1 FISICA DELLA COMPUTAZIONE E MODELLI ALTERNATIVI DI COMPUTER

1 – PORTE LOGICHE UNIVERSALI

La logica standard del computer è costruita su un numero di limitato di porte logiche. Le più comuni sono le porte: AND, OR e NOT. La porta NOT è indicata con una barra sul bit di input e cambia il valore del bit originale. Ciò vuol dire che $\mathrm{NOT}(A) \equiv \overline{A}$ è uguale a 1 se $A = 0$ e a 1 se $A = 0$.

Le tabelle delle verità associate alle porte AND e OR sono riportate in Tab. **??**. Tradizionalmente l'operazione logica AND su due bit A e B viene indicata alternativamente come A AND B oppure con il simbolo $\wedge$, ovvero A AND $B \equiv A \wedge B$. Il risultato dela porta AND può essere espresso anche in termini di operazioni decimali (più comuni). In particolare, A AND $B = A \cdot B$ dove $A \cdot B$ è l'usuale moltiplicazione decimale. Infatti, se gli input A e B sono binari (assumono solo i valori 0 e 1), $A \cdot B = 1$ solo se entrambi A e B sono uguali a 1 mentre sarà 0 in tutti gli altri casi.

L'operazione logica OR viene indicata come A OR B oppure con il simbolo $\vee$, ovvero A OR $B \equiv A \vee B$.

L'insiemi di AND e NOT oppure di OR e NOT sono insiemi universali. Questo significa che, ad esempio, usando porte solo combinazioni di porte AND e NOT è possibile implementare una qualsiasi funzione booleana [**FeynmanLecturesComputation**, **Functional_completeness**].

Ci sono anche delle singole porte logiche universali che combinate opportunamente permettono di implementare una qualsiasi funzione booleana. Queste sono le porte NAND e NOR. La porta NAND o *negative*-AND è costituita da una porta AND seguita da una porta NOT quindi A NAND $B = \overline{A \wedge B}$. In maniera analoga la porta NOR o *negative*-OR è costituita da una porta OR seguita da una porta NOT quindi A NOR $B = \overline{A \vee B}$. Le tabelle delle verità associate alle porte AND e OR sono riportate in Tab. **??** e Tab. **??**.

A	B	A AND B	A NAND B
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

A	B	A OR B	A NOR B
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

Table 1: Tabella delle verità per le porte AND, NAND, OR e NOR.

Per utilizzo futuro, introduciamo anche la porta *exclusive OR* o XOR la cui tabella delle verità è mostrata Tab. 2. Anche l'operatore XOR può essere associato ad un'operazione decimale indicata con il simbolo $\oplus$: A XOR $B \equiv A \oplus B$. L'operatore $\oplus$ denota la somma modulo 2; ovvero, A XOR $B \equiv A \oplus B = A + B \pmod 2$. Infatti,

essendo A e B binari la loro somma è 0 se $A = B = 0$, è 1 se $A = 0$ e $B = 1$ o $A = 1$ e $B = 0$. Nel caso $A = B = 1$, la loro somma decimale è 2 ma visto che l'operazione è modulo 2, $A + B \pmod 2 = 0$.

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

Table 2: Tabella delle verità della porta XOR.

1.1.1 Porte logiche reversibili

Pur formando dei set universali, le porte AND, OR, NAND e NOR sono però *irreversibili*. Questo perchè ricevono due bit di input ma generano un solo bit di output. Dunque parte dell'informazione iniziale viene persa.

A livello concettuale è interessante introdurre delle porte logiche che siano *reversibili*. Questo vuol dire che se combiniamo in sequenza una porta logica reversibile con la sua inversa, riotteniamo l'informazione originale (ad esempio, la stringa di bit di input). Le porte reversibili devono avere necessariamente un uguale numero di input e output.

Esiste una porta logica che è allo stesso tempo reversibile e universale. Venne proposta nel 1982 da Fredkin e Toffoli [**Fredkin-Toffoli**]. La sua tabella delle verità è riportata in Tab. 3 e mostrata in figura 1. La porta di Fredkin può essere interpretata come uno *switch* controllato di bit. Il bit di controllo è A; se questo è acceso i bit B e C vengono scambiati altrimenti vengono lasciati identici.

In altre parole, il bit di output $O_1 = A$ per ogni combinazione di input. Se $A = 0$, $O_2 = B$ e $O_3 = C$. Se invece $A = 1$, $O_2 = C$ e $O_3 = B$.

