INTRODUZIONE ALLE RETI NEURALI ARTIFICIALI

Marco Gori

Nonostante gli straordinari successi dell'elaborazione dell'informazione, che stanno esercitando un impatto di portata storica nella vita quotidiana, competenze percettive quali localizzare un oggetto in una scena, riconoscere la voce in ordinarie condizioni reali, prendere decisioni basate sul "senso comune", risultano ancora compiti estremamente difficili per le macchine. Nel seguito, viene presentato un quadro generale dell'elaborazione neurale: successi, fallimenti e prospettive applicative.

1. IL MONDO SOTTO-SIMBOLICO

Gli odierni sistemi di elaborazione dell'informazione hanno compiuto prodigi che sono sotto gli occhi di tutti. Le macchine hanno automatizzato perfettamente processi considerati tipicamente di pertinenza umana, quali recuperare informazione in un archivio ed eseguire calcoli. Con l'intelligenza artificiale si sono spinte verso l'automazione del ragionamento simbolico, fino ai sistemi esperti, in grado di modellare e rendere fruibile la conoscenza di esperti in specifici settori. Ma nonostante i formidabili risultati conseguiti nell'automazione di alcuni processi intelligenti, soprattutto di alto livello, le macchine offrono ancora un comportamento piuttosto primitivo e incomparabile con l'uomo nella simulazione della maggioranza dei processi percettivi. La difficoltà di automatizzare tali processi è spesso trascurata per il fatto che già nel mondo animale sono presenti capacità percettive talvolta straordinarie. Tali capacità, sviluppate in secoli di processi evolutivi, risultano difficili da replicare usando i modelli di calcolo simbolico alla base degli attuali elaboratori.

Si consideri, per esempio, il problema della descrizione dell'informazione presente in un'immagine che richiede la localizzazione e il riconoscimento di oggetti significativi per l'uomo. Tale processo richiede la capacità di segmentazione che, tuttavia, non può aver luogo solo con operazioni di basso livello basate, per esempio, sul rilievo di variazioni di luminosità. La segmentazione in una scena non può prescindere da processi cognitivi in grado di esibire competenza sugli oggetti e, più generalmente, del mondo oggetto dell'elaborazione. Inoltre, soprattutto nel mondo tridimensionale, gli oggetti si propongono con molteplici viste e, di nuovo, la loro percezione, sembra richiedere modelli ben più sofisticati di semplici comparazioni con oggetti di un dizionario. L'analisi attenta della nozione di similarità di oggetti o quotidiane acquisizioni quali, per esempio, "la facciata di una casa" (Figura 1) indicano lo sviluppo di competenze che non sono basate su elaborazione simbolica e ragionamenti qualitativi di dif-

ficile formalizzazione. L'esempio indicato in figura 1 illustra, inoltre, un livello di sofisticazione dell'analisi umana delle scene che risulta molto difficile da trasferire alle macchine: con uno sguardo attento si riconosce facilmente che le case nella scena sono, in realtà, miniature. La foto ritrae un paesaggio di un piccolo paese della montagna pistoiese in occasione del Natale, ma discernere il mondo artificiale miniaturizzato dal mondo reale è un compito realmente arduo per le macchine. Il problema del riconoscimento automatico della voce offre difficoltà simili. Il segnale vocale rappresentato in figura 2, assieme al suo spettrogramma, illustra alcuni aspetti della difficoltà del problema. Si presenta di nuovo il problema della segmentazione; anche in questo caso non si può fare affidamento su elementari elaborazioni di basso livello del segnale, quale per esempio il controllo del livello per separare le parole. Infatti, basta per esempio la presenza di occlusive sorde all'interno di una parola per il fallimento della segmentazione; la parola *compute* (si veda, a tal proposito, il segnale corrispondente a una sua pronuncia in figura 2) verrebbe, infatti, spezzata in due parti per la presenza dell'occlusiva sorda "*p*". Per via dell'enorme variabilità dovuta alla velocità di pronuncia, alla prosodia, al parlatore e a varie altre condizioni di rumore, le parole, inoltre, non sono facilmente rappresentabili mediante un dizionario di centroidi, ovvero di "istanze medie" di riferimento delle parole del dizionario.

I problemi menzionati si presentano ormai in molteplici applicazioni nelle quali l'elaborazione di informazione multimediale assume un ruolo sempre più rilevante. Si pensi, per esempio, alla navigazione autonoma di un *robot*, ai *data base* visuali, alla definizione di interfacce personalizzate, alla gestione di immagini di documenti, alla concezione di modelli per l'estrazione di informazione dal *web*. I problemi menzionati, assieme ad altri, hanno in comune il fatto che non sembrano naturalmente affrontabili mediante soluzioni basate su elaborazione simbolica. L'informazione da elaborare si presenta con una codifica a cui non è semplice attaccare significato. Il prodigio della scienza dei calcolatori proviene tipicamente dalla conoscenza del significato degli ingressi e dalla loro conseguente elaborazione mediante algoritmi. Non è, tuttavia, sempre possibile, o comunque verosimile, associare agli ingressi un significato e una conseguente caratterizzazione simbolica[1]. Serve, dunque, dotare i calcolatori di processi computazionali che non siano necessariamente basati sulla metafora dell'algoritmo, secondo cui la soluzione di un problema avviene mediante un processo costruttivo atto ad esplicitare, simbolicamente, le elaborazioni sugli ingressi caratterizzati simbolicamente. Per alcuni problemi, questo approccio non appare naturale e risulta di difficile, se non impossibile, formalizzazione.

FIGURA 1

Un paesaggio della montagna pistoiese durante il Natale

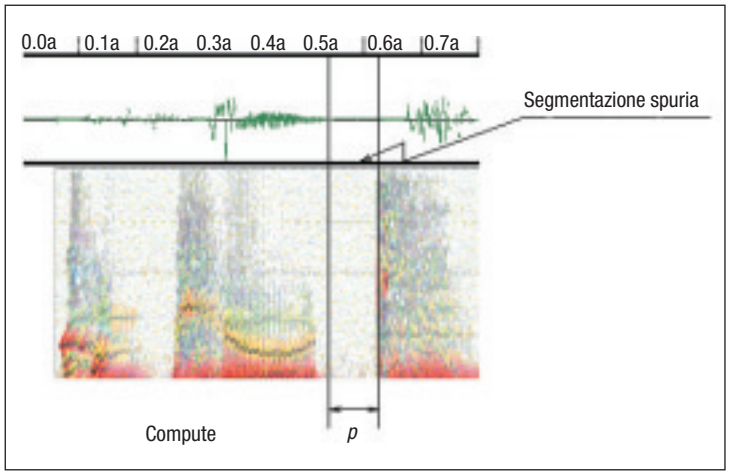


FIGURA 2

Segnale e spettrogramma relativo alla pronuncia del verbo "to compute"

[1] Nelle scienze cognitive questo problema è noto come *the symbol ground problem*.

2. LA METAFORA NEUROBIOLOGICA

Allo stato attuale, a differenza delle macchine, l'uomo è un ottimo esempio di "sistema" in grado di elaborare informazione sotto-simboliche. Tali elaborazioni, come ogni altro processo cognitivo, hanno sede nel cervello, una complessa struttura neurobiologica, attualmente decifrata in modo piuttosto accurato per quanto riguarda gli aspetti anatomici. È noto che c'è un "mattone elementare" che caratterizza tutte le strutture cerebrali, una cellula, denominata *neurone*, che è sede di processi elettrochimici responsabili per la generazione di campi elettromagnetici. Come è illustrato in figura 3, i neuroni sono composti da un corpo detto *soma* e da due tipi di diramazioni: i dentriti e il cilindrasse o assone.

Nel cervello umano sono presenti tipicamente oltre 100 miliardi di neuroni, ciascuno interconnesso a circa altri 10.000. Nelle interconnessioni ha luogo la *sinapsi*, un processo elettrochimico atto a rinforzare o inibire l'interazione cellulare. I segnali rilevabili hanno un potenziale dell'ordine di alcune decine di millVolt e si presentano come treni di impulsi con frequenza intorno ai 100 Hz, con opportune modulazioni. Sono noti modelli sofisticati che esprimono il potenziale della cella (attivazione) in funzione del potenziale delle celle interconnesse. È opinione condivisa da ricercatori nel mondo delle scienze cognitive che i segnali elettrici presenti nei neuroni siano alla base dell'elaborazione dell'informazione a livello cerebrale. Le capacità cognitive sarebbero, dunque, in relazione all'elaborazione dei segnali presenti nei neuroni. Inoltre, c'è evidenza sperimentale per sostenere che la struttura cerebrale e le sinapsi siano influenzate dalla vita degli individui, dalle loro esperienze, dall'apprendimento di compiti specifici. È il particolare *pattern* di interconnessioni e la forza delle connessioni sinaptiche che definisce le proprietà funzionali di una particolare porzione del cervello. Si è, infatti, verificato sperimentalmente che le funzioni cognitive risiedono in particolari zone e che tali funzioni possono essere perdute a seguito della "rottura" dei legami sinaptici ed

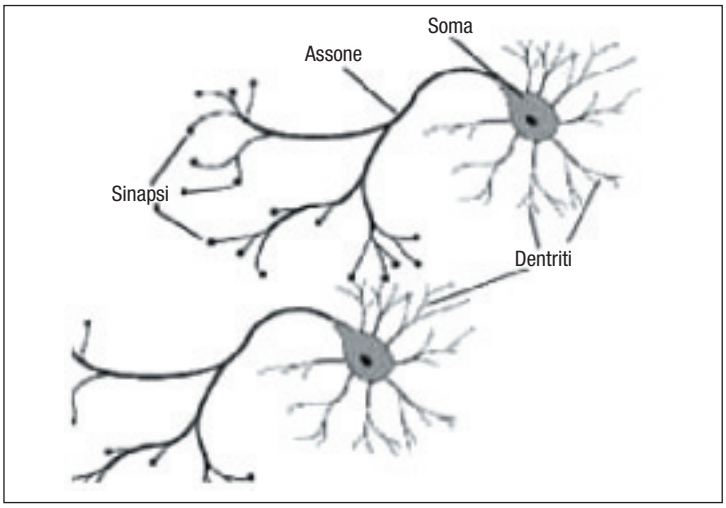


FIGURA 3

Neurone e sua struttura cellulare con soma, dentriti e connessioni sinaptiche

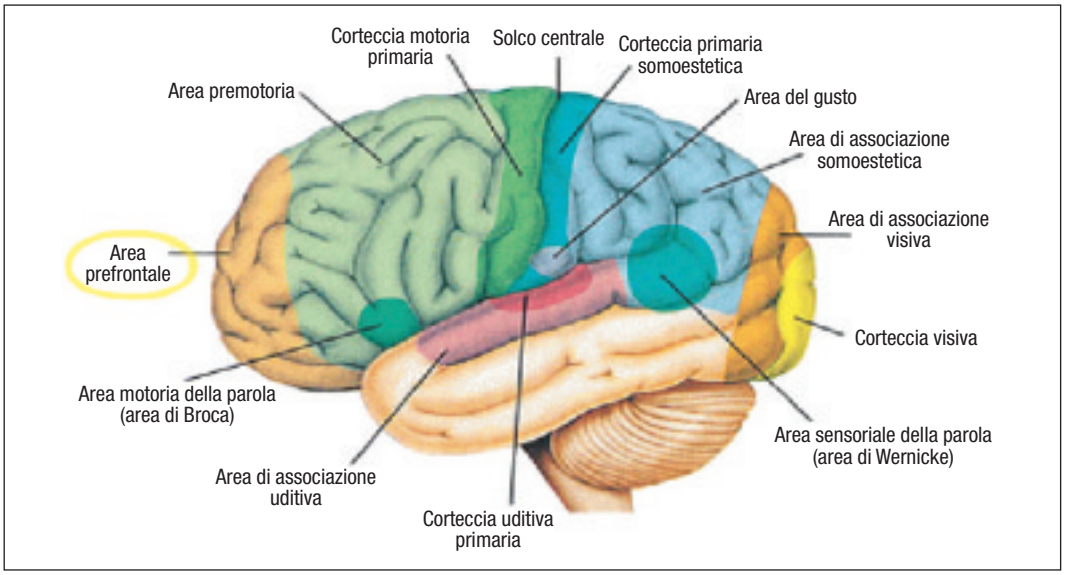


FIGURA 4

Organizzazione del cervello umano e sua localizzazione funzionale [13]

eventualmente recuperate, almeno in parte, con successivi processi di apprendimento atti a instaurare nuovi pattern di interconnessione sinaptica (Figura 4).
Dato che la struttura cerebrale e il comportamento elettromagnetico delle singole celle neuronali sono noti, i ricercatori si sono ben preso chiesti se si possono operare induzioni sui comportamenti collettivi delle cellule neuronali, e dunque del cervello umano, e se si possono trarre utili suggerimenti e ispirazioni per la costruzione di macchine in grado di replicare compiti connotati da una forte componente di elaborazione sotto-simbolica, attualmente di difficile soluzione per i calcolatori. Il lavoro di McCulloch & Pitts [8] è forse il primo significativo passo in questa direzione, la prima analisi completa, soprattutto dal punto di vista formale, che fa intuire come semplici unità con sinapsi eccitatorie e inibitorie e con apposita soglia siano in grado, in virtù di un processo collettivo, di rappresentare complesse proposizioni. E questo sembra indurli a un certo ottimismo anche per le possibili implicazioni sulla comprensione dei processi cognitivi umani: "*Mind no longer goes more ghostly than a ghost*". Tuttavia il lavoro di McCulluch e Pitt sembra essere stato più rilevante per gli sviluppi nel settore dei calcolatori che non delle scienze cognitive. Carpire i segreti della mente dall'osservazione dell'attivazione cerebrale è una sfida affascinante, ma questo problema di *reverse engineering* sembra essere terribilmente intrappolato nella complessità del sistema neuronale umano. Si tratta sostanzialmente di indurre regolarità e leggi dall'osservazione, come in altri settori delle scienze. Questo problema possiede, tuttavia, un'infinità di sfaccettature e, soprattutto, richiede un processo di induzione che sembra inerentemente intrappolato nella complessità dei sistemi dinamici oggetto dello studio. Inferire regole dagli esempi sembra essere difficile anche in casi elementari; si pensi, a titolo di esempio, al problema dell'inferenza induttiva di grammatiche, che consiste nel determinare la grammatica che genera un linguaggio presentato mediante esempi. Sfortunatamente, questo problema è intrattabile perfino in caso di semplici grammatiche [3].
Era, tuttavia, ben chiaro ai padri dell'informatica che non è necessaria una perfetta emulazione dei processi neurobiologici per l'emergenza di capacità cognitive. Molti modelli connessionistici sono, infatti, solo ispirati dal paradigma biologico a livello di unità neuronale e si basano sulla struttura indicata in figura 3, dove si eredita il principio che l'attivazione neuronale (potenziale associato all'unità) è soggetta a eccitazioni e inibizioni dalle unità connesse. In particolare, l'attivazione dell'unità *i* dipende dall'attivazione della generica unità *j* mediante un parametro associato alla connessione tra le due unità, che modella il principio elettrochimico della sinapsi. In seguito, sarà illustrato come l'utilizzo di modelli di calcolo basati su reti neurali artificiali sia in grado di esibire quello che Lotfi Zadeh ha definito *softcomputing* secondo cui il requisito "trova sempre la soluzione esatta" diventa "trova spesso una soluzione approssimata".
La ricerca sulle reti neurali artificiali si è evoluta attraverso alterne vicende. Sin dagli albori dell'informatica, l'elaborazione basata su algoritmi e i modelli neurali centrati sull'apprendimento da esempi si sono sviluppati in parallelo. Verso la fine degli anni '60, Marvin Minsky e Simon Paper [10] pubblicano "Perceptrons", un libro che analizza con grande lucidità ed elegante formalizzazione le capacità computazionali del percettrone di Rosenblatt. La comunità scientifica recepisce principalmente l'analisi critica del percettrone e segue una fase di stagnazione che si protrae fino agli inizi degli anni '80. L'interesse rifiorisce, in particolare, per i lavori di Hopfield e del *Parallel Distributed Research Center* sulle reti neurali multistrato con l'algoritmo di apprendimento *Backpropagation*. Altre tappe importanti della ricerca nel settore sono riassunte nella tabella 1.

3. ARCHITETTURE NEURALI

Le neuroscienze hanno permesso di stabilire che la struttura cerebrale è caratterizzata dalla presenza di cellule neuronali con comportamenti vari e, soprattutto, da pattern di interconnessioni neuronali diversi a secon-

I era	Eventi significativi
1943	McCulloch and Pitts, formalizzazione del neurone artificiale [8]
1949	D. Hebb e l'apprendimento per auto-organizzazione [6]
1956	"Dartmouth Summer Research Project on AI" con (Minsky, McCarty, Rochester, Shannon)
1960	Widrow: ADALINE [14]
1962	Il perceptron di Rosenblatt [11]
1969	"Perceptrons", Minsky & Papert (edizione espansa [10])
70s	Periodo "buio": degni di nota gli associatori di Anderson, i modelli per apprendimento senza supervisione di Kohonen, gli studi di Grossberg
II era	**Eventi significativi**
1982	Reti di Hopfield: memorie associative e soluzione di problemi [7]
1986	PDP e diffusione di Backpropagation [12]
1987	La prima conferenza significativa dell'IEEE a San Diego (II era)
1989	I chip neurali si affacciano sul mercato: *Analog VLSI and Neural Systems* [9]
1990	J. Pollack e le reti neurali che elaborano strutture dati
1994	Prima Conferenza Mondiale sull'Intelligenza Computazionale (Orlando)
1994	Nasce il progetto NeuroCOLT (*Computational Learning Theory*)
2001	L'IEEE approva la creazione della "*Neural Networks Society*"

TABELLA 1
Alcuni eventi significativi che hanno marcato la storia delle reti neurali artificiali

do del compito cognitivo. Per i modelli artificiali è stata seguita una metafora simile: sono stati studiati diversi tipi di neuroni e diverse architetture associandovi le modalità di elaborazione concepite per implementare un determinato compito cognitivo. In figura 5 sono illustrati i due tipici neuroni artificiali che risultano, attualmente, i più interessanti dal punto di vista applicativo. Il primo, denominato neurone sigmoidale, è l'evoluzione del percettrone di Rosenblatt [11], in cui il processo di decisione ha luogo mediante una funzione a gradino, invece, della funzione sigmoidale illustrata. Questi neuroni si eccitano per punti che sono situati sopra il piano di separazione, dove si annulla l'attivazione, e si inibiscono per punti situati al di sotto. Le unità del secondo tipo si eccitano per punti contigui al centro (w_{ia}; w_{ib}; w_{ic}) e si inibiscono quando ci si allontana, con una velocità commisurata al parametro radiale σ_i.
È interessante notare che nei neuroni biologici l'attivazione ha la tipica forma di "un treno" di impulsi. La dipendenza dalle connessioni sinaptiche illustrata in figura 5 ha effettivamente una genesi biologica, ma l'attivazione deve interpretarsi come la frequenza di treni di impulsi più che come un valore assoluto di potenziale. Sono stati anche studiati modelli formali, denominati *spiking neurons* che producono, a differenza dei due precedenti tipi di neuroni, treni di impulsi come i neuroni biologici. Gli studi sulle aggregazioni di tali neuroni e sui modelli di apprendimento costituiscono un interessante settore di ricerca che, tuttavia, non ha, ad oggi, prodotto risultati applicativi paragonabili a quelli conseguiti con i modelli semplificati in figura 5.

3.1. I percettroni

I singoli neuroni descritti in precedenza possono essere utilizzati per il calcolo di semplici predicati, ma non possono certamente soddisfare molte significative esigenze reali. Per esempio, è immediato verificare che i neuroni sigmoidali non possono calcolare tutte le funzioni booleane di due variabili. In particolare, le funzioni $x_1 \otimes x_2$ e $\overline{x_1 \otimes x_2}$ non sono linearmente separabili, mentre, come è illustrato in figura 6 **A**, lo sono tutte le 14 altre

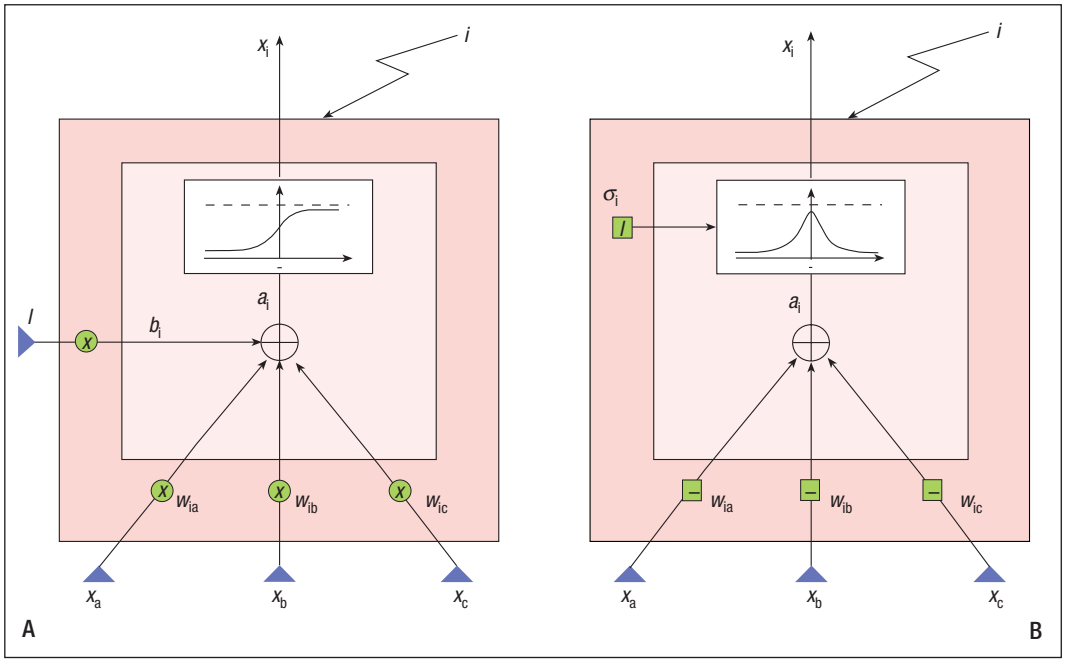
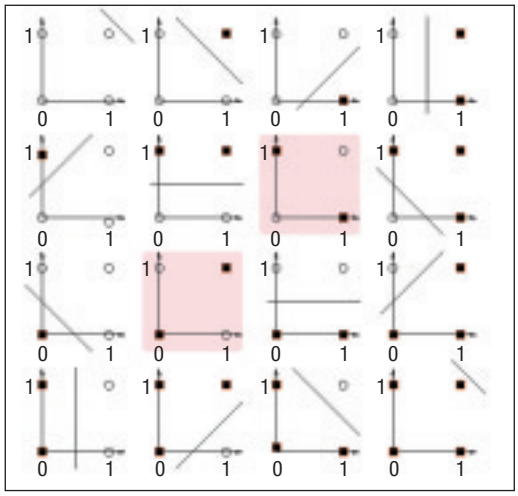


FIGURA 5
Due classici esempi di neuroni artificiali

rimanenti. Più complesso è stabilire il comportamento del neurone di Rosenblatt nel caso di interessanti compiti cognitivi, quali il riconoscimento di forme.
Nella figura 6 **B** l'immagine è pre-elaborata mediante l'operatore di *pre-processing* $\Phi(x)$ che fornisce *feature* significative. Tale condizione impone, ovviamente, che il dominio dell'operatore sia limitato in modo da stabilire feature locali presenti nell'immagine indipendentemente da traslazioni e rotazioni. Indipendentemente dalla scelta dell'operatore, Minsky & Papert [10] hanno dimostrato che alcuni predicati topologici importanti, quali stabilire la connessione di una figura, non possono essere calcolati. Nella seconda metà degli anni '80, grazie soprattutto agli studi del *Parallel Distributed Processing research group*, sono state studiate, in modo sistematico, architetture neurali con architettura a grafo aciclico, in cui è definito un ordinamento parziale sui vertici. In tali architetture, un neurone può avere per genitori sia unità che ingressi (per esempio, il nodo 4, in Figura 7 **A**). Lo schema di calcolo si basa sulla "propagazione in avanti" delle attivazioni dei neuroni seguendo l'ordinamento parziale del grafo aciclico.
Per reti multi-strato lo schema di calcolo si riduce a una *pipe* sui livelli. Tali architetture erano, in realtà, già state concepite agli inizi



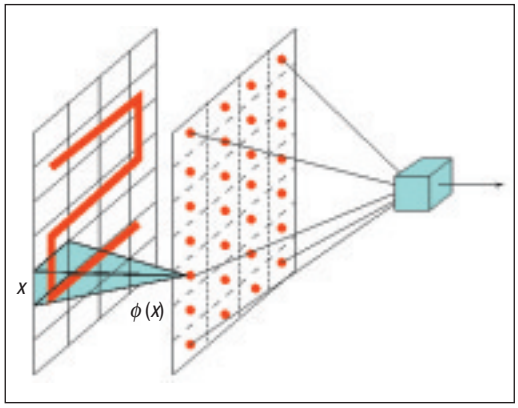


FIGURA 6
*Due esempi delle limitazioni del percettrone. **A** Funzioni booleane a due variabili; **B** Un percettrone con il compito di stabilire se la figura elaborata è connessa*

degli anni '60 e non differiscono sostanzialmente dallo schema illustrato in figura 6 **B**. Tuttavia, in quegli anni, si faceva riferimento a una sola unità neuronale in cui aveva luogo l'apprendimento e le altre unità erano semplicemente il risultato di una pre-elaborazione definita in modo esplicito, senza apprendimento. Nelle reti neurali di figura 7 le unità sono tutte uguali e sia la precedente elaborazione in avanti che l'apprendimento hanno luogo in modo omogeneo sui neuroni.

È stato dimostrato che le reti neurali *feed-forword* hanno un potere computazionale universale, ovvero sono in grado di calcolare ogni funzione di ragionevole interesse pratico con un numero sufficientemente grande di neuroni. Purtroppo, non sono disponibili concrete indicazioni di progetto, ma la proprietà di computazione universale può essere facilmente compresa almeno in due casi notevoli.

