^{15/2})$ operazioni, ove L è la lunghezza, binaria, del numero in input. Se il numero scelto casualmente è primo si tiene il numero, se è composto lo si scarta e si ripetono i passaggi precedenti. Il teorema dei numeri primi afferma che $\pi(n) \sim n/\log(n)$, dove $\pi: \mathbb{N} \rightarrow \mathbb{N}$ è la funzione che conta i primi $p \leq n$. Da questo risultato si può stimare la probabilità di ottenere un numero primo, scegliendo uniformemente un numero tra 0 e $2^L - 1$, con $1/\log(2^L) = 1/L$.

Operando L volte con il procedimento appena descritto, quindi, si ha una grande probabilità di ottenere un primo cercato. Tale ricerca, quindi, ha una richiesta di $O(L^{17/2})$ operazioni, risultando comunque efficiente. Algoritmi più efficienti per la verifica della primalità possono abbassare tale richiesta fino a $O(L^3)$ operazioni, come descritto in [NC11].

A questo si aggiungono solo la ricerca di e e del suo inverso moltiplicativo d . La ricerca del primo valore, in genere, ha un impatto trascurabile sulle richieste computazionali: a volte il valore di e è persino hardcoded negli algoritmi, o al più, è ristretto in un piccolo intervallo. In genere tale intervallo è scelto per massimizzare l'efficienza dell'algoritmo di cifratura, che usa la chiave pubblica – contenente e . Individuato e , poi, il calcolo del suo inverso d modulo $\varphi(n)$ si effettua tramite l'algoritmo di Euclide. In particolare questo procedimento ha una spesa di $O(L^3)$ operazioni – si veda [Knu97] per un'analisi dettagliata.

Per quanto riguarda il secondo problema, invece, si opera con l'algoritmo 3.2.2 per l'esponentiazione modulare, il quale, come descritto in precedenza, ha una richiesta di $O(L^3)$ operazioni.

Il sistema di crittografia RSA risulta, quindi, praticamente implementabile in computer classici.

B.1.2 Bucare RSA

Ci sono due principali strade per eludere la cifratura, senza conoscere la chiave privata.

Il primo metodo consiste nell'individuare una fattorizzazione di n , primo elemento della chiave pubblica. Si ottengono, quindi, p e q , con i quali è semplice computare $\varphi(n) = (p-1)(q-1)$. Conoscendo questo valore risulta facile calcolare l'inverso di d modulo $\varphi(n)$ operando, come descritto in precedenza, con l'algoritmo di Euclide. L'inverso di d è proprio e : conoscendo la fattorizzazione di n si ottiene efficientemente la chiave privata.

Il secondo metodo, invece, non comporta l'individuazione della chiave privata, ma permette lo stesso di decifrare il messaggio cifrato. Sia M un messaggio, cifrato in $M^e \bmod n$. Se M^e non è invertibile in $\mathbb{Z}/n\mathbb{Z}$, allora $\text{MCD}(M^e, n) \neq 1$, per cui, con

l'algoritmo di Euclide, si può estrarre un fattore primo di n , ovvero p o q . Da questo punto si può operare come sopra e ottenere la chiave privata.

Se, invece, M^e è invertibile ne si può calcolare l'ordine moltiplicativo r , ovvero il minimo intero tale che $(M^e)^r = 1 \pmod n$. Per il teorema di Lagrange si sa che $r \mid \varphi(n)$, in quanto $\varphi(n)$ è l'ordine del gruppo moltiplicativo $\mathbb{Z}/n\mathbb{Z}^*$. Il numero e è coprimo con $\varphi(n)$, per costruzione, quindi è coprimo anche con r . Per questo motivo e ammette un inverso d' modulo r . Esiste, quindi, un $k \in \mathbb{N}$ tale che $ed' = 1 + kr$. Inoltre l'ordine di M^e è esattamente l'ordine \tilde{r} di M diviso per $\text{MCD}(e, \tilde{r})$. In quanto, come prima, $\tilde{r} \mid \varphi(n)$, si desume che e è coprimo anche con \tilde{r} , per cui $\text{MCD}(e, \tilde{r}) = 1$. Si ha, quindi, che $\tilde{r} = r$, per cui $M^r = M^{kr} = 1 \pmod n$.