Per dimostrare l'unversalità della porta di Fredkin è sufficiente dimostrare che una delle porta universali NAND o NOR può essere costruita con una combinazione di porte di Fredkin. Dalla tabella 3, si può notare che se il bit C è fissato a 1, l'output O_2 è equivalente a una porta OR applicata sui bit A e B. Ovvero, se $C = 1$, $O_2 = A \vee B$. In maniera analoga, notiamo che se fissiamo i bit $B = 0$ e $C = 1$, l'output O_3 è la negazione dell'input A; se $B = 0$ e $C = 1$, $O_3 = \overline{A}$.

Visto che fissando i bit di input è possibile ottenere una porta OR e una porta NOT da una porta di Fredkin, è sufficiente combinarne due per ottenere una porta NOR. Il modo è mostrato in Figura 2. Gli input sono A e B mentre nella prima porta il bit C è fissato a 1. Il secondo output della prima porta di Fredkin diventa il bit di controllo della seconda porta dove i bit di input B e C sono fissati a 0 e 1, rispettivamente.

A	B	C	Out1	Out2	Out3
0	0	1	0	0	1
1	0	1	1	1	0
0	1	1	0	1	1
1	1	1	1	1	1
0	0	0	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
1	1	0	1	0	1

Table 3: Tabella delle verità per la porta di Fredkin.

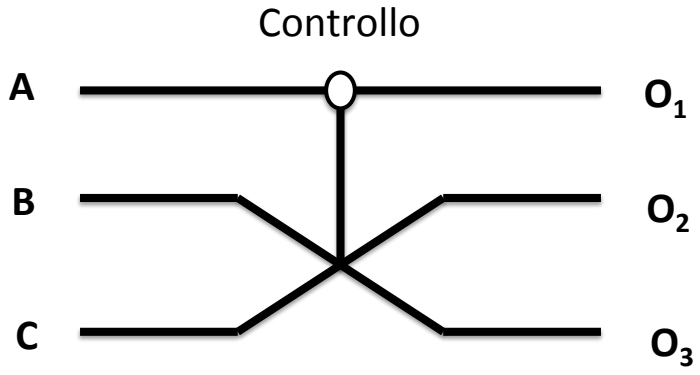

Figure 1: La porta di Fredkin schematizzata come uno switch controllato.

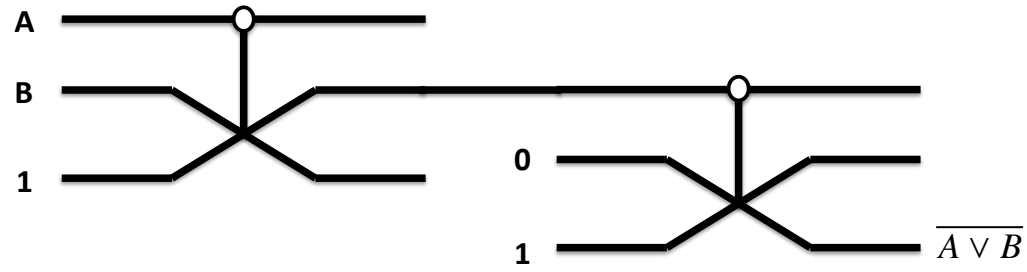

Figure 2: La composizione di due porte di Fredkin può generare una porta NOR.

II – OPERAZIONI BIT-A-BIT

Se invece di singoli bit abbiamo delle stringhe di bit, possiamo comunque manipolarle, sommare o moltiplicarle. Queste operazioni saranno utili in seguito quando parleremo degli algoritmi quantistici. Prima però è importante ricordare come si associa ad un intero una stringa di bit.

Consideriamo uno spazio logico generato da n bit. A una stringa di n bit possiamo associare un intero compreso fra 0 e $N-1$ (associati rispettivamente alle stringhe 000...00 e 111...11) con $N = 2^n$. Al crescere di n lo spazio logico cresce esponenzialmente come 2^n. In genere, è possibile passare da un intero ad una stringa di n bit (con n opportuno) nel seguente modo. All'intero x associamo la stringa di bit $x_1x_2...x_n$ con $x_i = 0, 1$ e $i = 0, 1, ..., n$ tale che $x = x_1 2^{n-1} + x_2 2^{n-2} + ... + x_n 2^0$ [**nielsen-chuang_book**]. Con questa notazione, tradizionalmente usata in informatica, possiamo codificare $N = 2^n$ interi ma questi saranno compresi fra 0 e $N-1 = 2^n - 1$.