1. Funzioni booleane

È facile rendersi conto che le reti feedforward possono realizzare ogni funzione booleana. Infatti, ogni funzione booleana si può esprimere in prima forma canonica. A titolo di esempio, in figura 8 è illustrata la realizzazione della funzione XOR.

2. Funzioni di appartenenza

Una funzioni di appartenenza è tale che $f_U(\boldsymbol{u}) = 1$ se e solo se $\boldsymbol{u} \in U[f_U(\boldsymbol{u}) = 0 \Leftrightarrow \boldsymbol{u} \in \bar{U}]$.

Per domini convessi (Figura 9) l'uscita si può determinare come AND di opportuni neuroni dello strato nascosto. Per domini non connessi (Figura 10) e/o concavi l'uscita si può determinare come l'OR di opportune unità nascoste (3 strati - eccetto l'ingresso). Si noti che questo metodo si basa su un processo costruttivo e che, dunque, non si può concludere che *servono necessariamente* due strati

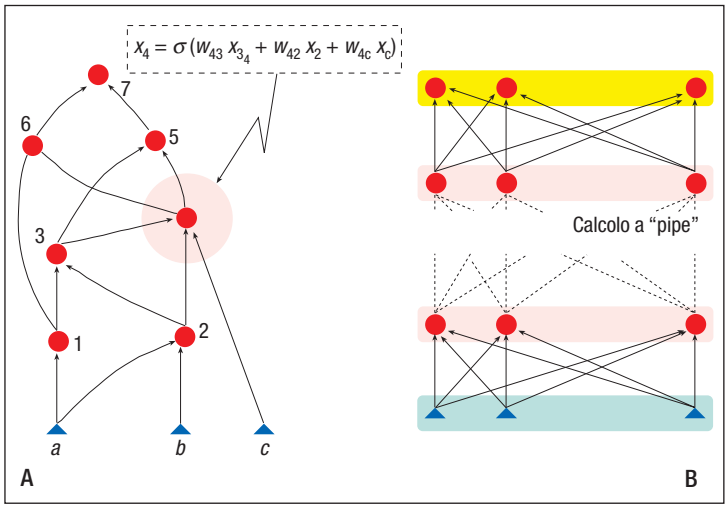


FIGURA 7

A Rete feedforward con struttura a grafo aciclico. B Rete multistrato

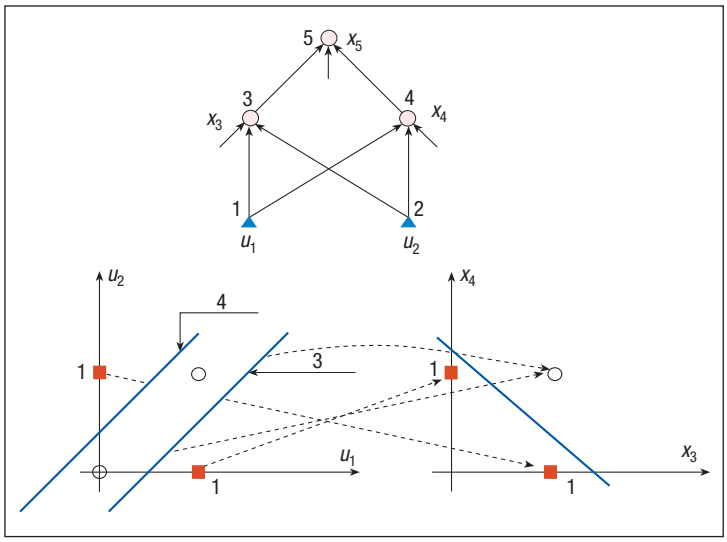


FIGURA 8

Realizzazione mediante percettrone multistrato della funzione booleana XOR

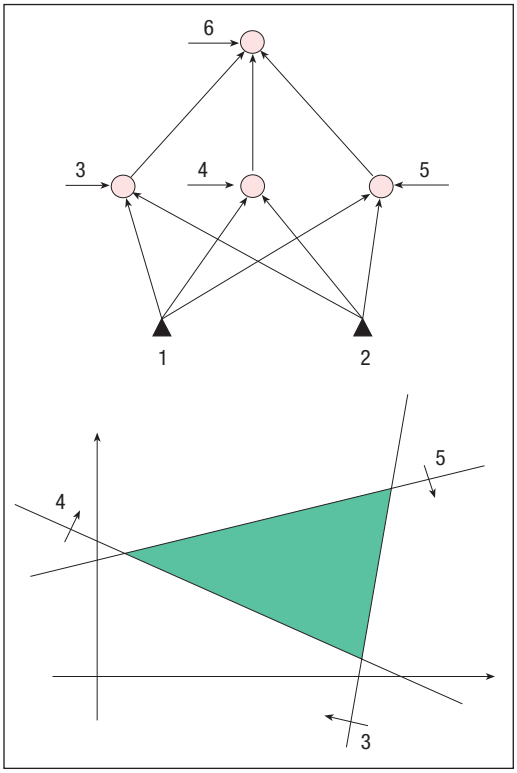


FIGURA 9

Tre unità nascoste che originano i tre iperpiani necessari per definire il dominio convesso

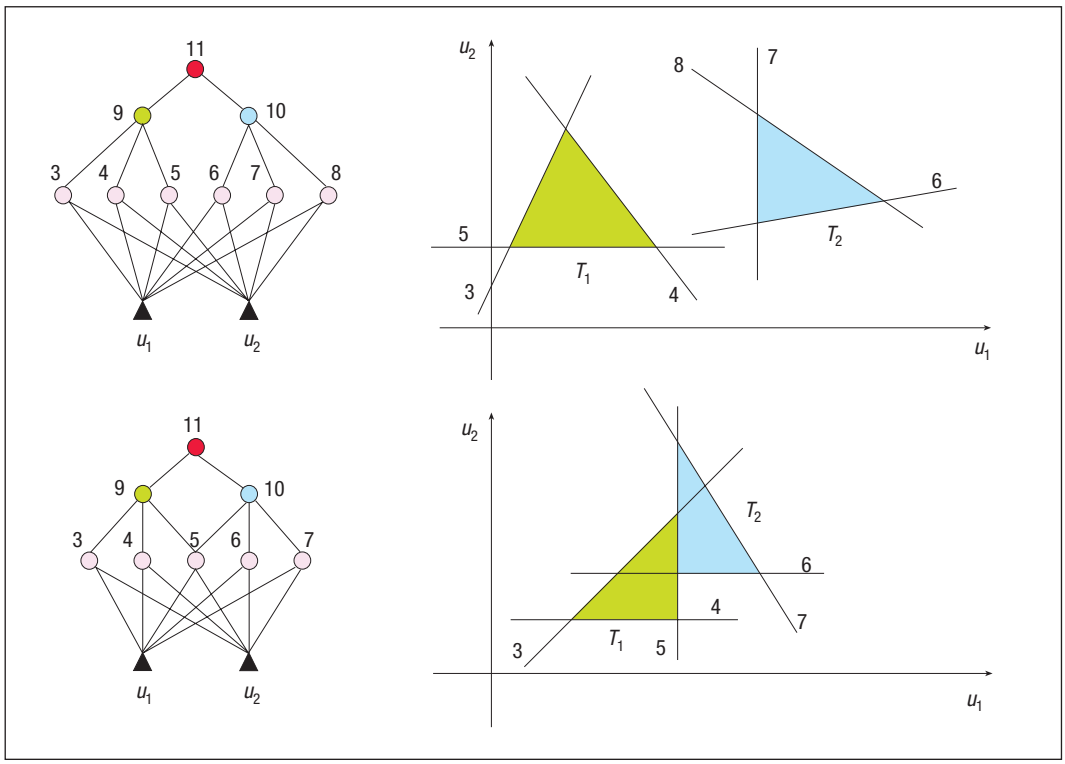


FIGURA 10
Domini non connessi possono essere ottenuti, per esempio, con due strati nascosti

nascosti per il calcolo di tali funzioni di appartenenza. Simili elaborazioni si possono eseguire anche usando neuroni a simmetria radiale di figura 5 **B**.

3.2 Reti neurali ricorsive

Le architetture neurali descritte in precedenza presuppongono schemi di "calcolo in avanti", basati su un ordinamento delle unità. Come per le funzioni booleane, la presenza di cicli conduce a elaborazioni più complesse che coinvolgono sequenze e non singoli pattern. Una tipica situazione in cui risulta naturale un'elaborazione sequenziale è quella del riconoscimento di fonemi illustrata in figura 11.

L'elaborazione è sincronizzata in corrispondenza a ogni *frame*. I neuroni sono ancora del tipo illustrato in figura 5, ma oltre agli ingressi provenienti dal frame corrente, ai neuroni dello strato nascosto afferiscono anche come ingresso il valore delle uscite relative al frame precedente.

Le reti neurali ricorsive presentano strette connessioni con gli automi, ma il calcolo che ha luogo nelle unità neuronali ha natura continua e non discreta. In virtù di tale natura, le reti neurali ricorsive presentano anche una

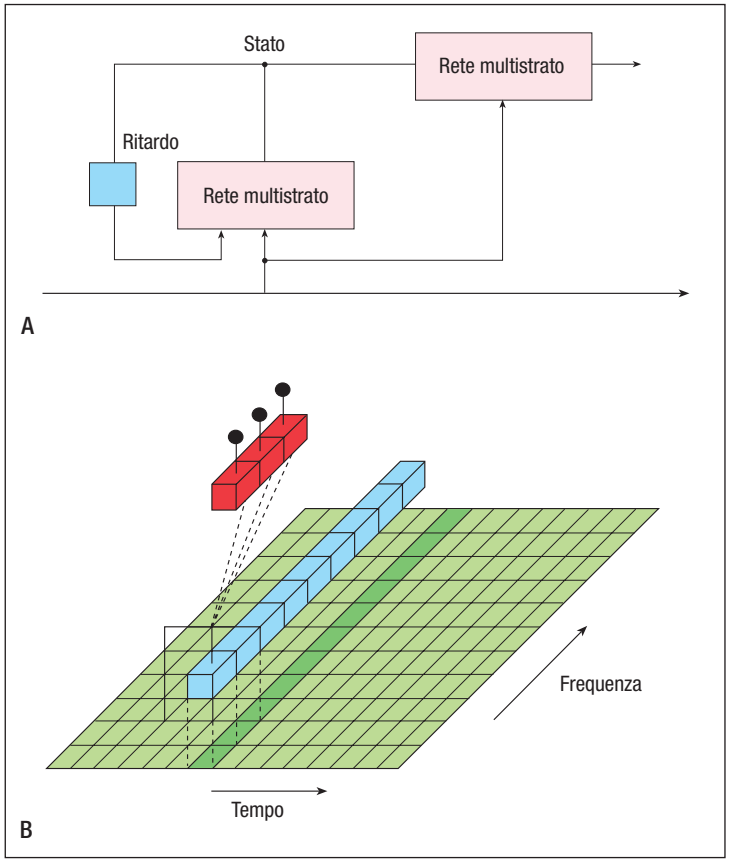


FIGURA 11
A Architettura di una generica rete ricorsiva. B Una rete neurale ricorsiva

forte connessione con i sistemi dinamici lineari, anche se la loro struttura dinamica è orientata a processi decisionali.
Le reti ricorsive non elaborano solo sequenze ma, direttamente, anche grafi i cui nodi contengono un vettore di numeri reali. Si potrebbe sempre ricondurre l'elaborazione di strutture a quella di opportune sequenze. Tuttavia, rappresentando un grafo come una sequenza si nascondono caratteristiche discriminanti per l'apprendimento. Inoltre, si può mostrare che la riduzione a lunghe sequenze derivante dalla codifica di strutture a grafo in stringhe rende il problema dell'apprendimento da esempi più costoso. L'elaborazione su grafi ha luogo estendendo il concetto di calcolo dello stato in un automa a stati finiti dal caso di sequenze a quello di alberi e, più generalmente, di grafi aciclici ordinati [5].
Per le architetture illustrate l'aggiornamento dello stato avviene in modo sincrono rispetto all'alimentazione di un nuovo ingresso della sequenza o della struttura dati. La struttura di una rete neurale ricorsiva può, tuttavia, operare anche elaborazioni sequenziali, mediante aggiornamento dello stato, di ingressi tenuti fissi. L'esempio più classico è quello delle reti di Hopfield, illustrate in figura 12. Si noti che l'uscita di ogni neurone è connessa a tutti gli altri e che non c'è connessione locale. Nell'esempio illustrato in figura un'immagine, che raffigura un pattern corrotto da rumore, viene presentata in ingresso alla rete ricorsiva. L'ingresso è costituito dai *pixel* dell'immagine o, più in generale, da una forma de-campionata a risoluzione più bassa dell'originale. Con opportune scelte dei pesi delle connessioni[2], mantenendo l'ingresso costante, la rete ricorsiva procede ad aggiornamenti delle attivazioni dei neuroni finché, dopo una fase di rilassamento, raggiunge un punto di equilibrio. Com'è illustrato in figura, il punto di equilibrio corrisponde all'immagine filtrata dal rumore. In pratica, una rete di Hopfield con N ingressi, permette in modo affidabile di memorizzare un numero di pattern intorno a $0,15N$ e può, pertanto, essere utilizzata come memoria associativa.

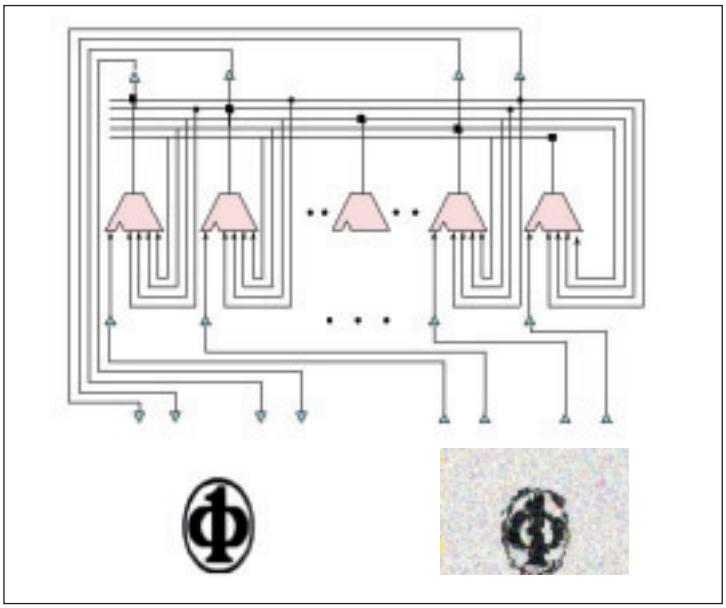


FIGURA 12
Rete di Hopfield utilizzata come memoria associativa per filtrare il rumore

4. APPRENDIMENTO DA ESEMPI

Nelle reti neurali artificiali, le architetture illustrate nel paragrafo precedente, assieme ai corrispondenti schemi computazionali, sono di scarso interesse senza il paradigma centrale dell'apprendimento, che viene ispirato a corrispondente paradigma neurobiliogico. Apprendere in una rete neurale artificiale corrisponde a modificare il valore dei pesi delle connessioni sinaptiche. Tale processo è influenzato dagli esempi che concorrono a sviluppare concetti. I dati e l'interazione con l'ambiente concorrono con diversi protocolli allo sviluppo di competenze cognitive. In particolare, si individuano tre diverse modalità di apprendimento a seconda del ruolo esercitato dal supervisore del concetto: l'apprendimento con supervisione, l'apprendimento con rinforzo e l'apprendimento senza supervisione.

4.1. Protocolli di apprendimento

Nell'apprendimento con supervisione e con rinforzo, la rete neurale deve sviluppare un concetto sulla base delle interazioni con un supervisore, che provvede a istruire la rete, fornendo informazioni sul concetto.
Si consideri, per esempio, il problema della

[2] In particolare, la matrice delle connessioni è simmetrica.

classificazione di insetti illustrato in figura 13. La rete neurale esibisce la sua classificazione e interagisce con il supervisore che può fornire un'informazione completa o parziale sul concetto. Quando l'informazione è parziale si parla di "apprendimento con rinforzo"; tale informazione deve essere utilizzata nell'apprendimento per rinforzare comportamenti corretti e penalizzare quelli che originano errori.

Nell'"apprendimento con supervisione", invece, il supervisore fornisce l'informazione completa sul concetto, definendo, in questo caso, esattamente la classe di appartenenza. L'apprendimento di un concetto non richiede necessariamente l'interazione con un supervisore e può aver luogo anche mediante un'auto-organizzazione degli esempi. Apprendere senza supervisione significa aggregare esempi simili in regioni neuronali topologicamente vicine. In figura 14 è illustrata l'auto-organizzazione di esempi di classi diverse e l'eccitazione dei neuroni spazialmente correlati al concetto. Mentre per i due precedenti protocolli di apprendimento la variazione delle connessioni sinaptiche avviene cercando di ottimizzare l'errore rispetto all'informazione fornita dal supervisore: in questo caso, l'apprendimento è guidato da criteri di "similarità" nei dati.

In generale, i tre protocolli di apprendimento descritti sono formulabili come ottimizzazione di una funzione dei pesi della rete neurale. Nel caso dell'apprendimento con rinforzo e dell'apprendimento con supervisione, per rendere il comportamento della rete neurale conforme alla supervisione occorre minimizzare una funzione di errore che dipende dalla scelta dei pesi e misura l'errore rispetto alle informazioni del supervisore. Nel caso dell'apprendimento senza supervisione, l'auto-organizzazione per similarità dei dati può ancora, generalmente, formularsi come l'ottimizzazione di una funzione di armonia. Il problema di ottimizzare funzioni in grossi spazi è generalmente difficile per la potenziale presenza di minimi locali, che può rendere inefficaci le classiche euristiche di ottimizzazione basate sulla tecnica di massima discesa del gradiente.

Il corretto funzionamento di una neurale sull'insieme di apprendimento non offre, ovviamente, garanzia di un altrettanto soddisfacente funzionamento su altri dati relativi allo stesso concetto, ma non utilizzati nella fase di apprendimento (insieme di test). Inoltre, è evidente che l'architettura della rete neurale gioca un ruolo fondamentale per l'efficienza della fase di apprendimento. Si consideri, ad esempio, il caso delle reti feedforward e il loro comportamento al variare del numero delle unità nascoste. In virtù della loro capacità universale di approssimazione, tali reti possono calcolare ogni concetto. Quando il numero delle unità nascoste cresce, non solo aumenta il potere computazionale, ma si può dimostrare che il problema della presenza dei minimi locali diventa progressivamente meno rilevante. Tuttavia, al crescere

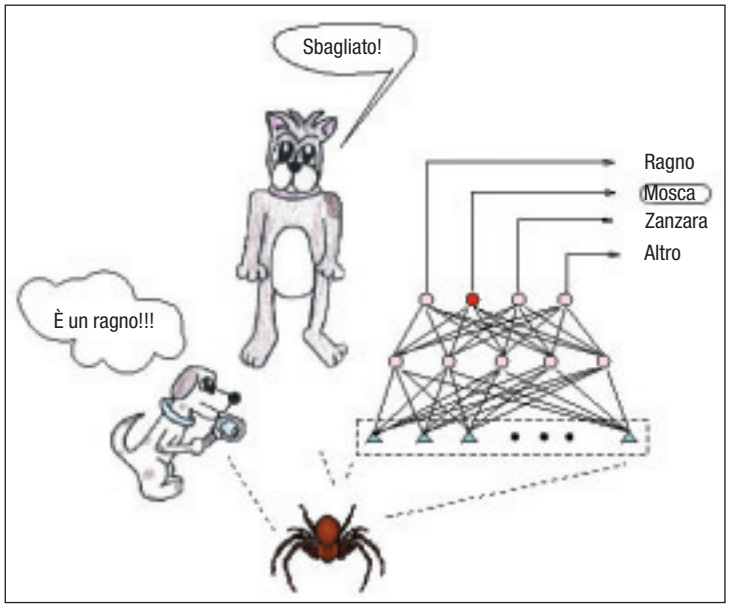


FIGURA 13
Il paradigma di apprendimento con supervisione e con rinforzo

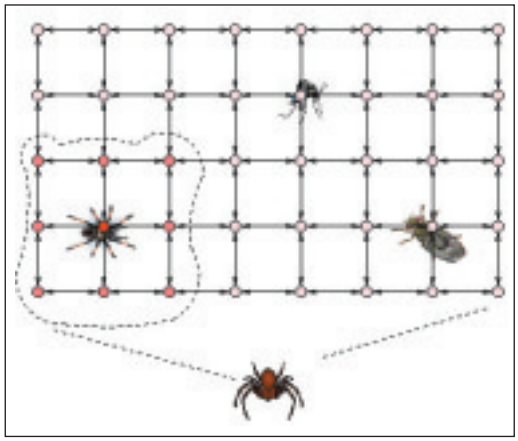


FIGURA 14
Apprendimento senza supervisione

della dimensione della rete la capacità di generalizzare su nuovi esempi tende a diminuire dato che il *fitting* sull'insieme di apprendimento ha luogo in un enorme spazio di parametri vincolati solo da pochi esempi. Questo origina una sorta di principio di indeterminazione dell'apprendimento secondo il quale non è possibile al variare dei pesi della rete neurale ottenere funzioni di errore senza minimi locali ed eccellente generalizzazione per nuovi esempi.

4.2. Algoritmi di apprendimento

La formulazione dell'apprendimento come ottimizzazione della funzione errore rispetto alla supervisione permette di attingere all'enorme letteratura di analisi numerica per la ricerca degli algoritmi più opportuni. Tuttavia, dato che in pratica si opera con reti neurali che possono raggiungere centinaia di migliaia di variabili[3], si restringe di solito l'attenzione all'uso dell'euristica di massima discesa del gradiente che, essendo una tecnica del primo ordine permette di limitare spazio e tempo di calcolo. Algoritmi di apprendimento direttamente basati su tecniche iterative, come il gradiente, prendono il nome di algoritmi di tipo *batch*. In tali algoritmi, la variazione dei pesi avviene solo dopo aver elaborato tutti gli esempi dell'insieme di apprendimento. Si possono, però, concepire algoritmi nei quali i pesi sono aggiornati in corrispondenza della presentazione di ogni esempio (algoritmi di tipo *on-line*).
L'entità di variazione dei pesi in corrispondenza degli esempi può condurre, in questo caso, a enfatizzare il comportamento corretto sugli "ultimi" esempi presentati, dimenticando i vecchi esempi su cui la rete aveva appreso. È evidente che tale entità deve anche essere commisurata alla numerosità dell'insieme di apprendimento. Un aspetto particolarmente rilevante degli algoritmi di apprendimento, che è talvolta sottovalutato, è costituito dalle proprietà di località spaziale e temporale che sono tipicamente gradite per motivi di efficienza computazionale. Si richiede, in sostanza, che lo schema di aggiornamento dei pesi preveda, per ogni neurone, l'uso di informazione disponibile per mezzo delle sole unità che sono direttamente collegate (località spaziale) e che tale informazione sia riferita solo all'istante di tempo precedente (località temporale).
Per reti ricorsive con architettura generica risulta difficile concepire schemi di apprendimento in grado di coniugare entrambe queste proprietà, mentre questo è possibile per architetture speciali, quali ad esempio quella di figura 11 **B**, in cui le connessioni che producono la ricorsività sono solo locali ai neuroni.
Nei protocolli di apprendimento, considerati fino a questo punto, si è assunto che l'apprendimento consiste solo nella variazione dei pesi e che questo ha luogo a partire da una rete neurale con architettura predefinita. Il principio di indeterminazione, precedentemente menzionato, suggerisce, tuttavia, che la definizione stessa dell'architettura possa ragionevolmente costituire oggetto del processo di apprendimento. Tale assunzione, che ha solide basi neurobiologiche, conduce dunque allo studio di algoritmi atti creare e cancellare connessioni sinaptiche oltre che a variarne il peso corrispondente. Sono stati concepiti algoritmi di *growing* e di *pruning* delle connessioni basati tipicamente sul principio della sensibilità dei pesi rispetto al comportamento della rete. Sono stati anche proposti algoritmi genetici per sviluppare l'architettura adeguata a un certo "task". Infine, la supervisione può essere fornita in modo più sofisticato mediante uno schema di insegnamento, che mira a presentare il desiderato *target* in modo progressivo.

5. PROBLEM SOLVING

L'architettura ricorsiva della rete di Hopfield descritta nel paragrafo 3 è stata oggetto di molti studi non soltanto per le sue proprietà di memoria associativa, ma anche perché ben si presta alla soluzione di molti interessanti problemi di soddisfacimento di vincoli, che hanno spesso natura combinatoriale. Per illustrare questa interessante proprietà si consideri il classico problema di scacchi di al-

[3] In alcuni esperimenti di riconoscimento vocale, nel gruppo di ricerca di Herve Bourlard (IDIAP, Svizzera) sono state utilizzate reti neurali con circa un milione di pesi.

locare 8 regine su una scacchiera in configurazione di non attacco, ovvero in modo tale che non si mangino[4] (Figura 15).
Questo è un problema di soddisfacimento di vincoli. Occorre, infatti, che, per ogni regina, non siano presenti regine sulla stessa riga, la stessa colonna e le due diagonali. La soluzione del problema si ottiene mediante una rete ricorsiva con un numero di neuroni pari al numero di caselle della scacchiera. L'eccitazione di un neurone corrisponde alla presenza della regina sulla casella, l'inibizione corrisponde, invece, alla casella vuota. Per risolvere il problema occorre tradurre i vincoli del problema in corrispondenti vincoli sul valore delle attivazioni dei 64 neuroni. Le connessioni della rete neurale, solo inibitorie, si costruiscono associando a ogni neurone un peso negativo proveniente dai neuroni associati alle caselle che si trovano sulla stessa riga, sulla stessa colonna e sulle stesse due diagonali cui appartiene la casella associata al neurone in oggetto. Non ci sono, dunque, auto-connessioni e, inoltre, la matrice dei pesi è simmetrica; se l'unità *i* è connessa all'unità *j* da un peso w_{ij} allora vale anche il viceversa, cioè $w_{ji} = w_{ij}$. Si noti che a differenza del caso in cui la rete di Hopfield opera da memoria associativa, in questo caso non ci sono ingressi collegati e che la codifica del problema è tradotta nel pattern di interconnessioni. Si può dimostrare che con un simile insieme di collegamenti, partendo da una qualunque configurazione iniziale, la dinamica della rete neurale evolve verso un punto stabile in cui la soluzione rappresenta configurazioni con regine in posizione di "non attacco". Tuttavia, partendo da una configurazione casuale, l'evoluzione della dinamica della rete non garantisce che tutte le 8 regine siano piazzate sulla scacchiera. Si può anche dimostrare che l'evoluzione della dinamica corrisponde alla minimizzazione di una funzione energia e che i suoi minimi globali corrispondono a soluzioni del problema. Come nel caso dell'apprendimento tuttavia, la funzione può essere popolata da minimi locali, offrendo pertanto soluzioni spurie. La soluzione sommariamente illustrata per il problema delle 8 regine può essere estesa con metodologie generali per risolvere generici problemi di soddisfacimento di vincoli, tipicamente molto complessi dal punto di vista computazionale quali, per esempio, il problema del commesso viaggiatore e il *knapsack*. Le soluzioni offerte da questo approccio sono estremamente efficienti oltre a permettere una computazione parallela per ogni passo del processo dinamico. Il problema fondamentale, tuttavia, è che, come per l'apprendimento, si hanno talvolta soluzioni sub-ottime che possono non risultare soddisfacenti. In sostanza, con soluzioni basate su reti di Hopfield, una volta "programmate" le connessioni per codificare il problema da risolvere, si può anche conseguire in modo efficiente una soluzione per problemi intrattabili, ma questo non è ovviamente garantito[5].