Utilizzando il numero d' calcolato si ottiene, quindi:

$$(M^e)^{d'} = M^{1+kr} = M \cdot M^{kr} = M \pmod n, \quad (\text{B.5})$$

ovvero il messaggio originale M , supponendo che fosse di lunghezza appropriata.

Il motivo per cui si ritiene il sistema di crittografia RSA sicuro è che i metodi appena descritti devono risolvere problemi per i quali si crede non esistano algoritmi classici efficienti. Purtroppo questa è solo una convinzione informale, non si hanno risultati teorici che garantiscano la difficoltà del problema di fattorizzazione o di ricerca dell'ordine su computer classici. Risulta, infatti, un sistema inefficace in ambito quantistico, in quanto entrambi tali problemi sono risolvibili con un algoritmo efficiente, ovvero sono in **BQP**.

Il fatto che siano state descritte due diverse strade per aggirare tale sistema crittografico, di cui una che non richiede nemmeno la chiave privata, dovrebbe far porre delle domande riguardo alla sicurezza del sistema. In realtà, come si è mostrato nel paragrafo 3.2.3, un algoritmo di individuazione dell'ordine efficiente può essere usato per implementare un algoritmo efficiente per la fattorizzazione. Praticamente, quindi, risulta che, se si dovesse riuscire a seguire una qualsiasi delle due strade, si potrebbe ottenere la chiave privata a partire da quella pubblica.

Risulta, però, un problema la possibilità di decifrare il messaggio anche senza possedere la chiave privata. Inoltre il fatto che non siano stati descritti altri possibili metodi per aggirare la crittografia RSA non significa che non ne esistano, né che non ne esistano di efficienti. In sostanza, dal punto di vista teorico, non si è dimostrata la sicurezza di tale metodo, ma solo la difficoltà di aggirarlo con le conoscenze attuali.

In quanto, però, dopo due decenni di ricerca non si sono trovate altre vulnerabilità degne di nota porta a fidarsi, nella pratica, del sistema crittografico appena descritto. Semplicemente si ha la consapevolezza che l'ambito di efficacia di tale sistema è limitata alla computazione classica, in quanto risulta facilmente aggirabile da computer quantistici.

Bibliografia

- [AKS04] Manindra Agrawal, Neeraj Kayal e Nitin Saxena. «PRIMES Is in P». In: *Annals of Mathematics* 160.2 (2004), pp. 781–793.
- [Ber98] Daniel J. Bernstein. «Detecting Perfect Powers In Essentially Linear Time». In: *Mathematics of computation* 67 (1998), pp. 1253–1283.
- [FT82] Edward Fredkin e Tommaso Toffoli. «Conservative logic». In: *International Journal of Theoretical Physics* 21.3 (apr. 1982), pp. 219–253.
- [Gud03] Stan Gudder. «Quantum Computation». In: *The American Mathematical Monthly* 110.3 (2003), pp. 181–201. eprint: <https://doi.org/10.1080/00029890.2003.11919955>.
- [Jac12] Nathan Jacobson. *Basic Algebra I: Second Edition*. Dover Books on Mathematics. Dover Publications, 2012.
- [Knu97] Donald E Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. 3ed. Vol. 2. Addison-Wesley series in computer science and information processing. Addison-Wesley Professional, 1997.
- [NC11] Michael A. Nielsen e Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 1^a ed. Cambridge Series on Information and the Natural Sciences. Cambridge University Press, 2011.
- [Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [San09] Boaz Barak Sanjeev Arora. *Computational complexity: A modern approach*. 1^a ed. Cambridge University Press, 2009.
- [Sho97] Peter W. Shor. «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer». In: *SIAM Journal on Computing* 26.5 (ott. 1997), pp. 1484–1509.
- [Tof80] Tommaso Toffoli. «Reversible computing». In: *Automata, Languages and Programming*. A cura di Jaco de Bakker e Jan van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 632–644.