Ad esempio, per rappresentare in forma binaria il numero $x = 3$ abbiamo bisogno di 3 bit (la dimensione dello spazio totale sarà 8 e potremo codificare gli interi $0 \leqslant x \leqslant 7$). Possiamo scrivere $x = 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 3$ da cui otteniamo che la stringa associata a è $x = 011$. La stringa $x = 101$ sarà associata all'intero $x = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5$.

1.2.1 Prodotto interno bit-per-bit

Date due stringhe a n bit, $x = x_1, x_2, ..., x_n$ e $z = z_1, z_2, ..., z_n$, indichiamo con $x \cdot z$ il prodotto interno (o scalare) bit-per-bit, modulo 2. Questo equivale a

$$x \cdot z \equiv x_1 z_1 + x_2 z_2 + ... + x_n z_n \text{ (modulo 2)} \tag{1.2.1}$$

Il prodotto $x \cdot z$ è binario e può assumere i valori 0 o 1. É anche chiamato prodotto AND *bitwise* perchè si ottiene prendendo le operazioni AND fra i singoli bit.

Si noti che in fisica e matematica l'operazione AND *bitwise* ricorda il prodotto scalare fra due vettori.

1.2.2 Somma bit-per-bit: *bitwise* XOR

Date due stringhe a n bit, $x = x_1, x_2, ..., x_n$ e $z = z_1, z_2, ..., z_n$, indichiamo con $x \oplus z$ la somma bit-per-bit, modulo 2. Il risultato questa volta è una stringa il cui $i-$esimo bit ha il valore $x_i + z_i (\mathrm{mod}\ 2) = x_i$ XOR z_i. Ovvero

$$\begin{aligned} x \oplus z &\equiv x_1 + z_1 (\mathrm{mod}\ 2), x_2 + z_2 (\mathrm{mod}\ 2), ..., x_n + z_n (\mathrm{mod}\ 2) \\ &= x_1 \text{ XOR } z_1 (\mathrm{mod}\ 2), x_2 \text{ XOR } z_2 (\mathrm{mod}\ 2), ..., x_n \text{ XOR } z_n. \end{aligned} \quad (1.2.2)$$

Anche questo caso, c'è un'analogia con la fisica e matematica dato che l'operazione XOR *bitwise* ricorda la somma fra due vettori.

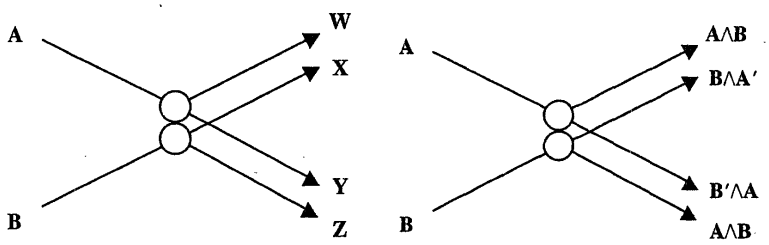

Figure 3: La porta elementare della porta di collisione del computer a palle di biliardo. A sinistra è mostrato lo schema fisico mentre a destra è mostrata la relazione logica fra input e output. L'immagine è presa da [**FeynmanLecturesComputation**] e l'apice indica la negazione: $A' = \overline{A}$.

III — COMPUTER E BILIARDO

Lo studio delle porte logiche elementari, universali e reversibili ci permette di descrivere un'implementazione singolare di un computer [**FeynmanLecturesComputation**]. Gli stessi Fredkin e Toffoli proposero un computer costruibile con palle da biliardo [**Fredkin-Toffoli**]. Sebbene l'idea sia puramente teorica ci permette di comprendere che il calcolo o la manipolazione dei dati e dell'informazione può essere svolta con strumenti differenti da quelli a cui siamo abituati.

Dalla nostra discussione precedente, è chiaro che qualsiasi sistema in cui si possano implementare delle porte logiche universali può essere usato per costruire un computer altrettanto universale.

Una porta logica di un computer a palle di biliardo è mostrata in Figura 4. Due palle da biliardo sulla sinistra e costituiscono gli input logici. La presenza di una palla da biliardo è associata all'input 1 mentre la sua assenza è associata all'input 0. Le palle da biliardo possono proseguire il loro moto in linea retta o urtarsi nel centro.