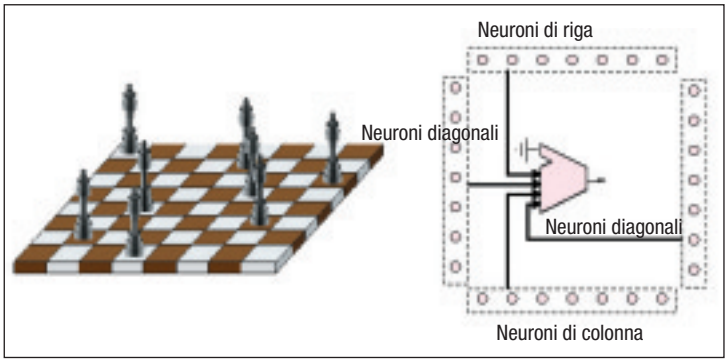


FIGURA 15
Il problema delle 8 regine e la sua soluzione mediante una rete di Hopfield

6. LE APPLICAZIONI

Uno dei motivi del successo delle reti neurali artificiali è probabilmente da ricercarsi nel loro massiccio utilizzo in innumerevoli applicazioni. Il paradigma di apprendimento da esempi su cui si basano permette, infatti, di affrontare problemi di natura anche molto di-

[4] La generalizzazione di questo problema al caso di *N* regine è stato per anni oggetto di congetture. Si riteneva si trattasse di un problema computazionalmente intrattabile, ma a metà degli anni novanta si è dimostrato che esiste una soluzione polinomiale per la determinazione di una configurazione.

[5] Si tratta, in sostanza, di una delle caratteristiche fondamentali del softcomputing menzionata nel paragrafo 2.

versa e di fornire soluzioni con uno sforzo relativamente limitato.
Questo è anche stato reso possibile dalla grande diffusione di pacchetti *software* per la simulazione dei modelli più importanti. I principali modelli neurali sono oggi disponibili anche in molti *tool* per *data mining* disponibili nei principali sistemi per basi di dati quali il DB2 (Database2). Oltre alla simulazione software, sono state studiate diverse soluzioni per l'implementazione in *hardware* di architetture neurali e dei corrispondenti algoritmi di apprendimento. Molti studi si sono concentrati su come utilizzare gli attuali modelli di calcolo parallelo per l'implementazione dello schema neurale, intrinsecamente parallelo. Sono fiorite innumerevoli soluzioni nei laboratori di ricerca che hanno avuto anche un certo impatto commerciale permettendo lo sviluppo di acceleratori neurali per integrare le capacità di calcolo di elaboratori tradizionali. Tali acceleratori sono tipicamente gestite da alcuni simulatori commerciali. L'impressionante evoluzione dei microprocessori che ha avuto luogo anche negli anni '90 ha, tuttavia, sostanzialmente ridimensionato l'importanza di tali soluzioni.
Si è anche assistito alla nascita di *chip* neurali analogici in grado di implementare i paradigmi di calcolo direttamente con variabili analogiche, senza bisogno di codifica discreta. In particolare, è degno di nota l'INTEL 80170, sviluppato nei laboratori INTEL all'inizio degli anni novanta. Studi simili sono stati computi soprattutto da Synaptics, (Object Recognizer Chip) e, in Italia, (TOTEM) della NeuriCam.
Uno dei problemi che ha, tuttavia, limitato lo sviluppo di chip tipo l'INTEL 80170 è la limitata precisione disponibile, che costituisce un problema soprattutto per gli algoritmi di apprendimento.

6.1. Applicazioni al riconoscimento di forme

Per illustrare la metodologia alla base di molte delle applicazioni riportate in tabella 2, si consideri il caso del riconoscimento di simboli grafici, eventualmente corrotti da rumore. Occorre pre-elaborare il pattern in modo da fornirne una rappresentazione più compatta da utilizzare in ingresso alla rete neurale. La limitazione del numero degli ingressi risulta particolarmente importante per limitare il numero degli esempi necessari per una corretta generalizzazione delle rete a nuovi esempi.
Nella figura 16, è illustrato l'uso di un percettrone multistrato per la classificazione di logo aziendali in 4 categorie. La rete ha 256 ingressi e 4 uscite, codificate in modo esclusivo, ovvero (1; 0; 0; 0); (0; 1; 0; 0); (0; 0; 1; 0); (0; 0; 0; 1). Il numero di neuroni nascosti si determina per tentativi utilizzando un test di validazione statistica.
Il percettrone multistrato dimostra eccellenti capacità di discriminazione di classi, ma non risulta efficace per attribuire un livello di confidenza nella sua decisione.
In altri termini, mentre molte applicazioni ne hanno dimostrato la grande efficacia nella discriminazione di classi note a priori, si è ormai accumulata evidenza sperimentale e supporto teorico per concludere che il percettrone non è in grado di attribuire in modo affidabile un peso alle sue decisioni. Questo rende tale rete neurale usata come classificatore inadatta a problemi in cui è necessario un comportamento di reiezione di pattern che non appartengono alle classi prestabilite.
Sempre usando il percettrone multistrato, si può ovviare a questo inconveniente mediante la configurazione ad autoassociatore illustrata in figura 17.

6.2. Sistemi ibridi

Molte delle applicazioni delle reti neurali a problemi reali richiedono un'opportuna organizzazione di sistema e non semplicemente l'utilizzo diretto dei modelli descritti in questo articolo. Per esempio, l'estrazione dell'informazione da una fattura acquisita mediante uno scanner richiede un opportuno sistema per la gestione documentale, dove le reti neurali possono giocare un ruolo strategico in alcune parti critiche.
A titolo di esempio, si consideri il problema del riconoscimento di targhe automobilistiche acquisite mediante ordinarie telecamere in ambiente autostradale. Tale applicazione è, per esempio, interessante per le società di gestione del traffico autostradale in corri-

Settore applicativo	Prodotto
Marketing	Airline Marketing Assistant, BehavHeuristics Inc Add-ins per Microsoft Excel, NeuroXL, 1998 AREAS, valutazione automatica immobili, HNC Software
Previsioni finanziarie	Neurodimension www.nd.com, 1991 NetProfit (profittaker.con), Neur. Appl. Corp. Appl.
Optical Character Recognition	Audre Neural Network, Audre Rec. Systems Appl. OmniPage 6.0 and 7.0 Pro for Windows, Caere OmniPage 6.0 Pro for MacOS AnyFax OCR engine FaxMaster, Delrina Technology Inc. VeriFone Oynx, lettore di assegni, Synaptics
Riconoscimento caratteri manoscritti	QuickStroke, ric. caratt. cinesi, Synaptics Teleform: ric. caratteri per fax, Cardiff Software Application, 1991
Riconoscimento manoscritti on-line	Apple Newton 120, Apple Lexicus Longhand, Lexicus (Motorola)
Nasi elettronici	AromaScan electronic nose, AromaScan Bloodhound Electronic Nose, Bloodhound Sensors Ltd e-NOSE 4000 electronic nose, Neotronics Scientific
Controllo di qualità cibi	test qualità birra Anheuser-Busch
Bond portfolio management	Global Bond, Econostat Ltd.
Controllo frodi (assegni)	Dunn and Bradstreet
Controllo frodi (carte credito)	Falcon, HNC Software Nestor In.
Verifica firma	Check Signature Verification System, NeuroMetric Vision System Inc.
Gestione rischio	Colleague, Aquarius, HNC Software
Predizione del consumo elettrico	Bayernwerk AG Application
Controllo chip microelettr.	INTEL
Controllo qualità gomme	Dunlop
Cancellazione di eco	AT&T/Lucent
Riconoscimento di banconote	BANK, D.F. Elettronica
Riconoscimento di targhe	PLARE, Società Autostrade e DII (Università di Siena)

TABELLA 2

Una lista di applicazioni di reti neurali che hanno dato origine a prodotti apparsi sul mercato

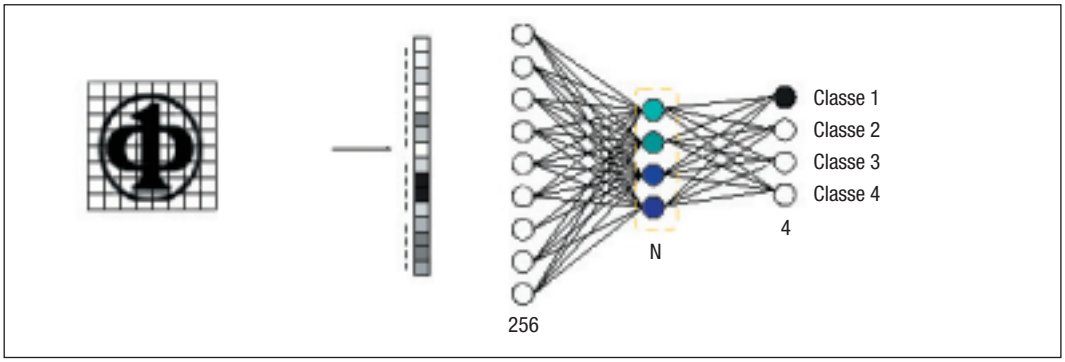


FIGURA 16
Classificazione di loghi aziendali con un percettrone multistrato

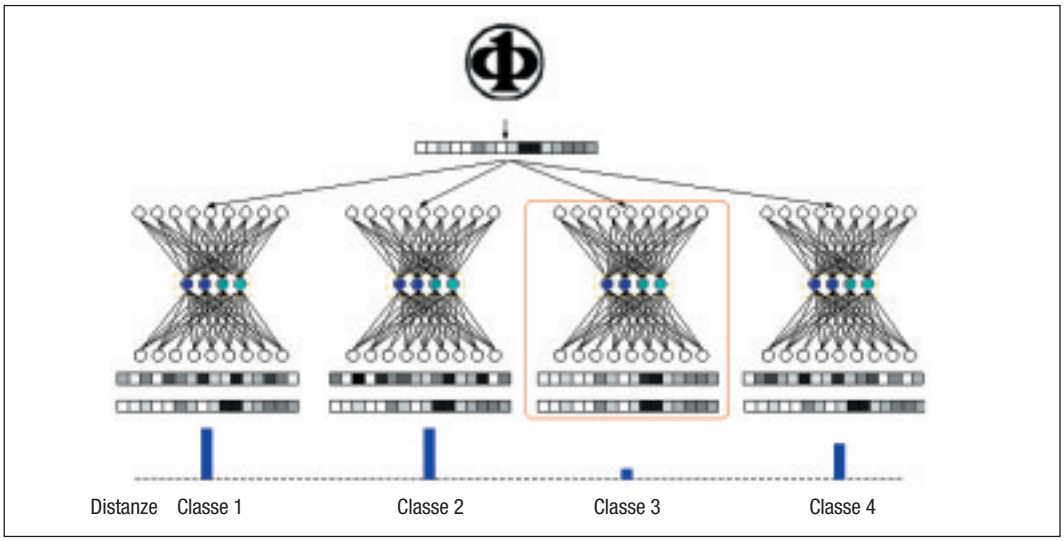


FIGURA 17
Autoassociatori neurali

spondenza delle stazioni di esazione a seguiti di infrazioni in impianti automatici.
In figura 18, è illustrata l'architettura complessiva di un sistema per il riconoscimento di targhe in sperimentazione presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Siena. Il sistema è composto da moduli sviluppati con tecnologia neurale (in rosa), da moduli basati su classici approcci di elaborazione delle immagini e da motori inferenziali. Un modulo di controllo provvede a sincronizzare le operazioni dei moduli *slave* delegati ad assolvere le funzioni di segmentazione della targa, dei caratteri e riconoscimento dei caratteri. Altri moduli esprimono vincoli grammaticali sulle stringhe possibili oltre a una probabilità a priori che si presenti una data targa. Il riconoscimento dei caratteri, che costituisce ovviamente l'attività critica, è basato su due moduli. Il primo contiene percettroni multistrato con struttura ad autoassociatore, che modellano le classi attese. La struttura può integrarsi dinamicamente quando si presenta una eventuale altra classe. Tale modulo ha la funzione di stabilire una lista di classi candidate, mentre il modulo a fianco, basato su percettroni multistrato con struttura a classificatore, serve a raffinare la decisione.
Tali classificatori sono tipicamente invocati dal modulo centrale quando i candidati si riferiscono a classi tipicamente molto confuse. In tal caso, appositi classificatori assolvono unicamente al compito di eliminare l'ambiguità derivante da classi molto confuse. Si noti che tali classificatori possono operare sulla stessa finestra di elaborazione del modulo precedente, ma anche su opportune finestre, decise dal modulo centrale, per enfatizzare le parti del pattern dove si localizzano verosimilmente le differenze. È anche interessante notare che il processo di segmentazione dei caratteri è raffinato dagli autoassociatori che posizionano la finestra in un intorno della posizione indicata

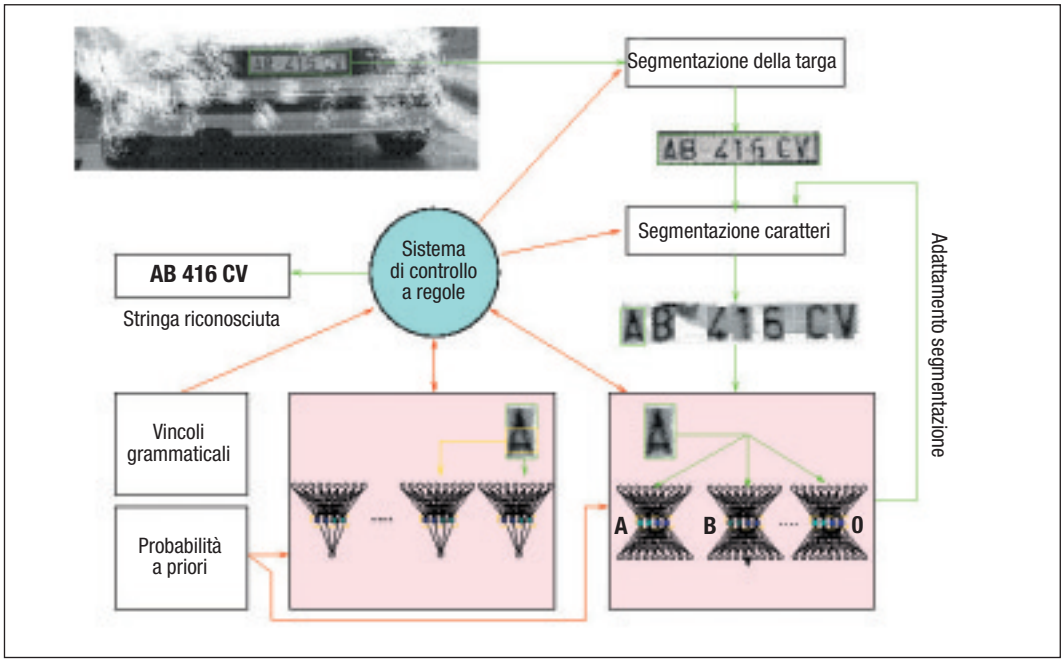


FIGURA 18
Architettura del sistema per il riconoscimento di targhe

dal modulo di segmentazione, a seguito di elaborazioni atte a stabilire la zona di massima risonanza.

7. LIMITI E PROSPETTIVE DI RICERCA

La ricerca nel settore delle reti neurali artificiali ha raggiunto un certo grado di maturità sia per quanto riguarda lo sviluppo sistematico delle metodologie fondamentali che il loro utilizzo in ambito applicativo. La comprensione dei limiti fondamentali sembra essere un passo fondamentale per lo sviluppo ulteriore del settore. Soprattutto in ambito applicativo, tali tecnologie sono state utilizzate talvolta in modo acritico confidando sul principio che la "forza bruta" derivante dall'impressionante sviluppo della microelettronica e il sogno della computazione inerentemente parallela potessero coniugarsi con l'apprendimento automatico per risolvere importanti problemi aperti con significativo risvolto applicativo. Un'analisi teorica dettagliata suggerisce la presenza di enormi ostacoli per un ulteriore sviluppo di approcci basati sugli attuali schemi di apprendimento automatico. Alcuni interessanti limiti erano già stati segnalati da Marvin Minsky nella sua edizione espansa di *Perceptrons* [10]. In particolare, Minsky aveva già individuato problemi legati all'euristica del gradiente, alla base di molti schemi di ottimizzazione utilizzati per le reti neurali e dettati dall'esigenza di ottimizzare in spazi di enorme dimensione. Una volta formulato nell'ambito dell'apprendimento neurale, la complessità inerente di un problema si rivela in termini della dimensione dello spazio dei pesi e della forma della superficie errore da ottimizzare. L'ostacolo fondamentale per l'euristica del gradiente è dovuto alla presenza di minimi locali sub-ottimi che intrappolano gli algoritmi di apprendimento. Per problemi complessi, l'esplosione del numero di tali minimi locali rende inverosimile la determinazione di soluzioni efficienti. Si è già accumulata evidenza teorica e sperimentale che, a fronte di problemi "complessi", la forma della superficie errore si "regolarizza" e diminuisce la presenza di minimi sub-ottimi all'aumentare della dimensione dello spazio dei pesi. Tuttavia, tale aumento di dimensione non solo conduce a un incremento di complessità nel calcolo del gradiente, ma introduce il problema addizionale dell'*over-training*, secondo cui l'apprendimento in presenza di spazi dei parametri troppo grossi non garantisce una corretta generalizzazione a nuovi esempi. È forse il momento di costruire teorie computazionali dell'apprendimento adatte al calcolo neurale, tipicamente definito nel continuo, invece, che nel tradi-

zionale contesto discreto. Il PAC (*Probably Approximately Correct*) *learning*, ampiamente utilizzato fin qui per la comprensione della complessità, sembra piuttosto sterile e non appare molto efficace per la comprensione di tipici contesti applicativi[6]. Sembra, inoltre, importante procedere nella direzione di sviluppare architetture e algoritmi di apprendimento nel contesto di ingressi strutturati, opportunamente rappresentati. Questo favorisce per altro lo sviluppo di integrazioni più forti tra modelli simboli e sotto-simbolici e sembra suggerire, in generale, la formulazione di teorie più generali per il trattamento di dati continui. A tal proposito, Margaret Boden, con riferimento alla novella del "Mago di Oz" scrive "[...] t*he pretty creature was visibly the same horse, changing colour as it trotted along. ... AI is one beast, like the Wizard's pony*", [4] proponendo il parallelo dei colori del pony con i diversi colori dell'intelligenza artificiale. Servono forse nuove interessanti miscele di colori, servono schemi per modellare in modo più naturale l'incertezza, serve comprendere più a fondo l'ingrediente evoluzionistico delle specie per coniugarlo con l'apprendimento automatico. E anche gli schemi di apprendimento devono verosimilmente risultare meno rigidi e, soprattutto, devono risultare attivi, permettendo un'interazione tra la macchina che apprende e il suo supervisore[7]. Forse non importa il "colore" del pony; le reti neurali, e più in generale la *computational intelligence*, devono integrarsi in modo più forte con i classici modelli simbolici. L'ibrido può non solo risultare vincente nelle applicazioni, ma può originare nuove miscele di colori, ben distinte dai componenti.

Bibliografia

[1] Ablameyko S., Goras L., Gori M., Piuri V.: *Limitations and Future Trends in Neural Computation*. IOS Publishing, (Eds 2003).

[2] Anderson J., Rosenfeld E.: *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, (Eds. 1988).

[3] Angluin D., Smith C.: Inductive inference: Theory and methods. *Computing Surveys*, Vol. 15, n. 3, 1983, p. 237-269.

[4] Boden M.: *Horses of a different colour? In Artificial Intelligence and Neural Networks*. V. Honavar and L. Uhr, Eds. Academic Press, 1994, p. 3-19.

[5] Frasconi P., Gori M., Sperduti A.: A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, Vol. 9, 1998, p. 768-786.

[6] Hebb D.: *The Organization of Behavior. Wiley, New York*, 1949. Partially reprinted in [Anderson and Rosenfeld, 1988].

[7] Hopfield J.: Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, USA, Vol. 79, 1982, p. 2554-2558. Also in Neurocomputing, The MIT Press, 1988.

[8] McCulloch W., Pitts W.: A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, Vol. 5, 1943. Reprinted in [Anderson and Rosenfeld, 1988].

[9] Mead C.: *Analog VLSI and Neural Systems*. Addison Wesley, Reading, 1989.

[10] Minsky M., Papert S.: *Perceptrons - Expanded Edition*. MIT Press, Cambridge, 1988.

[11] Rosenblatt F.: *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanism*. Spartan Books, Washington D.C, 1962.

[12] Rumelhart D., Hinton G., Williams R.: *Learning internal representations by error propagation*. In Parallel Distributed Processing, D. Rumelhart and J. McClelland, Eds. Vol. 1. MIT Press, Cambridge, Chapter, Vol. 8, 1986, p. 318-362. Reprinted in [Anderson and Rosenfeld, 1988].

[13] Seeley R., Stephens T., Tate P.: *Essentials of Anatomy and Physiology*, McGraw-Hill, 2002.

[14] Widrow B., Hoff M.: Adaptive switching circuits. In *IRE WESCON Convention Record*. IRE, New York, Vol. 4, 1960, p. 96-104.

MARCO GORI è professore ordinario all'Università di Siena presso il Dipartimento di Ingegneria dell'Informazione. Ha ottenuto il Dottorato di ricerca all'Università di Bologna, completando la formazione presso la "School of Computer Science" di McGill University, Montreal. I suoi interessi di ricerca riguardano l'intelligenza artificiale e le sue applicazioni. È attualmente chair del capitolo italiano della Neural Networks Society ed è presidente dell'Associazione Italiana Intelligenza Artificiale.
marco@dii.unisi.it

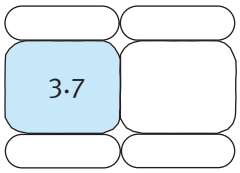
[6] Per lo stato dell'arte sui limiti e sulle prospettive del calcolo neurale si può far riferimento a Ablameyko *et al.* [1].

[7] La ricerca nel settore del *learning from queries and examples* ha già fornito promettenti indicazioni di riduzione di complessità, ma sembra ancora essere in una fase incipiente, soprattutto per le ricadute applicative.

IL RUOLO DEL SOFTWARE SULL'INSICUREZZA DEI SISTEMI ICT

Danilo Bruschi

L'affermazione di Internet ha contribuito a far emergere il problema dell'insicurezza delle tecnologie ICT. Questo fenomeno è spesso riconducibile al software: errori presenti in esso o il suo "cattivo" uso determinano gran parte dei problemi di sicurezza. Per capire la natura del fenomeno "sicurezza informatica", e sapersi ad esso rapportare, è necessario capire come un errore presente in un programma possa essere utilizzato per compromettere la sicurezza del sistema su cui opera.



1. INTRODUZIONE

Nell'ultimo decennio, il mondo occidentale ha dovuto prendere atto, accanto all'affermazione delle tecnologie dell'informazione e della comunicazione (ICT), della "insicurezza" di tali sistemi. Il fenomeno già noto agli esperti a partire dai primi anni '60, ha assunto, con la diffusione delle rete Internet, dimensioni tali da non poter essere più circoscritto a un ristretto numero di persone ed è diventato argomento d'interesse per larghe fasce di utenti. Oggi, è difficile che ci sia un qualunque evento mediatico che affronta il tema delle tecnologie dell'informazione e della comunicazione senza considerarne gli impatti sulla sicurezza.

Ma precisamente che cosa si intende per tecnologia dell'informazione e della comunicazione sicura? Nel gergo comune, l'aggettivo *sicuro* denota principalmente due diverse proprietà di un sistema. Infatti, si dice che un sistema è sicuro quando, anche in presenza di malfunzionamenti, non provoca danni a chi lo usa (è, per esempio, in questa accezione che si parla di automobili sicure, o di apparecchi elettrici sicuri); oppure, un sistema è sicuro quando è in grado di preservare il proprio contenuto da accessi non autorizzati. È in questa accezione che, per esempio, si parla di una banca sicura, di una casa sicura ecc.. Nell'ambito informatico l'aggettivo *sicuro* viene usato con quest'ultima accezione. Quindi, un sistema informatico è sicuro quando protegge da "accessi non autorizzati" il proprio contenuto, cioè i dati in esso memorizzati e i servizi che eroga.

Da questa elementare considerazione discende la definizione universalmente accettata di sistema informatico sicuro. Tale definizione può essere così sintetizzata: *un sistema di calcolo viene considerato sicuro quando è in grado di garantire il soddisfacimento delle proprietà di confidenzialità, integrità e disponibilità*. Più precisamente, quando il sistema è in grado di garantire che: ogni utente può accedere esclusivamente alle informazioni di sua competenza (confidenzialità o riservatezza), ogni utente può modificare solo informazioni di sua competenza (integrità) e ogni azione intrapresa da

persone non autorizzate mirata a compromettere il corretto uso di una qualunque risorsa del sistema, è preventivamente bloccata (disponibilità). La sicurezza informatica è, invece, quella branca dell'informatica che studia e individua le tecniche e le metodologie per rendere le tecnologie ICT sempre più sicure.