Supponiamo che gli urti delle palle da biliardo siano elastici (l'energia è sempre conservata) e idealmente precisi. Una descrizione dettagliata del processo fisico implicherebbe l'introduzione del concetto di quantità di moto (o momento) e della sua conservazione. Qui ci limiteremo ad usare l'intuizione e le osservazioni della vita quotidiana.

Se non ci sono palle, $A = 0$ e $B = 0$ e la dinamica è banale dato che tutti gli output saranno nulli $W = 0$, $X = 0$, $Y = 0$ and $Z = 0$. Se c'è una palla inizialmente in A e nessuna in B (a livello logico diremo che $A = 1$ e $B = 0$), questa proseguirà il suo moto uscendo nel punto Y. Negli altri punti W, X, Z non ci saranno palle. In termini, logici per $A = 1$ e $B = 0$ abbiamo $W = 0$, $X = 0$, $Y = 1$ e $Z = 0$. In maniera analoga, se $A = 0$ e $B = 1$ otteniamo $W = 0$, $X = 1$, $Y = 0$ e $Z = 0$. In fine, se $A = 1$ e $B = 1$, le due palle urteranno e usciranno dai canali W e Z mentre gli altri saranno vuoti. In termini di logica abbiamo che per $A = 1$ e $B = 1$, $W = 1$, $X = 0$, $Y = 0$ and $Z = 1$. Concludiamo che se vogliamo che la porta agisca come un AND basterà osservare l'output in W o Z. Gli output in X equivalgono alla porta $B \wedge \overline{A}$ mentre quelli in Y a $\overline{B} \wedge A$. I risultati sono riportati in tabella 4.

A	B	$W = A \wedge B$	$X = B \wedge \overline{A}$	$Y = \overline{B} \wedge A$	$Z = A \wedge B$
0	0	0	0	0	0
1	0	0	0	1	0
0	1	0	1	0	0
1	1	1	0	0	1

Table 4: Tabella delle verità per la porta fondamentale di un computer a palle di biliardo.

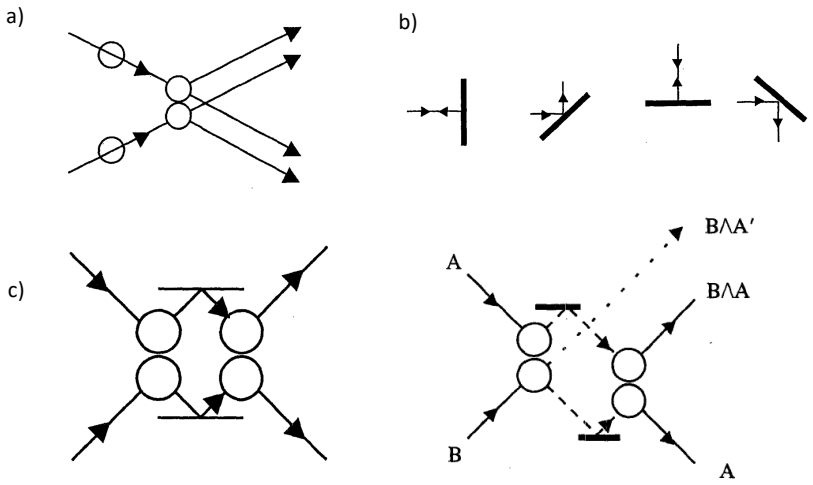

Figure 4: Le porte logiche a) *collision gate* , b) quattro *redirection gates*, c) *crossover gate* e d) *switch gate*.

La porta logica discussa deve essere affiancata da altre operazioni mostrate in Fig. 4. Sono la *collision gate*, una *crossover gate* e quattro *redirection gates* e dalla *switch gate* [**FeynmanLecturesComputation**]. Nella prima due palle da biliardo sono piazzate (e in quiete) al centro e vengono urtate da quelle che definiscono gli input logici. A due palle da biliardo entranti corrispondono quattro uscenti. A livello logico questa porta non è rilevante ma diventa fondamentale a livello fisico per ridirezionare le palle da biliardo. L'operazione di reindirizzamento è completata dalle quattro *redirection gates* mostrate in Fig. 4.