Il problema della sicurezza delle tecnologie ICT è stato affrontato a partire dalla metà degli anni '60, sotto la spinta di due avvenimenti: l'avvento dei sistemi operativi multi-utente e l'introduzione massiccia dei sistemi di calcolo in ambiti militari. È ARPA (*Advanced Research Projects Agency*) l'agenzia per la ricerca del DoD (*Department of Defense*) statunitense ad avviare il primo progetto relativo alla costruzione di un sistema operativo sicuro: il **MULTICS**, che viene sviluppato a partire dai primi anni '60, al Massachusetts Institute of Technology (MIT) in collaborazione con Honeywell e General Electric. Alla base del progetto c'è la considerazione che se la condivisione delle risorse di un sistema di calcolo è prima di tutto una necessità economica, nel momento in cui più utenti di un calcolatore condividono memoria centrale, CPU (*Central Processing Unit*) e spazio su disco, è possibile per un utente interferire (accidentalmente o volontariamente) con le attività svolte da un altro utente. Diventa, quindi, prioritario individuare e realizzare una serie di meccanismi sia *hardware* che *software* (in particolare a livello di sistema operativo) per garantire che nell'ambito di sistemi condivisi da più persone, un utente sia vincolato a svolgere solo le operazioni di propria competenza.

Sullo slancio di queste motivazioni venne dato l'avvio a un'intensa attività di studio e ricerca svolta principalmente a livello accademico, mirata all'individuazione dei meccanismi più efficaci per la protezione dei dati e dei programmi nell'ambito dei sistemi e nell'individuazione di sistemi per la verifica formale dei programmi.

MULTICS è stato sottoposto a un processo di certificazione di sicurezza, secondo i criteri TCSEC (*Trusted Computing Security Evaluation Criteria*) proposti dal DoD. Nell'ambito di questo standard i sistemi informatici sono classificati secondo sette livelli di valutazione: D, C1, C2, B1, B2, B3, A1 in base alle funzionalità di sicurezza che sono in grado di offrire. D è il livello assegnato ai sistemi, che non offrono alcuna garanzia di protezione. Verso la fine degli anni '80 viene certificato che MULTICS soddisfa i requisiti imposti dal livello di certificazione B2. Una certificazione di sicurezza di tutto rispetto se si pensa che la configurazione di *default* di un attuale sistema operativo non soddisfa nemmeno i criteri di sicurezza previsti per il livello C2 e che il massimo livello di certificazione sinora ottenuto da un qualunque altro sistema operativo commerciale non ha mai superato B1.

I primi risultati di queste ricerche, in particolare, sono immediatamente applicati all'interno del sistema operativo MULTICS, che ancora oggi risulta essere il sistema operativo commerciale più sicuro che mai sia stato realizzato.

Già dall'esperienza MULTICS apparve evidente che la presenza in un sistema di calcolo di meccanismi di protezione non bastava da sola a garantire la sicurezza del sistema stesso. Infatti, i ricercatori che lavoravano a questo progetto constatarono immediatamente che il software che implementava questi meccanismi, o il software di sistema, su cui questi meccanismi tipicamente si appoggiano, conteneva errori commessi nelle fasi di progettazione e/o di programmazione, genericamente indicati come *security bug*. Questi errori consentivano, in determinate occasioni, di "bypassare" i sistemi di protezione rendendoli, quindi, del tutto inutili. Questo problema ha poi accompagnato tutti progetti software di un certo rilievo sviluppati sino ai nostri giorni, e ancora oggi è tra le cause principali dell'insicurezza dei sistemi informatici.

Non si è, quindi, di fronte a un problema determinato dalla mancanza di strumenti concettuali o tecnologici per la sua soluzione, ma *si è di fronte a un problema generato nella stragrande maggioranza dei casi da una incapacità di implementare correttamente soluzioni*[1]. Questo aspetto rende la realizzazione di *sistemi informatici sicuri* estremamente complessa, o meglio impossibile. Infatti, realizzare un sistema informatico sicuro significa, innanzitutto, realizzare un sistema il cui software non contenga security bug, e che sia correttamente installato. Purtroppo, ad oggi, non esiste alcun tipo di pro-

1 Sono esclusi da questo discorso i problemi di sicurezza legati alla non disponibilità dei sistemi, che solo rare volte sfruttano *security bug*.

cedimento manuale o automatico che consente la verifica *a priori* di tali proprietà, e non si intravede nemmeno la possibilità di realizzarlo in futuro. Quindi, l'unica cosa che resta da fare è prodigarsi per aumentare il livello di sicurezza dei sistemi attuali, con interventi mirati. Anche in questo caso vale comunque la pena ricordare che nonostante oggi si posseggano metodologie e strumenti per progettare sistemi informatici con un buon livello di protezione, la realizzazione degli stessi è sempre al di sotto delle aspettative. Il problema è sempre il solito: il *gap* che esiste tra il *dire* e il *fare* è nell'ambito della sicurezza fatale. Il fattore umano (in termini di conoscenza, esperienza ma anche limiti naturali) gioca un ruolo determinante nella realizzazione di questi sistemi.
Capire quali sono i presupposti affinché tutto ciò possa verificarsi e come un errore presente in un programma possa stravolgere i principi di funzionamento di un sistema significa capire quali sono i fondamenti su cui si basa l'(in)sicurezza dei sistemi informatici, e, quindi, possedere le nozioni necessarie per poter correttamente affrontare il problema. In questo articolo si cercherà di fornire al lettore le nozioni necessarie per poter cogliere questi elementi. Non si parlerà, quindi, delle metodologie e tecnologie per realizzare sistemi sicuri, ma delle principali cause del fenomeno, convinti che solo una conoscenza approfondita delle stesse sia un fattore abilitante per l'individuazione di soluzioni efficaci anche se solo parziali.

2. LE DIMENSIONI DEL PROBLEMA

Quali sono le conseguenze di un security bug? Quanti sono i security bug noti fino ad oggi? Sono queste alcune delle domande necessarie per poter quantificare correttamente il problema sopra descritto. Ad oggi, sono circa 10.000 [3] le vulnerabilità note per poter aggirare i meccanismi di protezione dei diversi sistemi informatici che popolano il nostro pianeta. Ognuna di queste vulnerabilità può dare origine ad una o più tecniche di attacco o di intrusione (per una definizione precisa di questi **termini** si veda il riquadro), che consentono lo svolgimento sul sistema attaccato di diverse operazioni non autorizzate. In particolare, in riferimento alle attività che un utente è in grado di compiere, le tecniche di attacco informatico sono raggruppabili in 8 diverse categorie riportate nella tabella 1.

3. L'AFFIDABILITÀ DEL SOFTWARE

Come è stato anticipato la principale causa dell'insicurezza dei sistemi informatici è riconducibile ai security bug. Questi errori possono essere stati compiuti in ciascuna delle fasi che costituiscono il ciclo di vita del software: disegno o progettazione, programmazione, test e installazione. Un errore in una di queste fasi si riflette inesorabilmente nella "messa in opera" di un prodotto "vulnerabile", che in particolari circostanze consente a un intrusore di compiere sul sistema attività non autorizzate. Si vedano, nel seguito, quali sono i tipici errori che possono essere commessi in ciascuna di queste fasi.
In fase di disegno o progettazione di un programma è necessario definire il *thread model*, cioè una descrizione semi-formale di tutti i possibili comportamenti scorretti che il programma potrebbe originare. Conseguentemente, il programma deve essere progettato per poter far fronte a ciascuno di

Quando si parla di sicurezza informatica, **termini** come incidente, attacco, vulnerabilità e minaccia sono usati molto frequentemente e spesso, in contesti diversi, lo stesso termine viene ad assumere significati diversi. È quindi estremamente importante quando si affrontano questi problemi definire con precisione il significato di questi termini. In questo contributo, si è deciso di rifarci a uno sforzo di standardizzazione sul significato degli stessi, in corso nella comunità scientifica e i cui contributi di riferimento sono:
attacco o intrusione: una serie di attività che possono essere svolte su un sistema per poter svolgere da un intrusore per poter svolgere attività non autorizzate;
attaccante: un individuo che esegua uno o più attacchi per poter raggiungere i propri obiettivi;
evento: un'azione diretta verso un determinato obiettivo, con l'intento di modificarne lo stato;
IT security incident: ogni azione avversa rivolta contro un sistema IT, con l'intento di raggiungere uno dei seguenti obiettivi: violazione della confidenzialità/integrità dei dati, compromissione del livello di disponibilità dei servizi;
vulnerabilità: una falla o debolezza presente in un sistema che consente di eseguire sullo stesso un'azione non autorizzata.

Tipologia dell'incidente	Definizione	Esempi di tecniche utilizzate[1]	Potenziali effetti ottenibili
Computer Fingerprinting	Attività svolte al fine di raccogliere informazioni in merito a un host	Probing e scanning	Elenco dei servizi disponibili sull'host vittima e sue caratteristiche
Codice Maligno	Compromissione di un host attraverso l'esecuzione di programmi indipendenti	Virus, Trojan, spyware	Violazione della riservatezza e integrità dei dati e indisponibilità dei servizi. Abuso dei sistemi
Denial of service	Accessi continui a un servizio ai fini di saturarne le risorse	SYN-flood, Ping of Death, Land, WinNuke, TFN, TFN2K, Trin00, Slice3	Messa fuori uso, temporanea, del servizio attaccato
Account Compromise	Accesso non autorizzato a un sistema o alla risorsa di un sistema, in qualità di amministratore di sistema o di utente	Buffer overflow, Format bug, o uso di credenziali di accesso (username e password)	Violazione della riservatezza e integrità dei dati e indisponibilità dei servizi. Abuso dei sistemi
Accesso non autorizzato alle informazioni	Accessi non autorizzati lettura e/o scrittura dati	SQL-injection, Spyware	Violazione della riservatezza dei dati
Accesso non autorizzato al canale di comunicazione	Interferenze senza le necessarie autorizzazioni alla trasmissione di dati	Hijacking, replay attack, sniffing, ARP poisoning	Violazione della riservatezza e dell'integrità dei dati trasmessi in rete
Accesso non autorizzato a sistemi di comunicazione	Uso non autorizzato di sistemi e protocolli per la comunicazione	DNS spoofing, mail relays, war driving	Violazione della riservatezza di dati e accesso non autorizzato ai sistemi
Modifica non autorizzata di informazioni	Modifica di dati presenti su un computer senza le necessarie autorizzazioni	Web defacements, viruses, SQL-injection	Violazione dell'integrità dei dati

[1] Il lettore interessato ad approfondire le diverse tecniche può consultare il sito web riportato in bibliografia [3].

TABELLA 1
Tassonomia delle tecniche di intrusione informatica, in base alle attività svolte sul sistema vittima

questi attacchi. L'omissione di questa fase o una noncuranza durante il suo svolgimento porta a definire un *thread model* incompleto e, quindi, a progettare un prodotto che contiene delle vulnerabilità. Nel prossimo paragrafo verrà illustrato un esempio di errori di questo tipo, che ha consentito la realizzazione di un attacco informatico, ancora oggi molto praticato.
In fase di programmazione possono essere fatte da parte del programmatore delle scelte errate nell'implementazione di alcuni algoritmi, oppure essere utilizzate istruzioni che consentono lo svolgimento di attacchi come il *buffer overflow* che sarà descritto nel paragrafo 5. Per esempio, un baco presente nel programma di sistema *finger*, nell'ambito del sistema operativo SunOS, ha consentito nel 1988 la realizzazione del più famoso attacco informatico avve-

nuto su Internet e noto come **Internet Worm**. Nel 2001, un errore di programmazione commesso nella *routine* che interpretava le stringhe di *input* nel programma *Internet Information Server* (IIS, il *web server* di Microsoft), consentiva a un qualunque utente di prendere il controllo del sistema su cui era in esecuzione il programma.

Il 3 novembre del 1988 Robert Morris, uno studente di Ph.D della Cornell University, attraverso un programma da lui scritto, riuscì a guadagnare l'accesso e a mettere fuori uso circa 6000 calcolatori operanti in Internet. L'attacco noto come **Internet Worm** mise fuori uso calcolatori appartenenti a università, laboratori di ricerca, enti governativi e industrie. La notizia ebbe una vasta eco da parte dei *media* di tutto il mondo, che così prendeva consapevolezza del problema della sicurezza informatica.

In generale, i bachi di sicurezza sono difficilmente individuabili, poiché difficilmente influenzano il comportamento di un programma. Un errore di sicurezza difficilmente è causa di calcoli errati, visualizzazioni errate, o di messaggi di errori. Un programma che contiene errori di sicurezza può essere perfettamente aderente alle specifiche funzionali per cui è stato concepito. Diventa, quindi, molto difficile individuare security bug e solo un'accurata fase di test può rivelarne la presenza all'interno del codice. In fase di test tutte le possibili alternative offerte dal programma dovrebbero essere verificate, in particolare a partire da un thread model è necessario verificare il comportamento del programma quando sottoposto a tutti i possibili tentativi di attacco. La dimensione dei programmi e la necessità di distribuire a ritmi sempre più sostenuti nuove versioni di software per poter reggere il passo della concorrenza e i costi da sostenere, fanno sì che la fase di test sia però generalmente svolta in modo molto approssimativo con inevitabili ricadute sulla sicurezza del prodotto finale. Nell'ambito di progetti che godono di una certa criticità è invalsa l'abitudine di coinvolgere nelle fasi di test quelli che in gergo sono chiamati *Tiger Team* o *Red Team*. Si tratta di gruppi di esperti che tentano di forzare i sistemi di protezione di un prodotto software con l'obiettivo di individuarne eventuali falle. I tiger team sono solitamente composti da personale esterno e complementano così l'attività di test.

Un ulteriore elemento di criticità per la sicurezza dei sistemi è la fase di installazione del software. In questa fase, infatti, possono essere compiuti, da utenti poco esperti, una serie di errori che possono essere sfruttati per accedere abusivamente ai sistemi. Per esempio, ancora oggi, uno degli errori più comuni in fase di installazione di un sistema operativo è la creazione di utenze senza *password*, che consentono, quindi, di accedere al sistema senza particolari controlli facilitando gli accessi non autorizzati. Questo tipo di problema ha, come quelli precedentemente accennati, una forte incidenza sulla sicurezza dei sistemi informatici. Per ragioni di spazio ci si concentrerà, nei prossimi due paragrafi, sui problemi che si ritengono concettualmente più importanti: ovvero, gli errori commessi in fase di progettazione e programmazione.

4. ERRORI DI PROGETTAZIONE

In questo paragrafo, verrà descritto un attacco portato al protocollo ARP (*Address Resolution Protocol*) e noto come ARP Poisoning. L'attacco è ancora oggi molto diffuso e deriva da una serie di scelte errate commesse in fase di progettazione. Lo scopo di questo paragrafo è fornire un esempio concreto di come errori commessi in fase di progettazione del software, si ripercuotano nelle fasi successive di vita del software sino a comprometterne il corretto funzionamento.

In breve, il protocollo ARP è utilizzato in tutte le reti locali che adottano il **protocollo Ethernet** (circa il 98% di tutte le reti locali). Tale protocollo prevede che tutti gli *host* della LAN siano identificati attraverso un indirizzo numerico di 48 bit noto anche come MAC *address*. Le informazioni che i vari host di una LAN devono trasmettersi sono racchiuse in uno o più pacchetti, contenti il MAC address del destinatario e inviati sulla rete. In ricezione quando un host sulla LAN riceve un pacchetto verifica se l'indirizzo presente nel pacchetto è il

Generalmente, un host possiede tanti identificativi quante sono le reti diverse a cui è connesso. Nel caso della LAN basate sul **protocollo Ethernet** (adottato nell'ambito dello standard IEEE 802.3), l'identificativo è un numero di 48 bit, espresso come una sestupla di coppie di cifre esadecimali (per esempio, FF-00-23-78-AB-11). Questo identificativo non deve essere confuso con l'indirizzo IP, che serve, invece, per identificare un host sulla rete Internet. Un indirizzo IP è un numero di 32 bit, che viene espresso come una quadrupla di numeri compresi tra 0 e 255. All'indirizzo IP può anche essere associato un indirizzo simbolico del tipo nome@dico.unimi.it

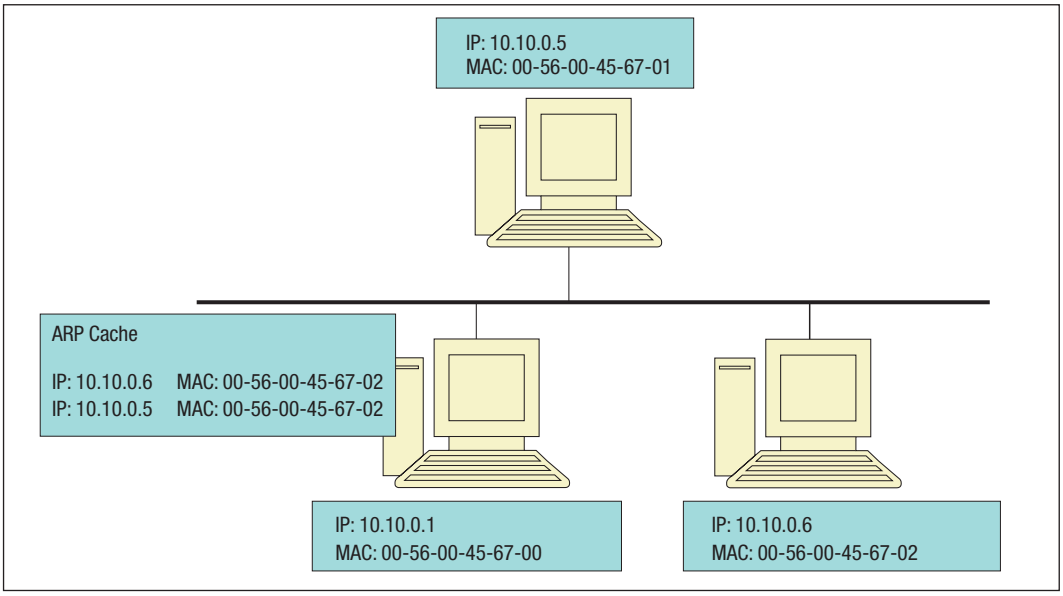


FIGURA 1
Schema di una semplice rete locale e di un attacco di ARP poisoning

suo, e in tal caso elabora le informazioni contenute, altrimenti lo elimina.
In genere, però il MAC address di un host non è noto, ma è noto solo il suo indirizzo IP (*Internet Protocol*) o indirizzo simbolico. È il protocollo ARP che nell'ambito di una LAN si preoccupa, dato l'indirizzo simbolico di un host, di individuarne il corrispondente MAC address[2]. Per esempio, si supponga che un host il cui indirizzo simbolico è host1@rete.dominio.it voglia comunicare con host2@rete.dominio.it, presente sulla stessa LAN. Prima che questa comunicazione avvenga, il protocollo ARP in esecuzione su host1@rete.dominio.it invia un messaggio a tutti gli host presenti sulla LAN (ARP Request), chiedendo all'host, il cui indirizzo simbolico è host2@rete.dominio.it, di fargli conoscere il proprio MAC address. Tra tutti gli host presenti sulla rete solo host2@rete.dominio.it risponderà a questo messaggio, e host1@rete.dominio.it provvederà a memorizzare le risposte ricevute, in una opportuna tabella chiamata *ARP cache*. Per ovvie ragioni di efficienza ogni host prima di inviare un *ARP request* consulta la propria ARP *cache* per verificare se il MAC address ricercato sia già presente.
ARP è stato progettato in modo da procedere all'aggiornamento dei valori presenti nella propria cache anche se non esplicitamente richiesti. Vale a dire, un host può di sua spontanea volontà, informare un altro host che il suo indirizzo MAC è stato modificato. L'host che riceve quest'informazione provvede a eseguire l'aggiornamento dell'informazione ricevuta, senza alcuna verifica sulla sua validità. Questa scelta progettuale dettata dalla necessità di realizzare un protocollo estremamente efficiente che prontamente si adatta alle modifiche della rete, è però anche una vulnerabilità che opportunamente sfruttata consente l'effettuazione del seguente attacco noto come ARP Poisoning.
Lo schema dell'attacco è abbastanza semplice. Si consideri la rete locale in figura 1. Si supponga che l'utente che opera sull'host 10.10.0.6 voglia intercettare tutto il traffico di rete diretto dall'host 10.10.0.1 all'host 10.10.0.5. Sfruttando le "proprietà" del protocollo ARP deve limitarsi a inviare un messaggio ARP per l'aggiornamento della cache all'host 10.10.0.1, e comunicargli che il MAC address di 10.10.0.5 è diventato 00-56-00-45-67-02. Quando l'host 10.10.0.1 riceverà questo messaggio aggiornerà il contenuto della propria cache, e ogni volta che dovrà inviare un messaggio all'host 10.10.0.5 lo invierà all'host che sulla rete locale ha l'indirizzo 00-56-00-45-67-02 quindi all'host 10.10.0.6, che raggiunge così l'obiettivo inizialmente preposto.

2 Qualora l'host ricercato non sia presente sulla LAN, il MAC address che verrà fornito dall'ARP è quello dell'Internet Gateway che provvederà a inoltrare il pacchetto sulla rete Internet.

Recenti studi sono stati intrapresi per cercare di ovviare a questa vulnerabilità e diverse soluzioni sono state proposte dalla comunità scientifica [2]. Tutte queste soluzioni presuppongono una o più *patch* a livello del *kernel* di sistema operativo: ciò significa che affinché queste soluzioni possano essere adottate universalmente devono essere incluse nelle distribuzioni ufficiali dei sistemi operativi. Nessun costruttore sembra però interessato al problema e le LAN continuano a essere soggette a questo tipo di attacco.
Quello che è stato appena descritto è un esempio emblematico dei problemi legati alla sicurezza dei calcolatori. Di protocolli vulnerabili come ARP ve ne sono molti, via via che sono individuati danno origine a nuove tecniche di attacco, che restano efficaci sino a che le corrispondenti patch correttive non sono individuate. Va comunque segnalato che per correggere errori di questa natura è necessario rivisitare il protocollo originale, e le modifiche in genere richieste sono abbastanza radicali (e possono a loro volta contenere errori). Non va poi dimenticato che il tempo che intercorre tra il rilascio delle correzioni e la loro messa in opera da parte degli utenti finali può anche essere dell'ordine di decine di mesi, in questo periodo, quindi, i sistemi restano comunque esposti a vulnerabilità note.

5. ERRORI DI PROGRAMMAZIONE

Tra le tecniche di attacco informatico più diffuse va sicuramente annoverato il *buffer overflow* introdotto da Morris con l'Internet Worm. Questo attacco usa delle peculiarità del linguaggio di programmazione C in cui è scritto gran parte del codice di sistema sia in ambito Unix che in ambito Windows. In particolare, il buffer overflow sfrutta il fatto che il compilatore C non controlla che in un'operazione di trasferimento dati la variabile sorgente abbia una dimensione superiore a quella di destinazione. In fase di esecuzione questa anomalia si traduce nel fatto che i dati superflui della variabile sorgente verranno scritti nelle zone di memoria circostanti la variabile di destinazione.
Per esempio, se all'interno di un programma

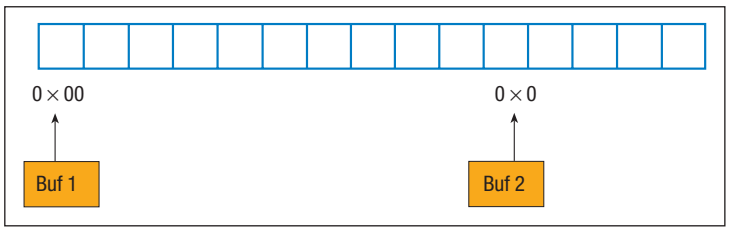


FIGURA 2
Schema di allocazione della memoria per le variabili Buf1 e Buf2

C sono presenti le seguenti istruzioni dichiarative:

```
char Buf1 [10];

char Buf2 [7];
```

il compilatore provvederà ad assegnare due aree di memoria contigue, in particolare poiché la dichiarazione di Buf1 precede quella di Buf2, a Buf1 saranno assegnate locazioni di memoria con indirizzi immediatamente inferiori a quelli assegnati alle locazioni destinate a contenere Buf2. Schematicamente la mappa di memoria può essere rappresentata nel modo indicato in figura 2.
Si supponga ora che, sempre all'interno dello stesso programma, si trovi la seguente istruzione di assegnamento:

```
Buf1 = "zzzzzzzzzzzz";
```

è facile constatare che con questa istruzione si tenta di inserire una stringa di 12 caratteri in una zona di memoria (Buf1) predisposta a contenerne 10.
Se, invece, del linguaggio C si stessero considerando altri linguaggi di programmazione come C++, Java, Pascal, C#, il compilatore rileverebbe questa inconsistenza e darebbe un messaggio di errore. Nel caso del compilatore C, invece, il problema non viene rilevato, anzi il compilatore provvede a recuperare lo spazio mancante dalle variabili contigue a Buf1. Più precisamente, la suddetta istruzione di assegnamento viene, schematicamente parlando, tradotta nel seguente modo:

```
i = indirizzo iniziale di Buf1;

j = 1;
```

while (l'elemento della stringa di dati da cari-

care in Buf1 di posizione *j* è diverso dal carattere di fine stringa)

Buf1 [i] = *j*-esimo elemento della stringa di input

i = *i* + 1;

j = *j* + 1;

endwhile

Questo significa che quando la suddetta istruzione di assegnamento sarà eseguita, la situazione che verrà a crearsi in memoria sarà quella rappresentata in figura 3.

Per esempio, la stringa http://www.bibliography.it/airchronicles/aureview/1979/jan-feb/schell.html digitata da un utente per accedere a un dato sito Web, viene memorizzata in un'apposita area di memoria del *browser*. Ovviamente, se nel browser non era stato predisposto un buffer di dimensioni opportune per ospitare le stringhe di input, la suddetta stringa ricoprirà i dati delle zone di memoria ad essa contigue.