Con questi elementi possiamo costruire porte logiche sempre più complicate. Ad esempio, in Fig. 4 b) è mostrata una *crossover gate* che scambia gli input. Più precisamente, lo stato 10 viene trasformato in 01 (leggendo gli input dall'alto verso il basso) e viceversa. Naturalmente, gli input simmetrici 00 o 11 rimangono gli stessi.

Le porte logiche ottenute fino ad ora sono sufficienti per avere un computer universale. Tuttavia possiamo fare un passo ulteriore e mostrare che è possibile implementare anche la porta di Fredkin e quindi un computer universale e reversibile. In questo caso, l'implementazione è molto più complicata ed è mostrata in figura 5.

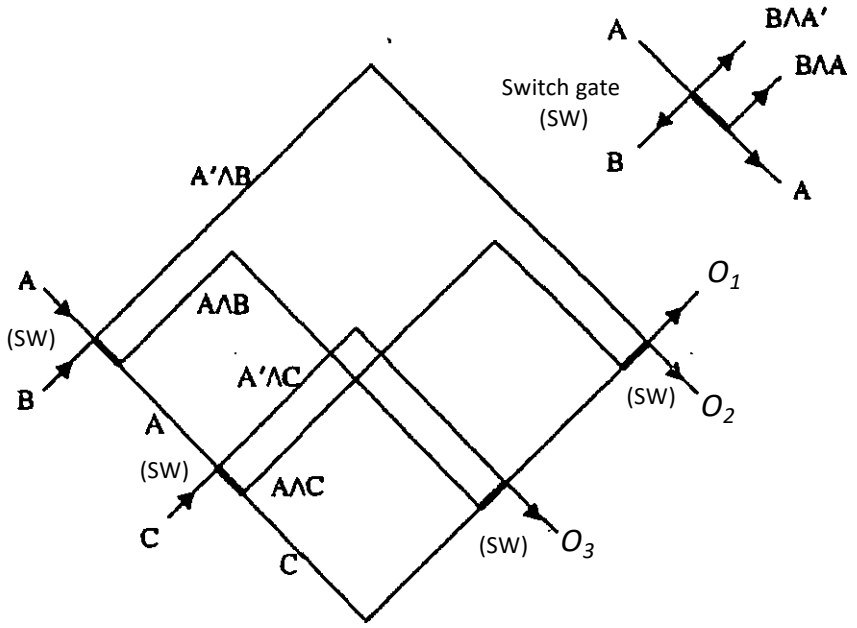

Figure 5: Implementazione della porta di Fredkin sul un computer a palle di biliardo [**FeynmanLecturesComputation**].

IV — COMPUTER CON DNA

Fino a questo punto abbiamo visto come, in linea di principio, sia possibile costruire un computer utilizzando un processo fisico inusuale (l'urto fra palle da biliardo). Questo suggerisce che altri sistemi possano essere usati per costruire dei computer e per fare calcolo.

In questa contesto, una delle scoperte più recenti è che anche i sistemi biologici possono svolgere calcoli. Riflettendo sulle radici stesse della biologia molecolare questo dovrebbe sembrare più naturale anche se comunque stupefacente.

La biologia e l'evoluzione degli esseri viventi è basata sulla conservazione e la trasmissione del patrimonio genetico. La conservazione e la trasmissione di informazione sono alla base del funzionamento dei computer e, in termini più astratti, delle macchine di Turing.

Le idee e la discussione che segue sono frutto della ricerca di Leonard M. Adleman [**Adleman1994**, **Adleman1998**]. Adleman è un informatico noto per il codice crittografico a chiave pubblica più usato al mondo. Infatti l'algoritmo sistema RSA prende il nome dai sui inventori: Ron Rivest, Adi Shamir e Leonard Adleman.

1.4.1 Breve compendio di biologia molecolare

Tutta l'informazione genetica è immagazzinata nel DNA (*Deoxyribonucleic acid* o acido desossiribonucleico). Il DNA si presenta come una doppia elica (Fig. 6). Le eliche del DNA rappresentano solo lo scheletro mentre l'informazione è im-

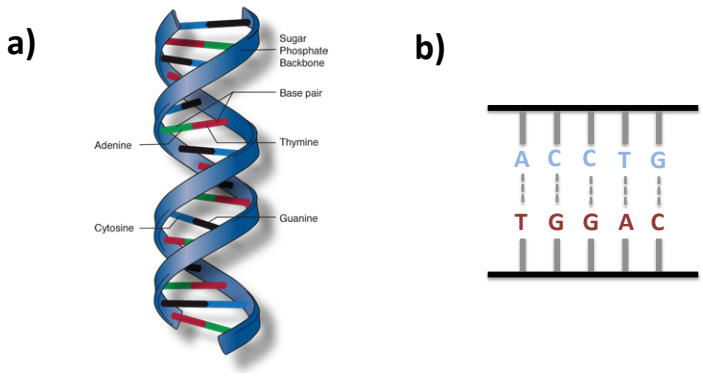

Figure 6: a) Rappresentazione tridimensionale della doppia elica del DNA. b) Schematizzazione della doppia elica del DNA. Lo scheletro è rappresentato dal segmento nero. Ad esso sono collegate le basi azotate. La sequenza rossa della seconda elica è la complementare di quella blu.

magazzinata in quattro basi azotate: Adenina (A), Citosina (C), Timina (T) e Guanina (G). La sequenza di queste basi azotate rappresenta l'informazione genetica delle diverse specie e individui. Ad esempio, l'informazione sul colore degli occhi potrebbe essere codificata nella sequenza ACCTGAA....TTGC. A livello informatico quindi l'informazione codificata nel DNA usa quattro stati; invece degli usuali bit 0 e 1 potremmo associare le basi azotate a dei numeri interi A=0, B=1, C=2 e D=3 [1].

Questa sequenza si trova su una delle eliche mentre sull'altra elica si trova la sequenza detta complementare. Data una sequenza di basi azotate, la sua sequenza complementare si ottiene scambiano A↔T e C↔G come mostrato in figura 6 [2].

La principale caratteristica del DNA è che si può duplicare. Questo avviene tramite un enzima che si chiama DNA polimerasi. Il processo è il seguente. Quando la doppia elica si apre, la DNA polimerasi passa un'elica leggendo le basi azotate. Quando la DNA polimerasi incontra una sequenza di attivazione (detto *primer*), inizia a duplicare la sequenza di basi azotate leggendole e scrivendo la sequenza complementare.

Altri enzimi importanti sono la DNA-ligasi che prende due sezioni del DNA e le unisce in un'unica catena e la DNA-nucleasi che legge il DNA e quando trova una sequenza di attivazione taglia la molecola in due parti.

Noi non discuteremo nel dettaglio l'uso pratico di questi enzimi ma basta sapere che è possibile tagliare e collegare dei tratti di DNA.

1 Se volessimo codificare la stessa informazione in termini binari dovremmo usare due bit. Ad esempio, potremmo porre A=00, B=10, C=01 e D=11.

2 Il motivo fisico-chimico o biologico per questo è che basi complementari possono sviluppare legami chimici. Quindi quando si trovano affacciate nelle due semi-eliche di DNA questi legami tendono a stabilizzare la struttura.

1.4.2 Problema del cammino Hamiltoniano

Dalla descrizione di come funziona la DNA polimerasi si può notare (è quello che fece Adleman) un'incredibile somiglianza con la macchina di Turing. Questa è la schematizzazione più semplice di modello matematico di computazione.

Questa è costituita da una *testa* e da due nastri. La *testa* può far scorrere i nastri, leggere da uno e copiare sull'altro. Si sa che la macchina di Turing è universale e quindi può essere programmata per compiere qualsiasi operazione logica.

Nello schema biologico che stiamo discutendo la *testa* è costituita dalla DNA polimerasi mentre i nastri sono le eliche di DNA. Da questa analogia è naturale aspettarsi che il DNA (come una macchina di Turing) possa essere un computer biologico universale.

Appurato e capito questo Adleman decise però di implementare e risolvere con il DNA un problema computazionale difficile. Scelse il Problema del Cammino Hamiltoniano (PCH) originalmente proposto da William Rowan Hamilton nel diciannovesimo secolo. Il problema si può impostare nel seguente modo [si veda la figura 7 a)]:
Date una serie di città e dei voli unidirezionali che le connettono, e date una città di partenza P *e una di arrivo* A*, è possibile trovare un cammino che parta da* P *e arrivi ad* A *e passi una e una sola volta per ogni città ?*

Per capire meglio, possiamo fare un esempio. In figura 7 b) è mostrata la rete di quattro città con sei voli. Supponiamo che le città di partenza e di arrivo siano, rispettivamente, Atlanta e Detroit. Vediamo subito che è possibile volare direttamente fra le due città . Ma per risolvere il PCH dobbiamo passare per le altre città una e una sola volta. Nell'esempio in figura 7 b) si vede immediatamente che una soluzione esiste ed è il percorso o cammino Atlanta-Boston-Chicago-Detroit. In modo analogo si può facilmente controllare che se la città di partenza è Detroit (per qualsiasi città di arrivo) non esiste nessun cammino Hamiltoniano.