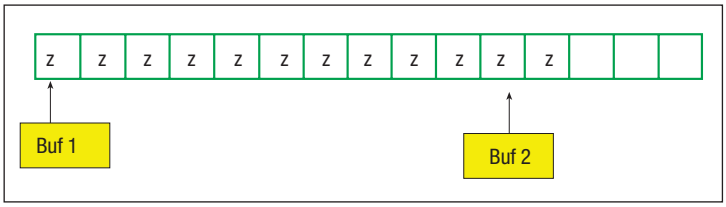


FIGURA 3

Configurazione dei due buffer dopo l'esecuzione dell'istruzione di assegnamento

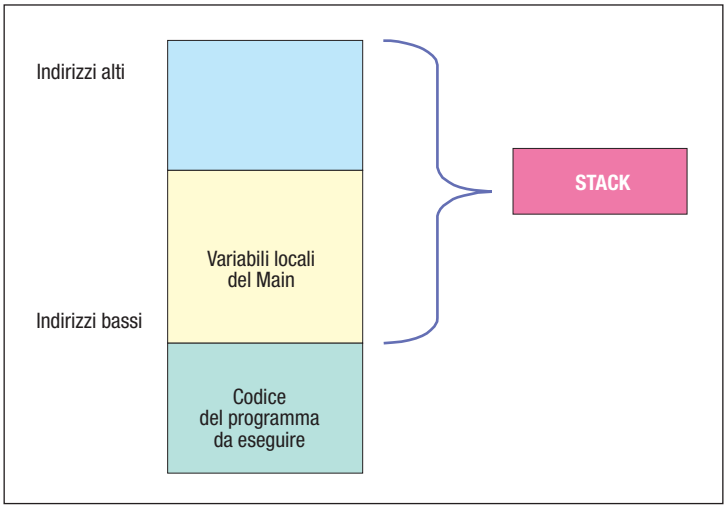


FIGURA 4

Schema di allocazione di un programma oggetto in memoria

Si veda nel seguito come questa caratteristica del C può essere utilizzata per compiere un attacco a un programma. Per fare ciò è necessario richiamare alcune nozioni relative alle modalità con cui viene caricato in memoria un programma per poter essere eseguito. Innanzitutto, si ricorda che un programma eseguibile è composto da due parti principali: una parte codice, che contiene le istruzioni da eseguire e una parte dati, che contiene i dati su cui il codice deve operare. Quando un programma deve essere eseguito sono caricati in memoria centrale le componenti codice e dati relative alla procedura principale (*main*). La parte dati, in particolare, viene caricata in uno *stack* i cui indirizzi decrescono, vengono cioè allocate prima le zone dello stack con indirizzi alti. Lo schema generale è riportato in figura 4.

Quando durante l'esecuzione del programma principale viene richiamata una procedura "secondaria", sullo stack viene salvato l'indirizzo di rientro al programma principale, cioè l'istruzione del programma principale che dovrà essere eseguita al termine dell'esecuzione della procedura secondaria, seguito dai dati relativi alla procedura stessa. Quindi, la situazione dello stack diventa quella riportata in figura 5.

Si supponga ora che la procedura secondaria usi una variabile di 512 caratteri per memorizzare dei dati di input. Tale variabile sarà allocata sullo stack e se nella fase di input venisse inserita una stringa più lunga di 512 caratteri, la stessa andrà a sovra scrivere (verso l'alto) le variabili adiacenti e paradossalmente questa operazione potrebbe essere effettuata dall'utente sull'indirizzo di ritorno. In questo caso, terminata l'esecuzione della procedura secondaria si passerebbe a eseguire l'istruzione il cui indirizzo è contenuto nella zona "riservata" all'indirizzo di ritorno che è però stato precedentemente sovra scritto, e quindi il programma originale non potrebbe continuare correttamente l'esecuzione. Se però, invece, di una stringa casuale l'utente avesse inserito una stringa contente il codice eseguibile di un programma, e avesse inserito l'indirizzo d'inizio di questo programma al posto dell'indirizzo di ritorno l'effetto otte-

nuto sarebbe stato decisamente diverso (Figura 6).
Nel momento in cui la procedura secondaria terminasse la propria esecuzione, verrebbe eseguita l'istruzione il cui indirizzo si trova nella zona di memoria riservata all'indirizzo di rientro e, quindi, il controllo invece che al programma originale sarebbe ceduto al programma scritto dall'utente. Questo programma potrebbe, per esempio, cancellare alcuni *file* dell'utente dal disco, modificare informazioni e nei casi peggiori arrivare anche a cancellare l'intero contenuto del disco fisso.
L'attacco è, ovviamente, molto difficile da realizzare, e richiede conoscenze molto approfondite delle architetture e dei sistemi operativi corrispondenti. In particolare, si è volutamente tralasciato una serie di particolari perché scopo di questo articolo è quello di consentire ai lettori di cogliere gli aspetti più importanti della tecnica.
Con questa tecnica sono stati compromessi i servizi di rete e i comandi più importanti di tutti i sistemi operativi. Diverse tecniche sono state finora proposte per far fronte a questo problema da parte della comunità internazionale [1]. Ancora oggi il buffer overflow è la tecnica più utilizzata per attaccare i sistemi e nonostante di questa tecnica si conosca ogni dettaglio realizzativo sono ancora molti i programmi di sistema che con questa tecnica sono e possono essere attaccati. Per esempio, il **worm Slammer** che nella primavera di quest'anno ha infettato più di 75000 host era basato su un attacco

Slammer (citato da alcuni autori anche come *Sapphire*) è stato il computer **worm** più veloce sinora realizzato. Slammer è "apparso" sulla rete il 25 gennaio 2003, e sfruttava attraverso il buffer overflow una vulnerabilità presente nei programmi Microsoft SQL Server e Microsoft SQL Server Desktop Engine (MSDE) 2000. La vulnerabilità era stata individuata nel Luglio 2002 e le corrispondenti patch correttive erano state rilasciate immediatamente dopo. Si stima che complessivamente Slammer abbia infettato più di 75.000 sistemi in tutto il mondo. La sua peculiarità è stata comunque la velocità di infezione. L'infezione della stragrande maggioranza dei sistemi è avvenuta nei primi dieci minuti di attività del virus, che ha poi rallentato la sua velocità di propagazione per errori presenti nel codice del virus stesso. A titolo di paragone si rammenta che prima di Slammer, la palma di virus più veloce era posseduta da Code Red, che era riuscito a infettare 359.000 sistemi in poco più di 19 ore.

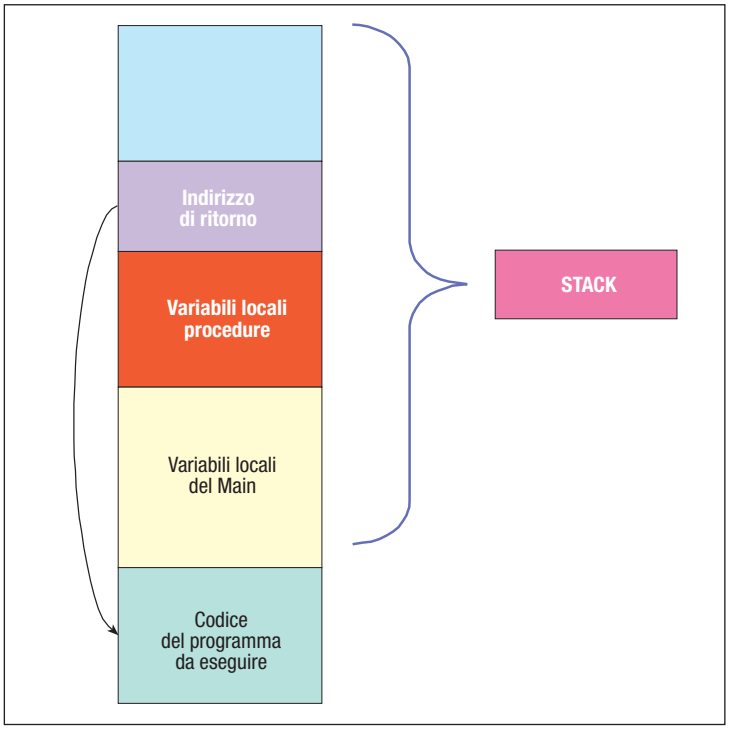


FIGURA 5
Schema di allocazione di un programma nella memoria centrale, dopo la chiamata di una procedura secondaria

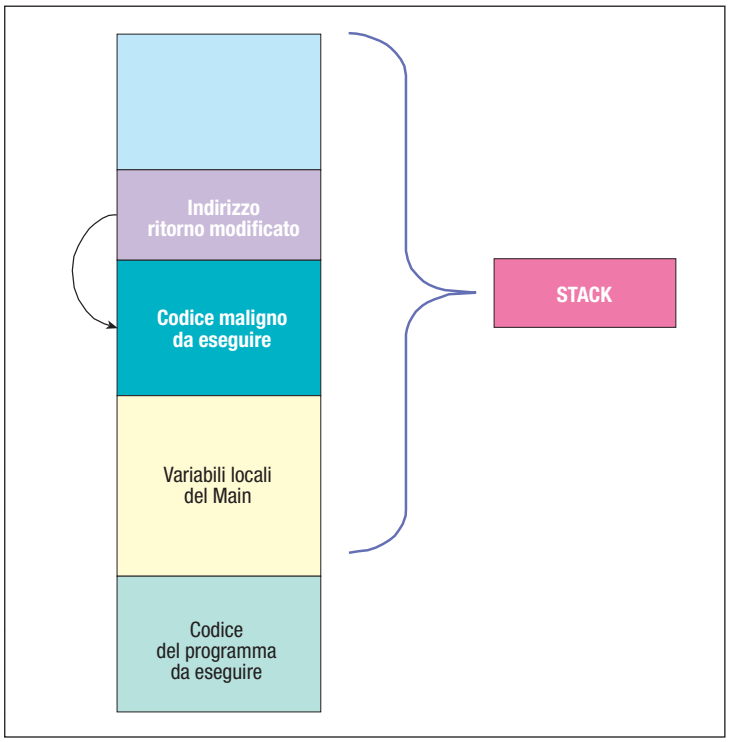


FIGURA 6
Schema di modifica della configurazione dello stack per consentire l'esecuzione di codice estraneo a quello del programma originale

di questo tipo applicato a una componente dei sistemi operativi Windows 2000 e Windows XP.

5. CONCLUSIONI

Con questo contributo l'autore spera di avere consentito al lettore di cogliere come la natura del problema sicurezza informatica sia da ricercarsi nei limiti naturali di chi su questi sistemi opera, e che nelle fasi di progettazione, realizzazione e uso dei programmi e delle tecnologie ICT commette inesorabilmente errori, solitamente dettati da una parziale conoscenza del problema o sottovalutazione di alcuni aspetti. Questo significa che la sicurezza dei sistemi ICT non può essere raggiunta attraverso l'adozione di opportune tecnologie o metodologie. Queste possono semmai contribuire ad alleviare il problema ma non a risolverlo. In realtà, chi scrive è fermamente convinto che esisterà sempre un problema sicurezza informatica, anche se l'adozione di nuove tecnologie potrebbe in un futuro non molto lontano consentire la realizzazione di sistemi molto più robusti di quelli attuali.

Bibliografia

[1] Bruschi D., Rosti E.: *A tool for pro-active defense against the buffer overrun attack*. Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, 2003.

[2] Bruschi D., Ornaghi A., Rosti E.: *Secure-ARP*. Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, 2003.

[3] Cert: http://www.cert.org/stats/cert_stat.html#vulnerabilities

[4] McLean J., "Security models". In Marciniak J: *Encyclopedia of Software engineering*. John Wiley &Sons, New York 1994.

[5] RFC2350: *Best Current Practice*.

[6] RFC2828: *Internet Security Glossary* by R. Shirey. May 2000.

[7] Schell R.G.: *Computer Security: the Achilles' heel of the electronic air force*. http://www.airpower.af.mil/airchronicles/aureview/1979/jan-feb/schell.html

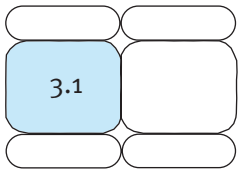
[8] John D. Howard, Thomas A. Longstaff: *A Common Language for Computer Security Incidents*. Sandia National Laboratories.

DANILO BRUSCHI è professore ordinario di Sicurezza delle reti e dei calcolatori presso il Dipartimento di Informatica e Comunicazione dell'Università degli Studi di Milano. È membro del Comitato Tecnico Nazionale per la Sicurezza Informatica e delle Telecomunicazioni per la Pubblica Amministrazione. È socio fondatore e Presidente dell'Associazione Italiana per la Sicurezza Informatica (CLUSIT).
bruschi@dsi.unimi.it

DAL COMPUTER CLASSICO A QUELLO QUANTISTICO: REALIZZABILITÀ E POTENZIALI APPLICAZIONI

Ernesto Hofmann

Gli odierni computer non sono altro che realizzazioni fisiche della macchina di Turing universale. Pur con delle sostanziali differenze, anche il più semplice PC può affrontare, seppur più lentamente, qualsivoglia problema risolubile da un supercomputer. Il computer quantistico è, invece, una macchina del tutto diversa, che, utilizzando i principi della meccanica quantistica potrebbe affrontare problemi che, anche in linea di principio, sarebbero insolubili per qualunque computer classico.



1. INTRODUZIONE

Qualche mese fa, su questa stessa rivista (Mondo Digitale n. 1, 2003, "Quantum Computing: sogno teorico o realtà imminente"), Emanuele Angeleri ha fornito un'affascinante descrizione del computer quantistico fornendo gli elementi fondamentali necessari per comprenderne il funzionamento.

Questo articolo può essere considerato un'ideale continuazione dell'analisi di Angeleri con l'obiettivo di approfondire alcuni aspetti più specifici, quali soprattutto le attuali tecnologie costruttive e alcune significative applicazioni.

Tutti hanno più o meno un'idea di come sia fatto, almeno esternamente, un computer. Dai grandi computer spesso apparsi in film famosi, come *2001 Odissea nello spazio*, ai *personal computer*, presenti ormai quasi in ogni casa, l'idea del computer si è sempre più diffusa nell'immaginario collettivo; e i computer, in tale immaginario, sembrano più o meno tutti simili, dimensioni a parte.

Ma se si potesse osservare un computer quantistico che cosa si vedrebbe? Certamente qualcosa di molto diverso da un computer tradizionale. Probabilmente si riconoscerebbero ancora uno schermo e una tastiera, ma il resto sarebbe molto differente.

Si vedrebbero dispositivi dalle forme inconsuete, come generatori di onde elettromagnetiche o di impulsi *laser* o, ancora, complessi dispositivi di raffreddamento. E i circuiti del computer quantistico, se di circuiti si può parlare, sarebbero anch'essi profondamente diversi. La maggior parte dei prototipi di circuiti quantistici, finora realizzati, sono aggregati di atomi o molecole, talora sospesi nel vuoto o immersi in sostanze liquide, e sottoposti a campi magnetici o a radiofrequenze.

Nulla di simile, quindi, a un tradizionale *chip* all'interno del quale circolano incessantemente microscopiche correnti. Ma la differenza è, in realtà, più profonda. Il computer quantistico non è un'evoluzione di quello classico ma una macchina del tutto diversa.

2. COME NASCE IL COMPUTER QUANTISTICO

Il computer classico è una macchina in grado di simulare la realtà con un certo grado di approssimazione. La modellazione aerodinamica, la progettazione di nuovi materiali, la bioinformatica consentono veri e propri esperimenti virtuali utili per comprendere i meccanismi della natura. La simulazione della realtà per mezzo del computer, dopo l'ipotesi teorica e l'esperimento, diventa il terzo pilastro della conoscenza scientifica.

Nel 1981, al *Massachusetts Institute of Technology* (MIT) si tenne un convegno che sarebbe stato il primo sul rapporto che esiste tra fisica e computazione. Richard Feynman, probabilmente il più grande fisico teorico del XX secolo, dopo Einstein, presentò una memoria dal titolo "Simulating Physics with Computers" [5]. Feynman non vedeva nulla di particolarmente eclatante nelle simulazioni approssimate della realtà fatte fino ad allora dai computer.

Era, invece, interessato alla possibilità di ottenere una simulazione esatta attraverso un computer che potesse fare le stesse cose che fa la natura.

Feynman già intuiva che la computazione non era solo una disciplina matematica ma anche fisica. La simulazione di un fenomeno sul computer classico richiede un mondo prevedibile in modo deterministico. Per esempio, in una partita a biliardo i movimenti delle palle obbediscono a leggi newtoniane ben note.

Per ogni causa (colpo) c'è un ben prevedibile effetto (traiettoria). Ci sono certamente limiti alla precisione della simulazione, ma questi limiti nascono dalla nostra ignoranza di ogni singolo elemento e possono, comunque, essere indefinitamente perfezionati.

Anche con un progressivo perfezionamento ingegneristico i circuiti elettronici operano, invece, sempre nello stesso modo e a fronte degli stessi dati in ingresso producono costantemente gli stessi risultati: 2 + 2 è sempre uguale a 4. Non ci sono incertezze nel comportamento di circuiti costituiti da miliardi di trilioni di atomi ed elettroni. "Ma un computer tradizionale fino a che punto può emulare il mondo quantistico?" si domandava Feynman, e aggiungeva: " ...non sono contento con le analisi fin qui fatte con la teoria classica perché la natura non è classica e se si vuole simulare la natura è meglio farlo quanto-meccanicamente, il che non sarà così facile".

3. COS'È IL MONDO QUANTISTICO

A chi non è capitato, guardando attraverso il vetro di una finestra, di vedere non solo il paesaggio esterno ma spesso anche la propria immagine, più o meno nitidamente?

E se si guardasse dall'altra parte forse si vedrebbe lo stesso paesaggio parzialmente riflesso. Questo fenomeno, troppo spesso osservato con indifferenza, è in realtà uno straordinario esempio alla portata di chiunque per entrare direttamente in contatto con il mondo quantistico.

La luce, si sa, è costituita di fotoni. Questi ultimi attraversano il vetro per mostrare il paesaggio; ma non è detto. Il mondo quantistico delle particelle elementari, come appunto i fotoni, non è un mondo di certezze ma di possibilità. Il fotone che colpisce il vetro può attraversarlo, ma può anche esserne riflesso: il fotone ha una certa probabilità di passare o meno attraverso il vetro. Il fenomeno è ancora più sottile e sfuggente per la logica aristotelica cui si è abituati: il fotone *passa e non passa*. È questo il senso di uno dei pilastri concettuali della meccanica quantistica: il *principio di sovrapposizione*. Se un'entità quantistica può assumere due valori o essere in due stati sarà in una sovrapposizione dei due, con una probabilità non nulla di essere nell'uno e nell'altro. In una sovrapposizione, a differenza di un miscuglio, non si può dire che un'entità si trovi realmente in uno stato o in un altro che però non si conoscono; la sovrapposizione contiene, invece, tutti i possibili casi, ma non equivale ad alcuno di essi.

A ogni particella si può poi associare un'onda, e ogni onda è una manifestazione di una particella. Max Born intuì per primo la natura di questa relazione: l'onda associata a una particella è un'onda di "probabilità", nel senso che indica quale sarà l'evoluzione possibile per quella particella. Lo stato di una particella non è più quello classico (posizione nel-

lo spazio e nel tempo e velocità). Lo stato di una particella è dato dalla sovrapposizione di tutti i suoi possibili stati futuri, ciascuno "pesato" con una probabilità. Si è lungamente riflettuto sul significato di una simile concettualizzazione: che cosa significa affermare che lo stato di una particella è un insieme di possibili stati? Il fotone che incide sul vetro passa o non passa? Un elettrone è qui o là? Nel mondo quantistico l'elettrone è sia qui sia là, ma con diverse probabilità di essere qui e là. Soltanto dopo una misura si può dire se sia qui. Tuttavia se si cerca di misurare una quantità di un sistema si fa collassare la funzione d'onda del sistema, e si ottiene un valore preciso per una quantità che prima era semplicemente una delle tante possibilità. È proprio l'osservazione che provoca la "scelta" di quel particolare valore fra tutti quelli possibili. Cosa causa il collasso di una funzione d'onda? La risposta a questa domanda è molto complessa ma nell'economia di questo articolo ci si può limitare, in maniera molto approssimativa, a rispondere che è l'*interferenza* tra il mondo quantistico e il mondo macroscopico.
Se, poi, a seguito di un certo fenomeno fisico, nascono due particelle esse saranno correlate tra loro come due gemelli omozigoti. In linea di principio, non si sa quali siano certe caratteristiche di una particella fino a quando non vengono misurate, e ciò vale ovviamente anche per l'altra. Ma se viene misurata una caratteristica di una particella immediatamente si determina l'analoga caratteristica dell'altra, dovunque essa si trovi nell'universo.
Detto così potrebbe sembrare quasi ovvio. Ma il fenomeno è molto più sottile. Occorre riflettere sul fatto che ciascuna particella è costantemente in una sovrapposizione di stati. Si potrebbe immaginare la situazione seguente. Due gemelli prima di uscire di casa si mettono un paio di calze; nel cassetto ce ne sono due paia: uno rosso e uno blu. Se si incontra un gemello e non si guarda sotto i suoi pantaloni le calze possono essere sia rosse sia blu: ma se si guarda sotto i pantaloni esse avranno un preciso colore e, quindi, si conoscerà anche il colore delle calze dell'altro gemello. Il mondo quantistico però non funziona in questo modo. Ciascun gemello sembra indossare entrambe le paia di calze e solo quando si va a verificare si costringe il gemello a sceglierne un paio: ciò immediatamente determina il colore delle calze dell'altro gemello. È il principio dell'*entanglement*, ossia della correlazione quantistica.
Si vedrà più avanti che *sovrapposizione, entanglement e interferenza* sono i tre pilastri alla base del funzionamento del computer quantistico.

4. PERCHÉ FEYNMAN AVEVA RAGIONE

Si immagini, allora, di avere una decina di particelle dotate individualmente di *spin*. Lo spin è una tipica grandezza quantistica che rappresenta una particolare caratteristica delle particelle elementari. Queste ultime potrebbero essere immaginate, in maniera quanto mai semplificata, come trottoline che ruotano intorno a se stesse. Tale rotazione può avvenire in senso orario o antiorario e può, quindi, rappresentare un bit; per esempio 0 per spin orario e 1 per spin antiorario. Si cerchi allora di contare i possibili stati in cui possono trovarsi le particelle stesse. Se fossero tutte nello stato di spin orario si avrebbero 10 bit a zero, se invece fossero tutte con spin antiorario si avrebbero 10 bit tutti a 1. In realtà, sono possibili anche tutte le combinazioni intermedie e, quindi, si hanno complessivamente 1024 possibili combinazioni di bit, ossia 2^{10}. Se le particelle fossero 20 si avrebbe 2^{20}, ossia oltre un milione di combinazioni; e se fossero 40 si avrebbe 2^{40}, ossia mille miliardi di combinazioni.
Le cose sono però ben più complicate. Infatti, la nostra semplice analisi ha considerato solo la possibilità che le particelle possano essere dotate di spin orario o antiorario.
Invece, secondo la meccanica quantistica lo spin può essere in uno stato di sovrapposizione, ossia in una qualsivoglia combinazione delle due direzioni, per esempio il 30% orario e il 70% antiorario. L'intero sistema è, quindi, un aggregato incredibilmente complesso di sovrapposizioni di tutte le possibili combinazioni di spin di ciascuna particella. L'evoluzione di un simile sistema, descritta da una complessa funzione d'onda probabilistica, è un problema non trattabile neppure oggi da qualsivoglia supercomputer.

Ecco allora l'idea di Feynman.
Cosa accadrebbe se la simulazione del sistema non fosse condotta su di un computer classico ma su di un computer che funzionasse anch'esso secondo le leggi della meccanica quantistica? I circuiti e i bit di memoria di quest'ultimo non sarebbero vincolati a una dualità di valori 0 e 1 ma potrebbero anch'essi operare in una sovrapposizione di stati 0 e 1.
Feynman, in realtà, nel 1981, non andò molto al di là di questa intuizione. Ma nello stesso convegno un altro fisico, Paul Benioff, presentava un modello di computer quantistico basato su di un'ipotetica macchina di Turing che funzionava eseguendo una sequenza di operazioni effettuate secondo le leggi della meccanica quantistica [1].
Il modello di Benioff era anch'esso molto approssimativo nei dettagli costruttivi, ma era sufficientemente solido dal punto di vista concettuale, tanto che tre anni dopo lo stesso Feynman ne presentava una propria versione semplificata e migliorata (senza peraltro citare il contributo di Benioff). Comunque anche il nuovo modello di Feynman restava ancora un astratto strumento concettuale. La differenza tra i modelli di Benioff/Feynman e un reale computer quantistico era abbastanza simile a quella che c'è tra una macchina di Turing e un personal computer.
La vera svolta, ancora concettuale, doveva avvenire solo un anno dopo con la pubblicazione di un nuovo articolo, di David Deutsch [3].
Come il computer di Benioff anche quello di Deutsch era basato su di una macchina di Turing: il programma era memorizzato insieme ai dati sul nastro. Il processore avrebbe eseguito il programma utilizzando operazioni di tipo quantistico. Tali operazioni, come in un computer classico, avrebbero agito sui bit del nastro. Poco veniva detto su come tale computer potesse essere realizzato praticamente, perché, in realtà, l'obiettivo era quello di esaminare la struttura concettuale di una simile macchina. Il grande passo avanti era costituito dalla dimostrazione dell'universalità di un simile modello. Deutsch mostrava come fosse possibile costruire una struttura logica in grado di eseguire qualunque tipo di calcolo. Il principio base di ogni computer classico è che ogni funzione che può essere calcolabile può essere calcolata da una macchina di Turing. Ma ora, secondo Deutsch, esisteva un computer universale in grado di effettuare qualunque tipo di calcolo che qualsivoglia entità fisica fosse in grado di attuare. La computazione non era più un ramo della sola matematica ma diventava una manifestazione delle leggi della fisica.
E poiché il mondo quantistico è intrinsecamente parallelo è possibile fare di conto in maniera completamente diversa rispetto alle tecniche tradizionali.
Si immagini, diceva Deutsch, di avere una funzione in grado di predire l'andamento del mercato azionario. Si immagini anche che occorra valutare tale funzione per soli due valori, 0 e 1, ossia $f(0)$ e $f(1)$. La funzione può, comunque, essere molto complicata e, quindi, si può anche immaginare che a un computer classico occorrano 24 h per calcolare la funzione stessa: quindi 48 h per verificare se $f(0) = f(1)$. Se però, utilizzando il parallelismo quantistico, fosse possibile calcolare contemporaneamente $f(0)$ e $f(1)$ si potrebbe sapere se $f(0) = f(1)$ in 24 h e, quindi, in tempo utile per investire con successo.

5. UNA NUOVA STRUTTURA FISICA

Si immagini di avere un atomo che abbia un solo elettrone nell'ultima orbita occupata. Questo elettrone può essere spostato, ossia "eccitato", in un'orbita più esterna illuminandolo con una luce di una determinata frequenza e durata.
L'elettrone fa così un salto quantico in uno stato di energia più elevata. Se tale stato è sufficientemente stabile lo si potrà utilizzare, insieme allo stato di energia più basso, per rappresentare rispettivamente i numeri 0 e 1.
Se un atomo "eccitato" viene colpito da un ulteriore impulso di luce, simile al precedente, l'elettrone ritorna nello stato di energia più bassa rilasciando un fotone.
Ma cosa accade se la durata del primo impulso di luce dura la metà del tempo necessario per commutare lo stato dell'elettrone? La risposta è sorprendente per la logica cui si è abituati: l'elettrone si troverà simultaneamente in entrambe le orbite. L'elettrone sarà allora in una "sovrapposizione" dei due stati, fondamentale ed eccitato.

Utilizzando in tal modo un atomo si può memorizzare un'unità di informazione, ossia un bit. Nel 1995, Ben Schumacher coniò il termine "qubit" (*quantum bit*) per denotare tale entità. Un bit digitale se viene misurato può essere solo 0 o 1, con certezza, mentre un bit analogico può assumere qualsivoglia valore tra 0 e 1. Un qubit è, invece, una "sovrapposizione" di 0 e 1 e può essere definito dalla notazione matematica a $|0> + b|1>$, intendendo con ciò che se misurato esso potrà valere 0 con probabilità $|a|^2$ e 1 con probabilità $|b|^2$, essendo *a* e *b* numeri complessi.
Non si vuole, per semplicità, approfondire la natura matematica degli stati rappresentati dal simbolo |>, ma è bene comunque ricordare che tale simbolo sta a rappresentare un vettore, per sua natura orientato. Lo stato $|1> - |0>$ è diverso dallo stato $|1> + |0>$, come si può vedere dalla figura 1. Lo stato di un qubit $a|0> + b|1>$ può essere rappresentato da un vettore che raggiunge un qualunque punto di una circonferenza. Nel disegno *a* e *b* sono numeri reali; se fossero numeri complessi (come in realtà sono) il disegno dovrebbe essere in tre dimensioni, ossia una sfera.
Da qui in avanti, per semplicità, non si useranno più le ampiezze di probabilità *a* e *b* assumendole eguali a $1/\sqrt{2}$ e ritenendo, quindi, i due stati $|0>$ e $|1>$ equiprobabili. Se due qubit vengono indipendentemente posti nella sovrapposizione $|0> + |1>$ si scriverà $(|0> + |1>)(|0> + |1>)$ ossia $|00> + |01> + |10> + |11>$. Si vede così che due qubit possono rappresentare "contemporaneamente" 4 valori, mentre 3 qubit ne rappresentano 8, $|00>$, $|001>$, ...$|111>$.
Ma come si possono modificare i valori memorizzati? In altri termini che tipo di operatori si possono utilizzare per modificare il contenuto di uno o più qubit? La domanda è, in realtà, ancora più generale.
In ambiente quantistico si è già visto come sia possibile invertire il contenuto di un qubit. L'impulso di luce di durata opportuna equivale a tutti gli effetti a un operatore NOT. Ma si è anche visto che illuminando l'atomo per metà tempo (rispetto a quello necessario per commutare lo stato dell'elettrone) si ottiene una sovrapposizione di stati. Detto in altri termini, se il bit era $|0>$ esso diventa $|0> + |1>$. Se a questo punto

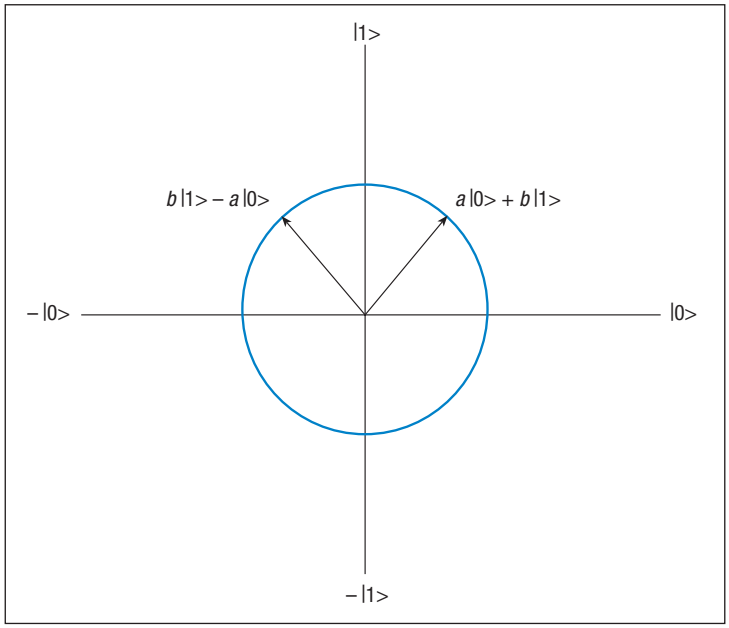


FIGURA 1

Lo stato di un qubit a $|0> + b|1>$ *può essere rappresentato da un vettore che raggiunge qualsivoglia punto di una circonferenza se le ampiezze a e b sono numeri reali e se la somma dei loro quadrati è uguale a* 1

riceverà un uguale impulso di luce passerà allo stato $|1>$. Un impulso completo equivale, quindi, all'operatore NOT. Ma uno di durata metà a cosa equivale? All'operatore radice quadrata di NOT. Infatti, due impulsi di questo tipo danno luogo a un:

$$\text{NOT} = \sqrt{\text{NOT}} \times \sqrt{\text{NOT}}$$

Oltre all'operatore $\sqrt{\text{NOT}}$ esistono in meccanica quantistica altri tipi di operatori, che possono essere applicati alle grandezze $|0>$ e $|1>$, e alle loro combinazioni $|0> + |1>$, $|0> - |1>$, .., per trasformarle opportunamente.
Uno di questi, che per semplicità non verrà esaminato matematicamente, è la cosiddetta trasformazione di Hadamard. Nell'attuale letteratura, l'operatore $\sqrt{\text{NOT}}$ viene indicato con *X* e l'operatore di Hadamard con *H*.
L'effetto di *X*, come si è visto, è $X\ |0> \tau \neq (|0> + |1>)$; e $X\ (|0> + |1> \tau \neq |1>$; mentre $X\ |1> \tau (|1> - |0>)$; $X\ (|1> - |0>) | \tau \neq \neq - |0>$.
L'effetto di *H* è $H\ |0> \neq \tau \neq (|0> + |1>)$; e $H\ (|0> + |1>)\ \tau \neq |0>$; mentre $H\ |1> \neq \tau \neq (|0> - |1>)$ e $H\ (|0> - |1>)\ \tau \neq \neq |1>$.
Ora si è finalmente in grado di esaminare il circuito quantistico proposto da Deutsch co-

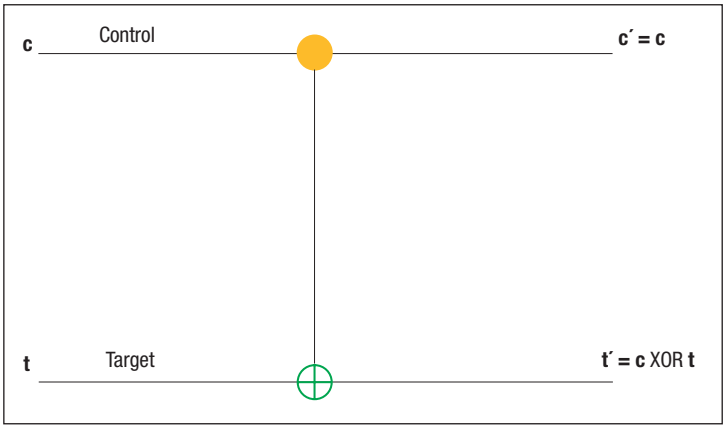


FIGURA 2
Nel circuito di OR esclusivo (XOR) si può vedere che il valore di c resta inalterato mentre il valore di t viene invertito se c è uguale a 1

me circuito universale per la costruzione di un intero computer quantistico.
Nell'articolo di Deutsch *c* è il control input, mentre *t* è il target input. Dalla figura 2 si può comprendere il significato dei due input.
Il circuito stesso prende il nome di *controlled not gate* e può essere schematizzato come segue:
Se $c = |0> + |1>$ e $t = |0>$ il risultato finale sarà $|00> + |11>$ e non $c = |0> + |1>$ e $t = |0> + |1>$.
Ciò vuol dire che se si misurerà *c* e si otterrà 0 anche *t* sarà a 0; viceversa se misurando *c* si otterrà 1 *t* sarà anch'esso a 1. Questo perché il valore 0 di *c* non fa commutare *t*, mentre il valore 1 lo fa commutare.
Ecco che finalmente entra in gioco il principio dell'entanglement visto precedentemente.
Se due qubit sono entrambi nella sovrapposizione di 0 e 1 vengono definiti entangled se il risultato della misurazione di uno di essi è sempre correlato al risultato della misura dell'altro qubit.
L'entanglement, insieme alla sovrapposizione, è la chiave di volta dell'intero funzionamento del computer quantistico. Senza l'entanglement, infatti, come si potrebbero correlare i risultati ottenuti con i valori in ingresso? Per comprendere più facilmente questo fondamentale concetto si può ricorrere a una semplice metafora. Si immagini di avere un insieme di domande, quali per esempio la moltiplicazione di diverse coppie di numeri molto grandi, e di distribuire tali moltiplicazioni tra più persone. Ciascuna di queste trascriverà il proprio risultato su di un foglietto che porrà in una scatola. La scatola in questo esempio rappresenta il registro di qubit in uscita. Estrarre di volta in volta dalla scatola un risultato equivale a far "collassare" il registro dei qubit a un valore preciso dopo una misura. Ma il risultato ottenuto a quale domanda, ossia a quale moltiplicazione, corrisponde se sul foglietto è scritto solo il risultato? Nel computer quantistico è proprio il meccanismo dell'entanglement che consente di associare i singoli risultati alle rispettive domande. Allo stesso tempo il principio dell'interferenza fa in modo che se viene estratto un foglietto con un risultato vengono contemporaneamente distrutti tutti gli altri.
Con i tre fondamentali meccanismi della sovrapposizione, dell'entanglement e dell'interferenza è possibile costruire un'intera logica circuitale quantistica, almeno a livello concettuale, con la quale si può mettere in luce la straordinaria capacità di calcolo di un computer quantistico.
Deutsch nel suo articolo si proponeva di calcolare in tempo utile una particolare funzione per due soli valori $f(0)$ e $f(1)$ per verificare se $f(0) = f(1)$.
Nel riquadro viene riportata in maniera semplificata una dimostrazione data da Artur Ekert.
A questo punto sorgono quasi spontaneamente due domande:
- quali classi di problemi può affrontare un computer quantistico ?
- come può essere fisicamente costruito un computer quantistico?

6. QUALI CLASSI DI PROBLEMI PUÒ AFFRONTARE IL COMPUTER QUANTISTICO

Da quanto detto finora, si può già intuire che il computer quantistico non è il computer tipico per navigare in Internet o per inviare la posta elettronica. A cosa può servire allora?
Si è già osservato che, secondo il dogma fondamentale della computazione, nell'ambito del calcolo non esiste alcuna macchina concettualmente più potente della macchina di Turing.
Dal punto dei vista della computabilità un algoritmo è caratterizzato anche dal numero di operazioni e dalla quantità di memoria richieste per un input di dimensioni *x*. Questa caratterizzazione dell'algoritmo determina quella che viene definita la complessità del-

l'algoritmo stesso. Tra i problemi considerati complessi ci sono certamente quelli che crescono come la potenza di un numero. La funzione $y = x^2$ cresce molto rapidamente. Per valori di x molto elevati occorre eseguire moltiplicazioni sempre più laboriose.
Se la potenza cresce ulteriormente, per esempio $y = x^4$ o $y = x^5$ la difficoltà aumenta ancora. Simili problemi, definiti polinomiali, sono oggi alla portata dei computer classici. Ma esistono problemi che crescono molto più rapidamente di quelli polinomiali. I problemi di tipo esponenziale aumentano di complessità più rapidamente di quelli polinomiali: e^x cresce molto più rapidamente di x^3, x^5, x^7,.. per valori crescenti di x.
La distinzione oggi più utilizzata è quella tra problemi che possono essere risolti in modo polinomiale (*P*), e considerati *trattabili*, e quelli che, invece, non possono essere risolti in modo polinomiale, e che vengono generalmente considerati *intrattabili*, e che possono a loro volta far parte di classi diverse. Tra queste ultime la prima è la cosiddetta classe NP. Semplificando, i problemi di tipo NP non possono, in generale, essere risolti da algoritmi deterministici di tipo polinomiale, e sono, quindi, in linea di principio intrattabili. NP, infatti, sta a significare *non-deterministic polynomial time*. Non deterministico significa che a un dato passo dell'algoritmo non si può stabilire in maniera univoca quale possa essere il passo successivo. Un pò come nel gioco degli scacchi: data una mossa dell'avversario non c'è al momento un algoritmo che possa, a priori, determinare deterministicamente, in tempo ragionevole, quale debba essere la mossa successiva.
Esistono ulteriori problemi, definiti NP-completi, che fanno parte di NP ma sono raggruppabili in gruppi tali che se si risolve un problema in tempo polinomiale allora tutto il gruppo è solubile.
Tra questi problemi rientra il celebre problema del commesso viaggiatore che debba visitare un certo numero di città, ciascuna una volta sola e senza tornare mai indietro, attraverso un percorso che abbia la lunghezza minima.
Per riassumere quanto detto finora si può dire che gli algoritmi possono essere classificati come:

- P: polinomiali: (per esempio, la moltiplicazione)
- NP: polinomiali non deterministici: (per esempio, la fattorizzazione)
- NP-completi: sottoclassi di NP tali che se uno della classe è trattabile lo sono tutti: (per esempio, il problema del commesso viaggiatore)

Sebbene le classi di tipo NP non siano le più complesse esse contengono comunque alcuni tra i problemi, al momento, di maggior interesse. Tra questi il problema della fattorizzazione di un numero è intimamente connesso con la possibilità di decrittare un sistema di crittografia, come per esempio il sistema RSA129 che utilizza chiavi costituite da 129 cifre. È stato valutato che, se per fattorizzare un numero di 129 cifre nel 1994 sono stati necessari 5000 MIPS-anni (MIPS: milioni di istruzioni al secondo), per fattorizzarne uno di 200 cifre occorrerebbero quasi 3 miliardi di MIPS-anni.
Ed è proprio nell'area di simili problemi che entra in gioco il computer quantistico.
Si è recentemente scoperto, per esempio, che proprio la fattorizzazione in fattori primi di numeri molto grandi può essere affrontata con successo da un ideale computer quantistico, che usi quindi sovrapposizione ed entanglement, con un metodo, detto algoritmo di Shor, ideato da Peter Shor nel 1994[1]. L'algoritmo di Shor è matematicamente abbastanza semplice e richiede un hardware quantistico abbastanza modesto, almeno per piccoli numeri.
Ed è interessante riflettere su quanto ha detto proprio Deutsch in merito. "... *vorrei lanciare una sfida a coloro che sono ancora attaccati alla concezione classica: spiegare come funziona l'algoritmo di Shor. Non si tratta soltanto di dimostrare che è valido – a tal fine basta semplicemente risolvere alcune equazioni – ma di fornire una vera spiegazione. Quando l'algoritmo di Shor fattorizza un numero utilizzando una quantità di risorse computazionali pari a circa* 10^{500} *volte quelle apparentemente presenti, dov'è il numero fattorizzato? Il numero degli atomi presenti in tutto l'universo visibile è all'incirca* 10^{80}, *un numero estremamen-*

[1] Bone S, Castro M: *A Brief History of Quantum Computing*. http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/spb3

te piccolo in confronto a 10^{500}. *Quindi, se l'universo visibile esaurisse la realtà fisica in tutta la sua estensione, questa non conterebbe che una piccola frazione delle risorse necessarie per fattorizzare un numero tanto grande.Chi l'ha fattorizzato allora? Come, e dove, è stata effettuata la computazione*?.." [3]
Senza voler ulteriormente approfondire il tema si può osservare che il computer quantistico potrebbe, forse, essere in grado di trasformare i problemi di tipo NP in problemi di tipo P. Se così fosse ci si troverebbe di fronte a una scoperta straordinaria.
Ma come sarà possibile costruire fisicamente un computer quantistico?

7. COME COSTRUIRE UN COMPUTER QUANTISTICO

Nel 1994, si tenne a Boulder (Colorado), nel laboratorio del *National Institute of Standards&Technology* (NIST) un convegno internazionale sulla fisica atomica.
Nel corso del convegno Artur Ekert, che aveva più volte collaborato con Deutsch, sostenne che era giunto il momento di lanciare la rivoluzione del computer quantistico. Quest'ultimo avrebbe avuto bisogno di molteplici circuiti quantistici in grado di operare in sinergia. Perché non cominciare allora a costruire almeno un "controlled NOT quantum gate"?
Si è visto prima come un singolo atomo che venga eccitato o meno possa funzionare concettualmente come un NOT (o anche come un $\sqrt{\text{NOT}}$).
C'è da aggiungere che già da molti anni si era riusciti, grazie soprattutto alle ricerche di Hans Dehmelt (vincitore per questo del premio Nobel per la Fisica nel 1989) a isolare in una camera a vuoto un singolo ione, ossia un atomo con una piccola carica elettrica, dovuta per esempio al fatto di aver "strappato" all'atomo un elettrone.
Utilizzando con grande maestria campi elettrici e magnetici era possibile isolare un unico ione e muoverlo all'interno di un opportuno dispositivo in grado di mantenere uno stato di vuoto pressoché assoluto, quasi come se fosse una mosca all'interno di una bottiglia.
Bombardando lo ione da ogni direzione con impulsi laser lo si può sospendere praticamente in un punto. I ricercatori Cirac e Zoller (dell'università di Innsbruck) intuirono che un simile ione potesse funzionare come un *quantum gate*[2]. Si immagini che lo ione abbia un solo elettrone nell'orbita più esterna. Se tale elettrone è nello stato energetico più basso si avrà uno 0, se sarà in uno stato energetico più alto si avrà un 1.
Basterà un opportuno impulso laser per far commutare lo ione da uno stato a un altro.
Si immagini ora di costruire, come se fosse un filo di perle, una catena di ioni adiacenti; catena mantenuta stabile da opportuni campi elettromagnetici. Le singole cariche degli ioni tenderanno a respingerli reciprocamente mentre i campi elettromagnetici tenderanno a mantenerli raggruppati.
Si potrebbe immaginare il tutto come un insieme di minipendoli affiancati l'uno all'altro (Figura 3).
È bene riflettere su di un aspetto importante. Il movimento orizzontale dei minipendoli non è simile a quello di pendoli reali perché a livello atomico il movimento dei singoli ioni è quantizzato anch'esso. Ciò vuol dire che i minipendoli non potranno vibrare con qualunque tipo di frequenza ma solo nell'ambito di precise frequenze intervallate l'una dall'altra in maniera discreta, ossia non continua.
Le conseguenze sono molto interessanti. In-

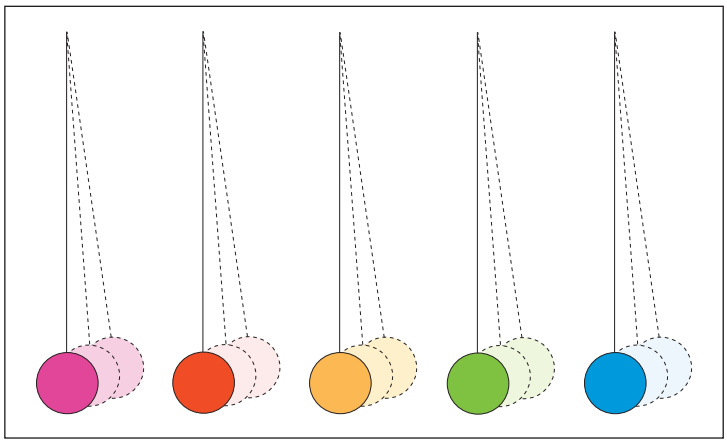


FIGURA 3
Un gruppo di ioni che oscillano: come pendoli essi possono oscillare all'unisono

[2] Cirac J, e al.: Quantum gates and Quantum Computation with Trapped Ions. (cap 4.3) in Bouwmeester D, Ekert A, An Zeilinger: The Physics of Quantum Computation [2].

fatti, tutto ciò che può assumere due stati ben distinti tra loro può essere considerato un bit - in questo caso un qubit - di informazione.
Il risultato complessivo di Cirac e Zoller è che diventa possibile costruire un registro quantistico con due tipi distinti di informazioni memorizzabili: il livello energetico dell'elettrone e la vibrazione orizzontale dello ione. Ciò che Cirac e Zoller avevano individuato era un metodo per ottenere operazioni quantistiche attraverso le interazioni dei qubit.
Si supponga, per esempio, che uno degli ioni sia in uno stato eccitato (1) e che stia vibrando con la frequenza base (0). Il dispositivo così realizzato è un miniregistro quantistico che contiene l'informazione 10. Con un opportuno impulso laser l'elettrone può essere fatto ritornare nello stato base mentre allo stesso tempo lo ione può essere fatto vibrare con frequenza 1. Il risultato sarà che 10 diventerà 01. Utilizzando opportunamente ulteriori impulsi il bit 1 può essere fatto viaggiare da ione a ione come se la catena di ioni fosse un vero e proprio bus di trasferimento.
Ma la cosa più importante è che il modo con il quale uno ione risponde a un impulso laser può dipendere dalla vibrazione delle catene di ioni.
Cirac e Zoller mostrarono come lo stato 01 potesse diventare 00, 10, 11. In conclusione, furono in grado di costruire un "controlled NOT gate".
Il dispositivo di Cirac e Zoller era basato su di una catena di ioni affiancati. Ma già nel 1995 due ricercatori del NIST, David Wineland e Christopher Monroe, riuscirono a costruire il primo vero "two-bit controlled NOT gate" utilizzando un solo ione di berillio (Be+)[3].
Isolarono uno ione di Be+ con un solo elettrone nell'orbita più esterna e, invece di utilizzare l'eccitazione di tale elettrone in un'orbita più esterna, sfruttarono i livelli di energia "iperfine" che dipendono dall'allineamento tra lo spin dell'elettrone e quello del nucleo dello ione.
Usando i due stati di energia iperfine (spin allineati o disallineati) e due stati dipendenti dai modi di vibrazione dello ione ottennero due qubit correlati.

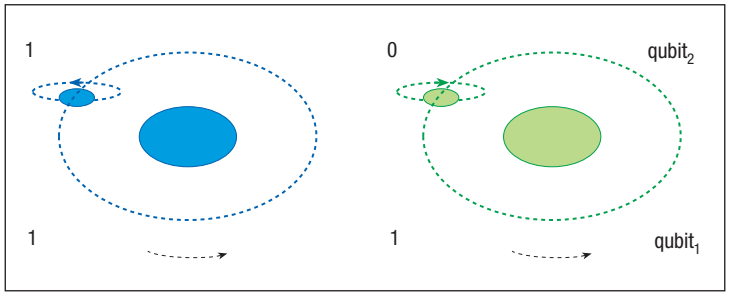


FIGURA 4
Un esempio di controlled NOT gate

Applicando opportuni impulsi laser furono in grado di dimostrare il funzionamento dello ione come "controlled NOT gate" (Figura 4).
Tra i due esperimenti di Cirac-Zoller e Wineland-Monroe ci sono però sostanziali differenze. Cirac e Zoller furono in grado di mantenere lo stato di sovrapposizione quantistica per almeno un decimo di secondo, ossia un tempo molto lungo su scala atomica.
Il gate di Wineland-Monroe era almeno cento volte meno stabile. Ma il problema era, in realtà, un altro. Per effettuare realmente dei calcoli occorre poter operare con moltissimi ioni. Wineland e Monroe sono stati in grado di arrivare a far cooperare 4 ioni, ma per eseguire i calcoli necessari alla scomposizione in fattori di un numero di molte cifre occorrono migliaia di qubit.
Il limite di *scaling*, ossia di capacità di crescita dimensionale, della tecnologia *ìon-trap* sembra in questo momento intorno ai 50 qubit. E soprattutto non si sa se un programma quantistico possa essere eseguito per il tempo necessario senza incorrere nel fenomeno della *decoerenza*. Uno dei problemi più complessi da risolvere è quello di impedire che l'interferenza dei vari calcoli si rifletta sul mondo macroscopico. Infatti, se un gruppo di atomi o di molecole è sottoposto a un fenomeno di interferenza e interagisce al tempo stesso con l'ambiente macroscopico non è più possibile rilevare l'interferenza con misure che riguardano solo gli atomi del gruppo originario che così cessa di effettuare un'attività di calcolo quantistico utile.
Una nuova interessante soluzione è sembrata arrivare nel 1997 con una tecnologia denominata NMR (*Risonanza Nucleare Magnetica*).
L'idea, in questo caso, è di utilizzare non ato-

[3] Monroe C, et al.: *Demonstration of a Fundamental Quantum Logic Gate* :http://qubit.nist.gov/

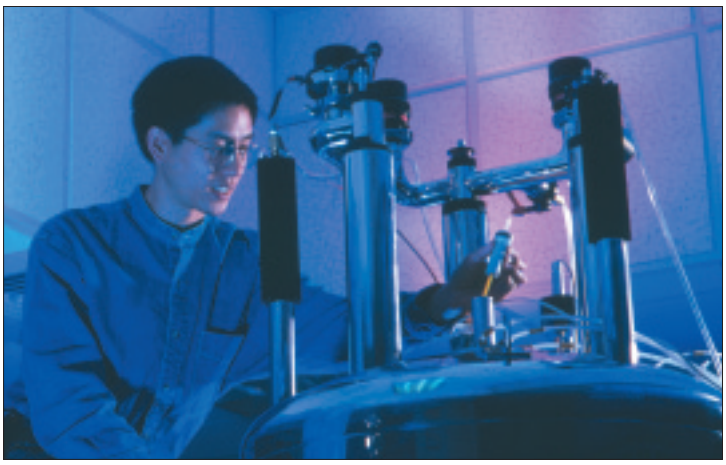


FIGURA 5
Isaac Chuang carica le molecole a 7-qubit nel dispositivo a risonanza nucleare magnetica

mi ma intere molecole. Una molecola di per sé è già una catena di atomi. Ma gli elettroni degli atomi non sono le sole entità che possono essere utilizzate come qubit. La risonanza nucleare magnetica dipende dal fatto che anche i nuclei possono essere utilizzati come gli elettroni. I nuclei sono aggregati di protoni e neutroni, ambedue dotati di spin. Tali spin si bilanciano più o meno tra loro. Ma se un nucleo è costituito di un numero dispari di protoni e neutroni potrà restare uno spin netto a 0 o a 1. Si è scoperto che tali spin possono essere usati anch'essi come qubit. Immersi in un campo magnetico i vari atomi di un nucleo (idrogeno, carbonio,..) rispondono ciascuno a un preciso impulso laser. Anche lo stesso atomo potrà rispondere in maniera diversa a seconda della sua posizione nella molecola. Per poter creare un circuito quantistico i nuclei devono anche essere in grado di interagire l'uno con l'altro attraverso i propri campi magnetici. Devono essere correlati (*entangled*). Che un nucleo possa commutare, o meno, il proprio stato a fronte di un impulso esterno può, quindi, dipendere dallo stato dei suoi vicini. Poiché gli atomi della molecola emettono anch'essi deboli segnali questi possono essere rilevati dall'esterno. Si immagini di avere una sostanza composta di molecole costituite di quattro atomi (*a*, *b*, *c*, *d*) che rappresentano quattro qubit. Tale sostanza viene diluita in un liquido che conterrà miliardi di trilioni di queste molecole. All'inizio i nuclei delle molecole punteranno in ogni direzione: un vero e proprio brodo di spin. Un intenso campo magnetico può però allinearne una piccolissima frazione che è pur sempre costituita di innumerevoli miliardi di molecole i cui nuclei saranno tutti allineati allo stesso modo, ossia 1111. Questo allineamento provoca un segnale complessivo che può essere letto da un operatore su di uno schermo. A questo punto, l'operatore può scegliere uno degli atomi - per esempio il secondo - e inviare un impulso che modificherà lo stato di tutti i secondi atomi di tali molecole. Ancora una volta, il segnale prodotto da tali molecole verrà letto sullo schermo come 1011.