Nell'esempio discusso è facile dare una risposta sull'esistenza di un cammino Hamiltoniano. Quando però il numero di città e di collegamenti cresce, non esiste nessun algoritmo efficiente capace di risolvere il PCH; nella pratica gli algoritmi si limitano a sondare tutto lo spazio dei possibili cammini fino a trovarne uno Hamiltoniano. Al crescere del numero di città , questa procedura in genere richiede delle risorse che crescono esponenzialmente. Addirittura è stato dimostrato che il PCH è un problema computazionalmente difficile e cade nella categoria dei problemi NP-C (*nondeterministic polynomial time complete*).

Vediamo ora come è possibile risolvere il PCH con il DNA. Prima di tutto è bene definire alcuni concetti che useremo. Con il termine grafo intendiamo l'insieme delle città (dette più in generale *nodi* o *vertici*) e dei voli aerei che le congiungono. Un esempio di grafo è mostrato in figura 7 a). Fatta questa precisazione, un algoritmo per risolvere il PCH può essere il seguente.

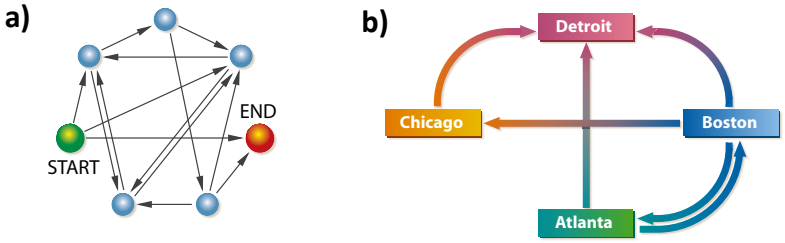

Figure 7: (Sinistra) Rappresentazione tridimensionale della doppia elica del DNA. (Destra) Schematizzazione della doppia elica del DNA. Lo scheletro è rappresentato dal segmento nero. Ad esso sono collegate le basi azotate. La sequenza rossa della seconda elica è la complementare di quella blu.

CITY	DNA NAME	COMPLEMENT
ATLANTA	ACTTGCAG	TGAACGTC
BOSTON	TCGGACTG	AGCCTGAC
CHICAGO	GGCTATGT	CCGATACA
DETROIT	CCGAGCAA	GGCTCGTT

FLIGHT	DNA FLIGHT NUMBER
ATLANTA - BOSTON	GCAGTCGG
ATLANTA - DETROIT	GCAGCCGA
BOSTON - CHICAGO	ACTGGGCT
BOSTON - DETROIT	ACTGCCGA
BOSTON - ATLANTA	ACTGACTT
CHICAGO - DETROIT	ATGTCCGA

SLIM FILMS

Figure 8

Algoritmo:
Dato un grafo a n vertici

1. Genera un insieme di cammini casuali sul grafo.
2. Per ogni cammino dell'insieme
   a) Controlla se il cammino parte e finisce con i vertici (ovvero città) giusti. Se non lo fa rimuovi dall'insieme.
   b) Controlla se il cammino passa per n vertici. Se non lo fa rimuovi dall'insieme.
   c) Per ogni vertice, controlla se il cammino passa per quel vertice. Se non lo fa rimuovi dall'insieme.
   d) Se l'insieme non è vuoto, c'è un cammino Hamiltoniano. Se l'insieme è vuoto, non c'è un cammino Hamiltoniano.

Questo algoritmo non è sicuramente il più efficiente ma ha il vantaggio di essere facilmente implementabile con il DNA. Come è possibile implementarlo con il DNA?

Prima di tutto è necessario codificare l'informazione nelle basi azotate. In particolare, ogni città sarà identificata da una stringa di otto basi azotate come mostrato in Fig. 8. Ad esempio, possiamo associare la città di Atlanta alla stringa ACTT-GCGA. Possiamo considerare le prime quattro basi azotate ACTT come il "nome" della città (per quanto questo possa avere significato) e le quattro finali GCGA come il "cognome" della città . Un'altra importante osservazione è che esisterà anche il complemento TGAA-CGTC che contiene esattamente la stessa informazione. Questo è generato e presente in tutte le soluzioni di DNA.