La molecola può essere stata scelta in base al fatto, per esempio, che l'atomo *b* commuta solo se l'atomo *d* è a 1. Altrimenti l'impulso verrà ignorato.

Isaac Chuang (Figura 5), Gregory Breyta, Mark Sherwood e Costantino Yannoni, dei laboratori IBM di Almaden, insieme a Lieven M.K. Vandersypen e Matthias Steffen, della Stanford University, hanno presentato, nel 2001, i risultati di un loro esperimento relativo alla fattorizzazione del numero 15 con l'algoritmo di Shor[4]. Sono stati in grado di disegnare e costruire una nuova molecola con sette spin nucleari – cinque di fluoro e due di carbonio – in grado di interagire come qubit e di essere programmati per mezzo di una radiofrequenza. Il loro contenuto poteva essere quindi letto per mezzo di dispositivi tipici della tecnologia NMR.

Il calcolo è stato eseguito da 10^{18} molecole. Sebbene il calcolo possa sembrare banale la gestione contemporanea di 7 qubit costituisce alla data il calcolo quantistico più complesso finora eseguito.

Ma si ritiene che non sarà facile spingersi molto più oltre con la tecnologia NMR. Ogni volta che un nucleo viene aggiunto alla catena di molecole la voce elettromagnetica di ciascun qubit diventa più debole e difficile da sentire. Probabilmente non è possibile andare al di là di 50 qubit.

Sono state però sperimentate anche altre tecniche, come i cosiddetti *quantum dot* che sono piccole isole di materiale semiconduttore all'interno di un chip. Un tipico quantum-dot può essere costituito da poche centinaia

[4] Lieven M, e al.: *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*. Nature 414, pp. 883-887 (2001).

di atomi. Applicando opportuni campi elettrici attorno a queste isole è possibile controllare il numero di elettroni mobili all'interno delle isole stesse.
Si può fare in modo di intrappolare, all'interno di un'isola, un unico elettrone. Intrappolato in tal modo l'elettrone può occupare solo stati discreti (quantizzati) di energia. Ogni elettrone può, quindi, essere usato come un qubit. Le tecniche costruttive dei quantum dot sono diverse da quelle della fotolitografia tradizionale ma sono comunque accessibili, e recenti sviluppi hanno mostrato che è possibile costruire un notevole numero di quantum dot sullo stesso chip.
I qubit possono interagire attraverso fenomeni elettromagnetici o di *tunneling* quantistico, ed eseguire quindi operazioni logiche. Poiché gli elettroni possono avere lo spin orientato verso l'alto o verso il basso tale direzione può essere utilizzata come informazione di on-off, ossia ciascun dot può essere a uno o a zero a seconda dello stato di spin. Recentemente[5] alcuni ricercatori sono stati in grado di correlare due quantum dot, controllare quanti elettroni erano in ciascun dot e determinare lo stato di spin in ogni dot. A differenza di un computer classico nel quale la corrente circola attraverso i circuiti per trasferire le informazioni, nei computer basati su quantum dot il flusso delle informazioni è basato sulla manipolazione degli spin degli elettroni confinati nei dot.
Molti ritengono che di tutte le tecnologie quantistiche quella dei quantum dot sia la più promettente e, soprattutto, quella che ha la maggior capacità di *scalability*, fino a 1000 qubit e oltre.
Purtroppo il tempo di decoerenza è molto breve – dell'ordine del milionesimo di secondo – e i chip stessi per operare devono essere mantenuti a temperature prossime allo zero assoluto. Ma qualunque tecnologia dovesse affermarsi dovrà comunque risolvere il problema degli errori che possono nascere per qualunque disturbo dal mondo esterno il sistema quantistico possa avvertire. Come trattare tali errori è materia di ulteriore analisi che esula, tuttavia, dalle finalità di questo articolo.

8. QUALI PROSPETTIVE

Non è possibile al momento attuale formulare previsioni sull'effettiva capacità tecnica di costruire un computer quantistico in grado, per esempio, di scomporre in fattori primi un numero di almeno 10 cifre. Ci sono almeno tre tipi di problemi che occorre risolvere.
Innanzitutto, il mantenimento dello stato di sovrapposizione quantistica dei vari elementi, e quindi un effettivo isolamento dei circuiti quantistici dal mondo macroscopico che li circonda.
In secondo luogo, la gestione degli errori che comunque si manifestano in un complesso circuitale così delicato.
Infine, la sapienza costruttiva necessaria per realizzare le funzioni di calcolo che attraverso sovrapposizione, entanglement e interferenza consentono di creare risposte dalle domande e di correlare le prime alle seconde.
La strada da percorrere è enormemente complessa e non è nemmeno certo che sia realmente percorribile. Ma anche se alla fine il computer quantistico si rivelasse grande come un edificio e richiedesse centinaia di specialisti per essere programmato e gestito, e costasse centinaia di milioni di euro, o ancor più, esso potrebbe rivelarsi un formidabile strumento di calcolo.
Dal punto di vista strategico o economico potrebbe consentire di decifrare qualunque chiave crittografica o di investigare in tempi brevissimi qualunque archivio.
Ma è soprattutto dal punto di vista conoscitivo che esso consentirebbe di entrare realmente in un mondo, quello della meccanica quantistica, che oggi appare ancora quasi incomprensibile anche se viene utilizzato per le più complesse teorie della fisica.
Anche un piccolo computer quantistico, permettendo, di manipolare concretamente fenomeni come la sovrapposizione o l'entanglement, farebbe apparire la meccanica quantistica molto meno surreale di quanto non appaia oggi.
Il computer quantistico permetterebbe, infine, di capire meglio non solo la realtà del mondo subatomico ma anche il significato più profondo di ciò che è realmente la computazione e il suo ruolo nel mondo.

[5] Jeong H, e al.: *The Kondo Effect in an Artificial Quantum Dot Molecule*. Science 21 Sept. 2001.

La dimostrazione matematica di Ekert

Viene qui riportata la spiegazione matematica (semplificata) data da Ekert per spiegare il ragionamento di Deutsch.
Nel ragionamento di Deutsch il controlled Not gate è stato lievemente modificato nel cosiddetto F-controlled NOT gate (Figura).
In questo circuito si vuole calcolare il valore di una funzione utilizzando (nel rettangolo F) operazioni quantistiche. Poiché le operazioni quantistiche devono essere reversibili l'output comparirà nella linea inferiore.
Si immagini allora che il circuito utilizzato per risolvere il problema di Deutsch sia il seguente (Figura):
Il qubit superiore (qubit$_1$) resta così intatto mentre la risposta è nel qubit inferiore (qubit$_2$). La correlazione avviene attraverso il meccanismo dell'entanglement: eseguendo una misura viene distrutto lo stato di sovrapposizione e restano solo una risposta e un domanda strettamente correlate.
Nella linea in basso (2), prima dell'operatore di Hadamard, la funzione *F* avrà creato una sovrapposizione:

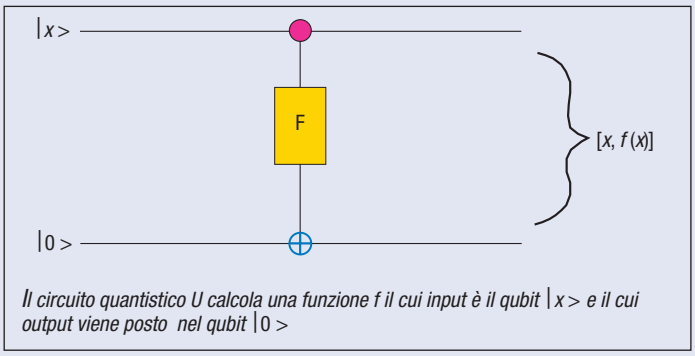
$|0, f(0)> + |1, f(1)>$

Si considerino ora i quattro casi possibili:

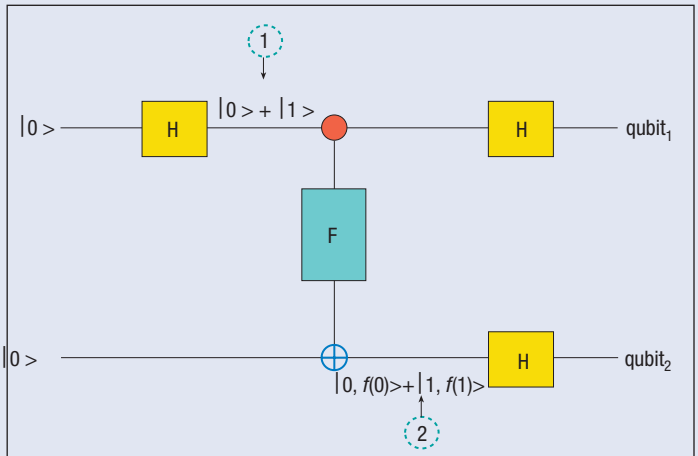
a. $f(0) = f(1) = 0$
b. $f(0) = f(1) = 1$
c. $f(0) = 0, f(1) = 1$
d. $f(0) = 1, f(1) = 0$

Nel punto (2) si avranno, quindi, le quattro seguenti possibilità:

a.	$(\|0> + \|1>)(\|0>)$	$(q_1)(q_2)$
b.	$(\|0> + \|1>)(\|1>)$	$(q_1)(q_2)$
c.	$\|0>\|0> + \|1>\|1>$	$q_1q_2 + q_1q_2$
d.	$\|0>\|1> + \|1>\|0>$	$q_1q_2 + q_1q_2$



Il circuito quantistico U calcola una funzione f il cui input è il qubit $|x>$ e il cui output viene posto nel qubit $|0>$



Le sezioni circuitali denominate H sono trasformazioni di Hadamard mentre la sezione F, quella principale, è preposta al calcolo della funzione f(x), che prende il ingresso il valore del qubit$_1$ e crea in uscita il qubit$_2$

Nei primi due casi il primo qubit è $(|0> + |1>)$ indipendentemente dal contenuto del secondo. In tal caso *F* è indifferente al valore in ingresso e quindi non c'è entanglement tra qubit$_1$ e qubit$_2$. Ma quando i valori sono differenti i bit diventano correlati (entangled) e il valore del primo qubit dipende dal valore del secondo. Nel terzo caso sono identici, nel quarto opposti.
La domandava che Deutsch poneva era: è possibile in un solo colpo sapere se $f(0)$ e $f(1)$ coincidono?
Per rispondere a questa domanda si facciano passare i qubit 1 e 2 attraverso due ulteriori operatori di Hadamard.
Nel qubit$_2$ si otterrà:

		q_1q_2 q_1q_2 q_1q_2 q_1q_2
$(\|0> + \|1>)(\|0> + \|1>)$	=	$\|0>\|0> + \|0>\|1> + \|1>\|0> + \|1>\|1>$
$(\|0> + \|1>)(\|0> - \|1>)$	=	$\|0>\|0> - \|0>\|1> + \|1>\|0> - \|1>\|1>$
$\|0>(\|0> + \|1>) + \|1>(\|0> - \|1>)$	=	$\|0>\|0> + \|0>\|1> + \|1>\|0> - \|1>\|1>$
$\|1>(\|0> + \|1>) + \|0>(\|0> - \|1>)$	=	$\|0>\|0> - \|0>\|1> + \|1>\|0> + \|1>\|1>$

Se ora si misura il qubit$_2$ si otterrà 0 o 1.
Se si otterrà 0 si potranno eliminare la seconda e la quarta colonna e si avrà:

$(|0> + |1>)|0>$
$(|0> + |1>)|0>$
$(|0> + |1>)|0>$
$(|0> + |1>)|0>$

(*segue a p. 60*)

Questo risultato non dice nulla essendo gli stati indistinguibili.
Ma se misurando il secondo qubit si ottiene 1 allora si avrà:

$$
\begin{array}{cc}
q_1 & q_2 \\
(|0>+|1>)|1> & \\
-(|0>+|1>)|1> & \\
(|0>-|1>)|1> & \\
(|1>-|0>)|1> &
\end{array}
$$

Facendo passare il qubit1 attraverso un operatore di Hadamard si otterrà:

$$
\begin{array}{l}
|0>|1> \\
-|0>|1> \\
|1>|1> \\
-|1>|1>
\end{array}
$$

Se ora si misura il qubit1 si avrà ancora 0 (primi due casi) o 1 (secondi due casi).
Ci sono, quindi, due valori ben distinti che correlano ai casi *a*, *b* (valori identici) e *c*, *d* (valori opposti).
Si può, quindi, concludere che se il qubit$_2$ vale 1 allora ha senso guardare il qubit$_1$ e se questo è uguale a 0 allora $f(1)$ è uguale a $f(0)$.
La risposta finale viene data quindi dalla lettura del qubit1 che però era stato lasciato intatto dalla funzione di calcolo *F*.
Ma ciò avviene soltanto a causa della retroazione che l'entanglement crea tra qubit$_1$ e qubit$_2$ dopo le operazioni quantistiche in *F*.
Ciò ha veramente qualcosa di magico ed è proprio qui che Deutsch ha indicato per la prima volta le straordinarie possibilità del calcolo quantistico.

Bibliografia

[1] Benioff P.: The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model. *J. Stat. Phys.*, Vol. 22, 1980, p. 563-591.

[2] Bouwmeester D., Ekert A., Zeilinger An.: *The Physics of Quantum Computation*. Springer,2000.

[3] Deutsch D.: *La trama della realtà*. Biblioteca Einaudi, 1997.

[4] Feynman R.: *Lectures on Computation*. Addison Wesley, 1996.

[5] Feynman R.: *Simulating Physics with Computers reprint*. In: Feynman Lectures on Computation. Addison Wesley, 1996.

[6] Lindley D.: *La luna di Einstein*. Longanesi 1998.

[7] Nielsen M., Chuang I.: *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[8] Penrose R.: *La mente nuova dell'imperatore*. Rizzoli, 1992.

[9] Turton R.: The Quantum Dot. Freeman, *Spectrum*, 1995.

Ernesto Hofmann è laureato in fisica presso l'Università di Roma. È entrato in IBM nel 1968 nel Servizio di Calcolo Scientifico. Nel 1973 è diventato manager del Servizio di Supporto Tecnico del Centro di Calcolo dell'IBM di Roma. Dal 1984 è Senior Consultant IBM. È autore di molteplici pubblicazioni sia di carattere tecnico sia divulgative, nonché di svariati articoli e interviste. Dal 2001, collabora con l'Università Bocconi nell'ambito di un progetto comune Bocconi-IBM.

ICT E DIRITTO

Rubrica a cura di

Antonio Piva e David D'Agostini

Scopo di questa rubrica è di illustrare al lettore, in brevi articoli, le tematiche giuridiche più significative del settore ICT: dalla tutela del *domain name* al *copyright* nella rete, dalle licenze software alla *privacy* nell'era digitale. Ogni numero tratterà un argomento, inquadrandolo nel contesto normativo e focalizzandone gli aspetti di informatica giuridica.

La tutela del Domain Name: da indirizzo Internet a marchio d'impresa

1. Internet e domain name

Il 24 ottobre 1997 Luca Armani registrava il dominio "armani.it", con il quale contraddistingueva il sito del proprio timbrificio. La ben più celebre casa di moda milanese, titolare del marchio "Armani", agiva in giudizio lamentando la violazione del proprio diritto esclusivo, nonché la concorrenza sleale posta in essere dall'artigiano e chiedendo, pertanto, l'adozione degli opportuni provvedimenti inibitori e risarcitori. Il Giudice ha sostanzialmente accolto la tesi della Giorgio Armani SpA, dichiarando l'illiceità della registrazione e dell'utilizzazione del dominio contestato. Questa recente pronuncia del Tribunale di Bergamo (sentenza del 3 marzo 2003) ha dato nuovo impulso al vivace dibattito sui nomi a dominio, fornendo l'occasione per fare il punto della situazione. Sarà utile, prima di entrare nel merito del problema giuridico, partire da alcuni concetti preliminari di ordine tecnico.

2. IP e nome di dominio

Dalla circostanza che in Internet (definibile come un insieme policentrico di reti autonome interconnesse) una moltitudine di macchine dialoghino tra loro, sorge il problema di inviare i dati allo specifico utente che ne aveva fatto richiesta e quindi, in buona sostanza, l'esigenza di identificare univocamente ogni terminale [2]. Per tale ragione a ciascuno di questi, nel sistema implementato dal protocollo TCP/IP (*Transmission Control Protocol/Internet Protocol*), viene attribuito un numero di identità della lunghezza di 32 bit, il che rende possibile la connessione di circa 4 miliardi di utenze (infatti, 232 = 4.294.967.296).
Tuttavia, sarebbe irrealistico pensare di digitare correttamente una stringa come la seguente: http://11000001001010110110000000001000.
La necessità di rappresentare in maniera più semplice la numerazione ha portato alla divisione in 4 gruppi da 8 bit, ognuno dei quali può esprimere, come noto, un valore compreso tra o e 255; ciò consente di identificare più facilmente un indirizzo: http://193.43.96.8.
Attualmente, è in fase di sviluppo il sistema IPv6 che prevede un indirizzo a 128 bit suddiviso in 8 blocchi da 16 bit in notazione esadecimale, tale da assumere una sintassi del tipo 2001:760:2:0:0:5bff:febc:5943.
Questa nuova generazione di protocollo supera alcuni problemi dell'attuale IPv4, tra cui il numero "limitato" di indirizzi, e gradualmente lo sostituirà, senza tuttavia apportare modifiche al meccanismo di DNS. Il DNS (*Domain Name System*) è un sistema specifico[1] che permette di trasformare i nomi di dominio, maggiormente intelligibili per gli uomini ma incomprensibili alle macchine, in indirizzi IP numerici, rendendo ulteriormente agevole all'utente la memorizzazione e la ricerca della risorsa desiderata. Grazie alla sua introduzione, l'indirizzo Internet, sopra indicato, assume l'aspetto maggiormente *user friendly* e familiare: aicanet.it

[1] Concepito nel 1982 e definito negli RFC 1033, 1034 e 1035.

La parte destra del *Domain Name*, nell'esempio ".it", è il primo livello di identificazione e viene detto *Top Level Domain* (TDL), mentre la parte sinistra, nell'esempio "aicanet", viene detta *Second Level Domain* (SLD).
Si ricorda che, in base alle regole tecniche, il SLD deve essere composto da un minimo di 3 a un massimo di 26 caratteri e gli unici ammessi sono:

- le lettere minuscole dell'alfabeto inglese: a – z;
- i numeri: 0 – 9;
- il segno - "meno" (che non può essere primo o ultimo carattere).

I TDL sono di due tipi: i gTLD, detti *generic* o tematici, quali per esempio .edu, .net, .com per *commercial*, .mil, .name; i ccTLD, detti *country code* o geografici, quali per esempio .it per l'Italia, .fr per Francia o .de per Germania.
Quanto al funzionamento del sistema di interrogazioni DNS, i *server* DNS, tramite le proprie tabelle di conversione tra indirizzi IP e nomi simbolici (i domini), sono in grado di rendere disponibile all'utente richiedente, tramite il proprio PC, l'indirizzo di rete del server richiesto. Infatti, quando un utente digita sul proprio computer collegato in rete un indirizzo del tipo www.aicanet.it, qualora il server primario, non fosse in grado di risolvere autonomamente il nome (utilizzando le proprie informazioni presenti nelle tabelle di conversione, ovvero quelle memorizzate nella *cache* in seguito alle interrogazioni svolte in precedenza), viene richiesto l'ausilio di altri server DNS in un processo "a cascata", come visualizzato nello schema della figura 1, inviando al richiedente l'indirizzo numerico indispensabile per stabilire la connessione tra il *client* e il server web, il tutto tramite un meccanismo totalmente trasparente all'utente.
Per consentire a questo sistema di funzionare, è necessario curare l'attività di assegnazione dei domini: per questo motivo, nell'ultimo decennio, hanno visto la luce una serie di enti preposti a tale compito. Al vertice di tale struttura si colloca l'ICANN (*Internet Corporation for Assigned Names and Numbers*)[2], mentre la gestione del country code TLD "it" è delegata dalla medesima alla RA (*Registration Autority*)[3] italiana costituita all'interno del CNR (*Centro Nazionale delle Ricerche*).
La RA svolge il proprio compito operando secondo le regole dettate dalla NA (*Naming Autority*)[4], tra le quali assume particolare rilievo il principio del *first come*, *first served* [1]: in buona sostanza, un nome di dominio viene assegnato a chi ne faccia richiesta per primo e ovviamente, in ottemperanza al cosiddetto "principio di univocità", non possono tecnicamente coesistere due domini esattamente uguali (potrà, invece, essere registrato il medesimo nome sotto TLD differenti, per esempio pippo.it e pippo.com).
La totale assenza di un controllo di merito da parte della RA ha dato luogo all'utilizzazione di marchi altrui come propri nomi di dominio, fenomeno che, in presenza di mala fede, viene definito con i termini di *cybersquatting*[5] o *domain grabbing* [3].

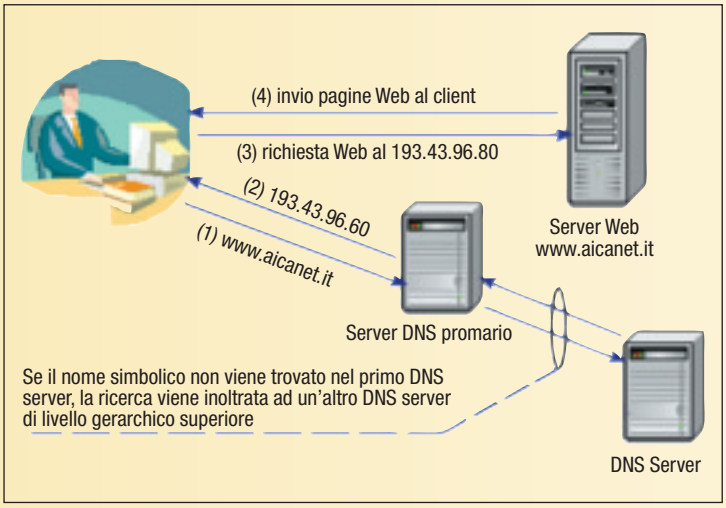


FIGURA 1
Sequenza di azioni svolte per il collegamento al sito Web www.aicanet.it

3. Illeciti e tutela giuridica: il rapporto fra nomi a dominio e segni distintivi

In realtà, gli aspetti problematici inerenti il nome a dominio sotto l'aspetto giuridico non sono limitati a tali comportamenti, ma riguardano anche pretese avanzate da più soggetti legittimati a utilizzare il medesimo nome, fatti-

[2] www.icann.org
[3] www.nic.it/ra
[4] www.nic.it/na
[5] Si nota che all'art. 10 delle regole di naming tale pratica viene espressamente vietata.

specie nella quale non c'è un usurpatore in malafede, bensì due parti in posizione paritetica [6]. Infatti, lo stesso marchio può essere legittimamente registrato in uno stato per indicare diversi beni e servizi oppure in Paesi diversi per lo stesso o per diversi beni e servizi, mentre in Internet, come già detto, non può essere assegnato lo stesso nome di dominio.

La Giurisprudenza italiana, dalla metà degli anni '90 a oggi, ha affrontato il problema emettendo un discreto numero di provvedimenti (per lo più ordinanze cautelari a seguito di ricorsi d'urgenza), che suggeriscono agli addetti ai lavori le prime linee interpretative.