I voli aerei saranno codificati da un'altra stringa di otto basi azotate. Le prime quattro contengono il "cognome" della città di partenza e le ultime quattro il "nome" della città di arrivo. Ad esempio, il volo Atlanta-Boston sarà codificato dalla stringa GCGA ("cognome" di Atlanta) e TCGG ("nome" di Boston); ovvero GCGA-TCGG.

Possiamo ora capire come il nostro algoritmo compie il primo passo e genera i diversi cammini. Quello che dobbiamo fare è costruire le stringhe associate alle città e ai voli e poi, tramite un processo chimico, farle reagire. In questo modo abbiamo implementato il punto 1 dell'algoritmo (*Genera un insieme di cammini casuali sul grafo*).

Punto 2.a)

Per implementare il punto 2.a) e selezionare i cammini che iniziano per Atlanta e finiscono per Detroit, si può usare la DNA polimerasi e la *polymerase chain reaction* (PRC). Questa tecnica permette di duplicare una parte di DNA compresa fra una sequenza di "inizio" e una sequenza di "fine" assegnate da noi. Nel nostro caso, la sequenza di inizio è identificata da GCAG (per Atlanta) e quella finale da GGCT (per Detroit). La sequenza iniziale impone la DNA polimerasi di iniziare a copiare il DNA fino a che non trova la sequenza di fine. In questo modo, duplichiamo solo i cammini che iniziano per Atlanta e finiscono per Detroit. Quelli che non soddisfano le condizioni sull'inizio e la fine della catena non verranno duplicati e diventeranno quindi irrilevanti nell'esperimento. Si noti comunque che in questo modo duplichiamo anche cammini non Hamiltoniani (ed esempio, le sequenze Atlanta-Detroit o Atlanta-Boston-Detroit).

Punto 2.b)

Nel punto 2.b) dell'algoritmo è necessario selezionare solo i cammini che passano per n vertici. Questo può essere fatto sfruttando il fatto che il peso della stringa di DNA dipende dal numero di vertici che la compongono. Se il DNA è posto in una placca di gel e gli si applica della corrente elettrica, le molecole di DNA cariche si sposteranno (DNA elettroforesi). La velocità di spostamento dipende dal peso e quindi dal numero di vertici che compongono il cammino. Quindi, se vogliamo selezionare solo quelli con n vertici, basta usare la tecnica del DNA elettroforesi, in modo da separare i cammini con diversi nodi e recuperare dal gel solo quelli desiderati. Questi verranno pui duplicati con altre tecniche in modo da avere campioni con un numero di molecole di DNA elevato.

***Punto** 2.c)*

Per controllare se i cammini passano per tutti i vertici (punto 2.c) è necessaria una procedura leggermente più complessa. Supponiamo di voler selezionare solo i cammini che passano per la città di Boston. In questo caso, usiamo delle microsfere di ferro a cui viene attaccata il complemento del nome della città di Boston (AGCC). Stimolando la separazione della doppia elica di DNA, le stringhe che contengono il nome Boston si attaccheranno alla stringa AGCC e quindi alla microsfera di ferro. A questo punto, usando un magnete o campo elettrico, è possibile attrarre e spostare le microsfere con i cammini selezionati. In questo modo, possiamo separarli dagli altri. Ripetendo questa procedura per tutte le città intermedie siamo in grado di selezionare solo i cammini che passano per tutte le città .

***Punto** 2.d)*

L'ultimo punto consta semplicemente nel controllo se l'insieme dei cammini selezionati è vuoto o no. Questo può essere facilitato, ad esempio, usando la PCR che aumenta il numero di eventuali stringhe di DNA presenti nella provetta. Se la provetta non è vuota, siamo sicuri che esiste un cammino Hamiltoniano. Si noti però che a questo livello non abbiamo informazione diretta sul cammino che risolve il PCH. Per averla dovremmo sapere la sequenza delle basi azotate ma questo richiede procedure più complicate a livello di laboratorio. In altre parole, abbiamo risolto il problema decisionale ma non trovato il cammino Hamiltoniano.

V – PROBLEMI NEL SIMULARE SISTEMI QUANTISTICI E COMPUTER QUANTISTICI (FEYNMANN)

[FeynmanLecturesComputation]