In primo luogo, si è trattato di fornire un'adeguata qualificazione giuridica del nome di dominio, data la sua connotazione "mista". Infatti, pur avendo una finalità tecnica primaria di indirizzo, possiede anche un'indiscutibile funzione distintiva soprattutto sotto il profilo commerciale, assolvendo funzioni simili a quelle del marchio o degli altri segni distintivi nel mondo reale.

La prima tesi, in realtà assolutamente minoritaria, è stata riconosciuta in data 29 giugno 2000 dal Tribunale di Firenze, secondo il quale "*la funzione del nome a dominio è essenzialmente quella di consentire a chiunque di raggiungere una pagina web, pertanto, in quanto mezzo operativo e tecnico, non è assimilabile al marchio d'impresa*" [4].

Oggi, invece, pare consolidata l'assimilazione del nome di dominio ai segni distintivi dell'impresa, pertanto "*qualora venga usato un domain name uguale a un marchio registrato, il conflitto va regolato in base alle norme concernenti il marchio*" [5].

Tra le varie pronunce assumono particolare rilievo, per essere le prime sentenze in materia, quelle del Tribunale di Napoli del 26 febbraio 2002 (caso Playboy) e del Tribunale di Bergamo (relativa al già citato caso Armani). Il nome a dominio è stato ufficialmente compreso tra i segni distintivi atipici, vale a dire nell'ambito dei diritti di proprietà industriale, potendo, quindi, essere tutelato ai sensi del r.d. 21 giugno 1942, n.929 (la cosiddetta **legge Marchi**), ferma restando l'applicazione dell'**art. 2598** c.c. in materia di concorrenza sleale, sempre che ne ricorrano i presupposti.

Legge Marchi

Art 1.1. I diritti del titolare del marchio d'impresa registrato consistono nella facoltà di far uso esclusivo del marchio. Il titolare ha il diritto di vietare ai terzi, salvo proprio consenso, di usare:

a) un segno identico al marchio per prodotti o servizi identici a quelli per cui esso è stato registrato;

b) un segno identico o simile al marchio registrato, per prodotti o servizi identici o affini, se a causa dell'identità o somiglianza fra i segni e dell'identità o affinità fra i prodotti o servizi, possa determinarsi un rischio di confusione per il pubblico, che può consistere anche in un rischio di associazione fra i due segni;

c) un segno identico o simile al marchio registrato per prodotti o servizi non affini, se il marchio registrato goda nello Stato di rinomanza e se l'uso del segno senza giusto motivo consente di trarre indebitamente vantaggio dal carattere distintivo o dalla rinomanza del marchio o reca pregiudizio agli stessi.

C.C. **art. 2598**. (Atti di concorrenza sleale). Ferme le disposizioni che concernono la tutela dei segni distintivi e dei diritti di brevetto, compie atti di concorrenza sleale chiunque:

1) usa nomi o segni distintivi idonei a produrre confusione con i nomi o con i segni distintivi legittimamente usati da altri, o imita servilmente i prodotti di un concorrente, o compie con qualsiasi altro mezzo atti idonei a creare confusione con i prodotti e con l'attività di un concorrente; [...].

Queste norme consentono al titolare dei marchi che godano di rinomanza[6] di vietare a terzi l'uso di un nome identico, qualora tale utilizzo consenta di trarre indebitamente vantaggio dal carattere distintivo del marchio oppure se gli reca pregiudizio. Questa tutela ha carattere ultramerceologico, infatti Luca Armani di Treviglio ha dovuto soccombere pur vendendo timbri e non vestiti, in quanto dallo sfruttamento della capacità attrattiva del noto marchio Armani otteneva un guadagno in termini pubblicitari.

Il giudice, nella motivazione della sentenza, si sofferma incidentalmente sul regolamento[7] di *naming* per la registrazione dei nomi a dominio che disciplina la procedura arbitrale e quella di riassegnazione.

La prima, disciplinata dall'art.15 del regolamento, permette di demandare a un collegio

[6] Secondo la Corte di Giustizia delle Cominità Europee, si definisce rinomato un marchio d'impresa "*conosciuto da una parte significativa del pubblico interessato ai prodotti o servizi da esso contraddistinti*". L'intera sentenza relativa al caso C-375-97 è consultabile nel sito http://curia.eu.int.

[7] Attualmente è in vigore la versione 3.9 approvata dal Comitato esecutivo della NA il 31 luglio 2002.

di 3 arbitri le eventuali controversie connesse all'assegnazione di un domain name; condizione necessaria è che entrambe le parti si siano impegnate previamente per iscritto a riconoscere come valide e vincolanti le decisioni assunte dal collegio.
La seconda, che viene illustrata nell'art.16 e non richiede tale accettazione, consente di riassegnare il nome in caso di registrazione in mala fede.
Per quanto abbiano un'indiscussa utilità e un lodevole vantaggio in termini di tempi e di costi rispetto alla giustizia ordinaria, queste norme (e, in generale, tutto il regolamento) sono - secondo il Tribunale di Bergamo - "*mere regole contrattuali di funzionamento del sistema di comunicazione della rete Internet, di carattere amministrativo interno*"; di conseguenza, non essendo leggi statali, difettano di imperatività nei confronti dei terzi, valendo esclusivamente tra le parti (il richiedente e la RA).
A conclusioni simili era pervenuto in precedenza il magistrato partenopeo il quale affermava la sussistenza della contraffazione di marchio, attribuendo a quest'ultimo una funzione sempre più rivolta alla comunicazione e alla promozione e, quindi, riconoscendo all'imitatore un vantaggio indebito e parassitario, in virtù anche solo del "richiamo psicologico" al marchio celebre.
La medesima sentenza ha pure dichiarato la responsabilità, a titolo di colpa, del *provider*[8] che ha attivato il sito, presentando anche la domanda di registrazione del nome di dominio "playboy.it".
Tale principio di responsabilità, che trova espressa previsione normativa nella direttiva sul commercio elettronico 2000/31/CE, ora recepita nell'ordinamento Italiano mediante il d.lgs. 9 aprile 2003, n.70, si fonda sulla considerazione che il provider, operando in maniera diligente, avrebbe dovuto opporsi a una registrazione che interferiva palesemente con il celeberrimo Marchio.
Chiarita la qualificazione giuridica e individuate le norme applicabili, rimane da dare conto degli strumenti di tutela posti a concreto rimedio, sia dalla legge marchi che dalle disposizioni in tema di concorrenza sleale. Con la sentenza che accerta la violazione, il giudice in primo luogo inibisce la continuazione del comportamento illecito, potendo fissare una somma di denaro a titolo di penale per ogni successiva inosservanza o per il ritardo nell'adempimento; dispone inoltre tutti i provvedimenti opportuni affinché vengano eliminati gli effetti dell'illecito stesso; infine, condanna al risarcimento del danno, purché ne sia fornita la prova del suo ammontare; quale pena accessoria può ordinare la pubblicazione della sentenza.

8 In questo contesto per *provider* si intende il soggetto che, grazie alla propria organizzazione, fornisce al titolare del sito l'accesso alla reteInternet.

Bibliografia

[1] Nespor S., De Cesaris A.L.: Internet e la legge. *Hoepli*, 2001, p.155.

[2] Pasquzzi G.: Scoperte scientifiche, invenzioni e protocolli relativi a Internet. In *AIDA* n. 5/1996.

[3] Tosi E.: *I problemi giuridici di Internet*. Giuffrè, 2001.

[4] Trib. Firenze, 29 giugno 2000 in Dir. industriale 2000, p. 331.

[5] Trib. Reggio Emilia, 30 maggio 2000 in Dir. informatica 2000, p.668.

[6] Varì P.: *La natura giuridica dei nomi di dominio*. CEDAM, 2001.

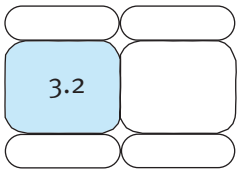
Antonio Piva, laureato in Scienze dell'Informazione, Presidente, per il Friuli - Venezia Giulia, dell'ALSI (Associazione Nazionale Laureati in Scienze dell'Informazione ed Informatica) e direttore responsabile della Rivista di Informatica Giuridica.
Docente a contratto di Informatica giuridica all'Università di Udine.
Consulente sistemi informatici, valutatore di sistemi di qualità ISO9000 e ispettore AICA ECDL core e advanced.
E-mail: antonio_piva@libero.it

David D'Agostini, avvocato, ha conseguito il master in informatica giuridica e diritto delle nuove tecnologie, fornisce consulenza e assistenza giudiziale e stragiudiziale in materia di software, privacy e sicurezza, contratti informatici, e-commerce, nomi a dominio, computer crime, firma digitale. Ha rapporti di partnership con società del settore ITC nel Triveneto.
Collabora all'attività di ricerca scientifica dell'Università di Udine e di associazioni culturali.
E-mail: david.dagostini@adriacom.it

L'EVOLUZIONE DEI LINGUAGGI DI PROGRAMMAZIONE: ANALISI E PROSPETTIVE

Giancarlo Succi

È meglio usare FORTRAN o Basic? C o Pascal? C++ o ADA? Java o C#? Generazioni di informatici hanno combattuto battaglie all'"ultimo sangue" per difendere il "loro" linguaggio di programmazione, adducendo le più svariate motivazioni per giustificare la scelta. Nei 50 anni di storia dei linguaggi di programmazione, la scelta tra "cambiare linguaggio" e "cambiare il linguaggio" sembra sia ricaduta sulla seconda opzione. In questo articolo, si analizza lo sviluppo dei linguaggi con riferimento ai loro processi di sviluppo.



1. LINGUAGGI ... MA SOLO DI PROGRAMMAZIONE?

Nel gergo informatico comune, quando si parla di un "linguaggio", si fa di solito riferimento a un linguaggio di programmazione. Tutti sanno che esistono linguaggi di specifiche, linguaggi di analisi ecc., ma alla domanda: "Che linguaggio usate?" tipicamente la risposta è "Java" oppure "C" o ancora "C++" e così via. Spesso, questa è stata definita come un'anomalia, un limite per il sistema di sviluppo. Può darsi. Di fatto, l'utente finale vuole un sistema *software* che funzioni. Analisi, progetto e tutte le fasi precedenti la codifica sono certamente importantissime a questo scopo. Tuttavia, solo la programmazione fornisce un sistema funzionante, ovvero ciò che ha un valore diretto e tangibile.
Posto in altri termini, mentre esistono prodotti software costruiti solo sul codice, magari costruiti molto male, ma tant'è funzionanti e rispondenti ai requisiti del cliente, non esistono prodotti software costruiti solo con i documenti di analisi, di progetto ecc., e tramite i linguaggi di riferimento per tali fasi.
Pertanto, e senza nulla voler togliere ai linguaggi non di programmazione, in questa trattazione ci si concentrerà sui linguaggi di programmazione (Figura 1).

2. CARRELLATA STORICA

Come si legge in tutti i libri di introduzione alla programmazione, nei primi calcolatori, il programma era definito da circuiti elettrici: veri e propri collegamenti fisici. Si passò poi ai codici binari e ai linguaggi assemblativi a essi associati, ciascuno dei quali si riferiva in modo quasi completamente univoco a specifici elaboratori. Cambiando elaboratore, di solito, cambiava anche il linguaggio.
Il primo sostanziale passo in avanti nella programmazione fu a cavallo tra gli anni '50 e gli anni '60 quando si passò ai cosiddetti "linguaggi di alto livello", ovvero a linguaggi che esprimevano la computazione da fare in modo procedurale, per esempio, il FORTRAN (*FORmula TRANslation*) [1] e il COBOL (*Common Business-Oriented Language*) [25], o in modo funzionale, per esem-

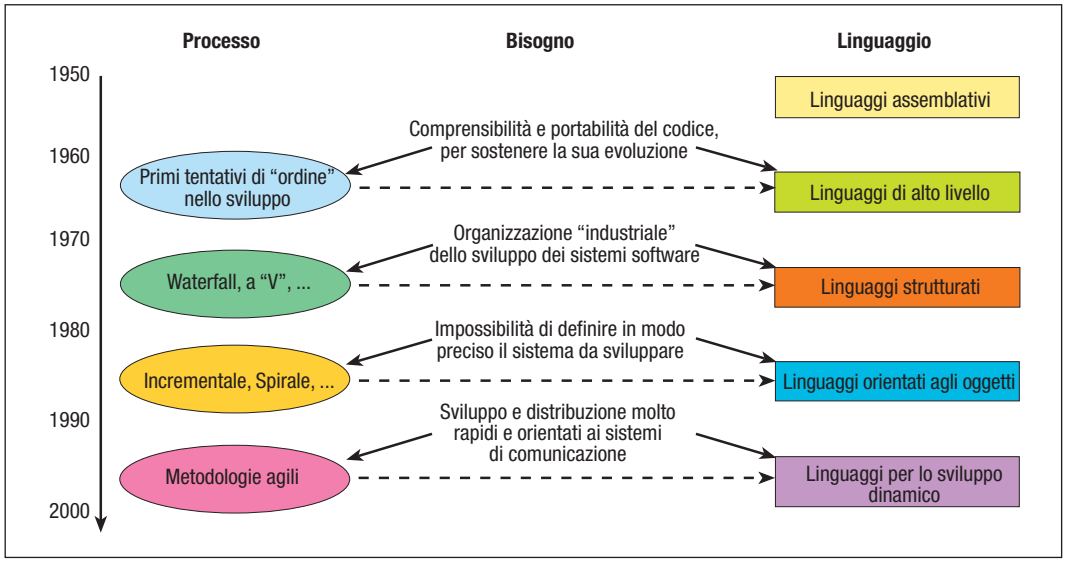


FIGURA 1
Evoluzione dei linguaggi

pio, il LISP (*LISt Processing language*) [24].
A quel tempo, i problemi significativi non riguardavano tanto l'organizzazione del processo di produzione, di solito lasciato nelle mani di pochi esperti, quanto la comprensione e la memoria di ciò che veniva sviluppato all'interno della stessa comunità degli sviluppatori. Di qui, la necessità di trascrivere la computazione in un formalismo familiare ai programmatori dell'epoca; era il tempo delle astrazioni: formule matematiche (FORTRAN), astrazioni logiche (LISP) o transazioni economiche (COBOL)[1]. Inoltre, la diffusione e la diversificazione del mercato dei calcolatori elettronici rese pressante il bisogno di scrivere programmi che funzionassero su più piattaforme di calcolo.
Presto ci si accorse che questo non bastava (fine anni '60 e primi anni '70). I programmi diventavano più grandi e richiedevano la collaborazione di più persone, anche più gruppi di persone. Occorreva strutturare il processo di sviluppo per renderlo più organico. Prendendo a prestito concetti e terminologie dalle teorie di *management* in voga all'epoca, e nei decenni precedenti per essere precisi, i*n primis* una versione molto statica del Fordismo [3], ci si concentrò sull'idea di organizzare il processo di sviluppo in moduli indipendenti con interfacce chiare e componibili. Si diffusero concetti quali la programmazione strutturata, il *data hiding* e l'incapsulamento. Nacquero l'ALGOL [10], il C [21], il Pascal [20], il Modula 2 [35] e l'Ada [15]: anche il FORTRAN fu fatto evolvere in questo senso.
Il processo di produzione modulare nella sua traduzione informatica, il modello di sviluppo "a cascata", dominò gli anni '70 e i primi anni '80. I limiti di questo modello, soprattutto la sua incapacità di gestire la flessibilità richiesta dalla produzione del software, cominciarono a essere evidenti verso la metà degli anni '80. Ci furono piccole variazioni di rotta, come i modelli di sviluppo "a V", ma sostanzialmente il modello modulare e gli associati linguaggi strutturati rimasero predominanti. L'idea guida era che la mancanza di precisione, e l'incertezza che la causava, poteva e doveva essere risolta a priori, tramite specifiche più accurate e circostanziate.
Negli anni '80 la comunità scientifica acquisì la consapevolezza che i problemi non erano solo legati alla carenza umana nelle attività di definizione del sistema, ma anche all'esistenza di una zona di ombra intrinseca allo sviluppo di un qualunque sistema software, che non permetteva di definire in modo completo e corretto fin dall'inizio le caratteristiche che il sistema software avrebbe avuto alla fine.
Il punto di partenza per lo sviluppo di un sistema software sono, in effetti, bisogni veri o percepiti come tali.
Ma tanti di questi bisogni si esprimono non in

1 Si tralascia di menzionare altri linguaggi che, pur molto popolari all'epoca, direbbero molto poco al lettore di oggi.

forma completamente razionale e comunicabile univocamente, ma semplicemente come un "senso di bisogno". Così lo sviluppo parte da requisiti generici e solo gradualmente evolve verso modelli maggiormente definiti.
La comprensione di questa problematica permise un salto nelle modalità di sviluppo del software. La suddivisione modulare lasciò il posto a sviluppi del software in maniera incrementale, per così dire, "a piccoli pezzi", in modo che si potesse usare ognuno di questi pezzi per calibrare lo sviluppo successivo. Facendo un'analogia con la costruzione di una casa, è come se ci si fosse concentrati a costruire una casa facendo prima la costruzione generale e poi una camera alla volta dalle pareti fino all'arredamento e, decidendo sulla base di tale camera, come proseguire.
Lo sviluppo a piccoli pezzi si concretò in molteplici modelli di produzione, quale quello *prototipale*, quello *incrementale*, quello a *spirale* ecc.. In termini di linguaggi di programmazione viene dato l'avvio, nel suo complesso, all'epoca dei linguaggi orientati agli oggetti.
Nei linguaggi orientati agli oggetti il sistema da costruire è rappresentato come un insieme di entità che interagiscono tra loro e che, interagendo, producono il risultato finale; pertanto, lo sviluppo si concentra nella identificazione di tali entità e nella successiva scrittura del codice relativo ad esse e alle loro mutue relazioni. I più famosi tra i linguaggi di programmazione orientati agli oggetti sono senza dubbio Smalltalk [16] e C++ [31], anche se il primo tra tali linguaggi è molto probabilmente SIMULA [9].
Non esistono solo linguaggi di programmazione orientati agli oggetti. Ci sono anche linguaggi di analisi orientati agli oggetti, linguaggi di progetto orientati agli oggetti e così via. A questo proposito, si vuole menzionare l'*Unified Modeling Language* (UML) [29], ora particolarmente in voga.
Si parla, pertanto, spesso di sviluppo orientato agli oggetti come una metodologia a se stante, in cui ci si concentra su analisi orientata agli oggetti, progetto orientato agli oggetti, codifica orientata agli oggetti ecc.
In realtà, è importante distinguere tra strumenti e fini: probabilmente è più adeguato parlare di metodi che usano i vari tipi di linguaggi orientati agli oggetti e che, qualche volta, per esempio nel caso del famoso *Rational Unified Process* (RUP) [22], usano *solo* linguaggi orientati agli oggetti.
Lo sviluppo delle telecomunicazioni, ivi compreso il mondo del web, portò alla necessità di definire processi di sviluppo che tenessero conto di questa realtà molto dinamica e di linguaggi che supportassero gli associati processi di sviluppo. È questo il periodo dell'avvento delle metodologie agili di programmazione, che danno supporto esplicito sia alla mutevolezza di requisiti che alla dinamicità del sistema da sviluppare.
Gli utenti considerano molto importante poter eseguire gli stessi, identici segmenti di codice su piattaforme diverse, nonché trasferirli e combinarli dinamicamente, secondo schemi di calcolo sia sequenziali che paralleli, per produrre rapidamente nuovi *sistemi di elaborazione*, se così ancora si possono chiamare.
I modelli orientati agli oggetti fornirono un importante punto di partenza in quanto permisero l'identificazione di entità base di computazione. Ma occorse dare un maggior impeto sia alla dinamicità dello sviluppo e della fornitura al cliente, che all'interoperabilità tra sistemi e con risorse disponibili in rete, quali archivi, liste di utenti, e banche dati in genere.
Divennero allora popolari i cosiddetti *linguaggi per lo sviluppo dinamico*, in quanto sono in grado di sostenere lo sviluppo di sistemi componibili dinamicamente e interoperabili tra loro.
Una parte di questi linguaggi era formata da semplici evoluzioni di linguaggi di *scripting*, ovvero linguaggi interpretati, finalizzati a guidare il sistema in ciò che esso doveva fare.
Si diffusero, però, anche linguaggi di programmazione completi che ottenevano gli stessi scopi, incorporando anche quei costrutti che via via, nell'evoluzione dei linguaggi, si erano affermati come essenziali. I più famosi esempi di questi linguaggi sono Java e C#.
Questa è la situazione in cui ci si trova oggi. È importante sottolineare che in questa carrellata si sono messi in relazione processi di sviluppo software con specifiche istanze di linguaggi di programmazione. Aver associa-

to un linguaggio a un processo non deve però in alcun modo far pensare che quel linguaggio sia nato come una risposta al bisogno definito dal processo. Anzi, molto spesso il linguaggio in sé anticipa il processo cui fa riferimento. L'ALGOL, uno dei primi linguaggi strutturati, fu concepito alla fine degli anni '50 [10]; SIMULA, l'archetipo dei linguaggi orientati agli oggetti, venne sviluppato negli anni '60 [9]; i primi i linguaggi per lo sviluppo dinamico sono degli anni '70: il *Concurrent Pascal* [19] e tutta la varietà di linguaggi prototipali prodotti dal gruppo di Hewitt [18].

3. LA PROGRAMMAZIONE DI ALTO LIVELLO

Vediamo ora il nesso tra processo di sviluppo e linguaggio ad esso associato, al fine di discuterne il legame.
Come precedentemente osservato, la programmazione di alto livello fu motivata dal bisogno di avere programmi scritti in formalismi comprensibili a un largo numero di sviluppatori e non dipendenti da architetture specifiche.
Tre particolari categorie di sviluppatori ebbero, in questo senso, un peso preponderante:

- gli scienziati e i tecnici - fisici, astronomi, chimici, ingegneri ecc., - che usavano i linguaggi per sviluppare applicazioni legate alle proprie attività di ricerca e sviluppo;
- i matematici, che stavano definendo le teorie alla base dell'informatica e dell'intelligenza artificiale;
- i responsabili dei sistemi informativi aziendali, che volevano automatizzare vari tipi di transazioni economiche e finanziarie sia per ragioni di efficienza che per evitare errori umani.

Di conseguenza, tre linguaggi acquisirono un'importanza tale che ancora oggi, nelle loro successive reincarnazioni, continuano ad essere usati: FORTRAN, LISP e COBOL.

❑ *FORTRAN*

Gli scienziati avevano bisogno di un linguaggio che permettesse la scrittura e la successiva elaborazione di formule complesse. Nacque così il FORTRAN [1, 2], che iniziò la propria lunga storia nel 1954. Così lunga da rendere reale la profezia degli anni '70: "Non si sa come sarà il linguaggio di programmazione dell'anno 2000, ma si chiamerà FORTRAN![2]".
Il FORTRAN funzionò bene per le applicazioni di calcolo scientifico e, di conseguenza, ebbe un notevolissimo successo in questo dominio, come evidenziato dalla profezia precedentemente citata.

❑ *LISP*

Il LISP fu ideato da John McCarthy nel 1958 con lo scopo di riportare in forma di linguaggio di programmazione il modello computazionale del λ-calcolo.
Il λ-calcolo fu definito dai padri dell'informatica, tra cui Church [8], e si diffuse presto, diventando uno dei modelli computazionali prevalenti. Avere un linguaggio di programmazione basato integralmente su di esso rivestiva, pertanto, un'importanza unica per gli scienziati di allora, che pensavano, così, di risolvere gran parte dei loro problemi di sviluppo software tramite la trascrizione, quasi pedestre, di modelli in programmi.
Per esperti di λ-calcolo l'uso del LISP fu, ed è tuttora, particolarmente indolore. Questa è una delle fortune maggiori di tale linguaggio, ancora oggi, dopo quasi 50 anni dalla sua prima concezione, particolarmente popolare nel settore dell'intelligenza artificiale.

❑ *COBOL*

Il linguaggio COBOL, originato all'inizio degli anni '60 [25], ambiva a trasporre in linguaggio di programmazione le transazioni economiche e finanziarie. Il punto di riferimento di questo linguaggio erano gli operatori del mondo degli affari che avevano esperienza di bilanci, scritture contabili, tabulati, regole precise e spesso un poco ridondanti.
Già la procedura seguita per la specifica del linguaggio evidenziava questa origine. Esso fu definito da un comitato formato dai *leader* statunitensi dell'epoca. Tra gli altri, si ricorda, per quanto concerne il settore privato, la partecipazione di Burroughs Corporation, IBM, Honeywell, RCA, Sperry Rand e Sylvania Electric Products. Anche tre enti pubblici particolarmente rilevanti presero parte ai lavori: US Air Force, David Taylor Model Basin e il National Bureau of Standards.

2 Questa profezia è stata attribuita a vari studiosi tra cui J. Backus, S. Cray e D. McCracken.

Il comitato si suddivise in tre sottocomitati: il primo per la pianificazione a breve termine, il secondo per quella a medio termine, e il terzo per il lungo termine; quest'ultimo in realtà non incominciò mai a lavorare.
È chiaro che, con queste premesse il linguaggio che sarebbe risultato, sarebbe stato preciso, ma anche un po' ampolloso e pesante da gestire. Questo fu, in effetti, ed è tuttora, il COBOL.
Tirando le somme, non c'è dubbio che anche il COBOL assolva molto bene i propri compiti di chiarire le azioni che l'elaboratore svolge in un linguaggio comprensibile dai propri utenti principali, ovvero da persone che si occupano di amministrazione, contabilità e finanza.

4. LA PROGRAMMAZIONE STRUTTURATA E DI SISTEMA

Come prima menzionato, la diffusione degli strumenti informatici nonché l'ingrandirsi dei sistemi da sviluppare, impose la definizione di un processo di sviluppo.
L'idea che parve più ovvia fu quella di basarsi su modelli modulari, concentrati sulla suddivisione del lavoro in parti e la successiva specializzazione dei compiti. Il modello di sviluppo di riferimento fu chiamato *waterfall*, ovvero, *a cascata* proprio da questo.
Si assistette, allora, a una particolare focalizzazione sullo sviluppo di linguaggi che permisero la decomposizione flessibile del sistema in sottosistemi.
Come si è detto, anche il COBOL permetteva una suddivisione; tale suddivisione, però, era rigidamente predefinita dal linguaggio: le divisioni erano "quelle tre".
Si desiderava, invece, poter dividere in modo che i moduli fossero definibili in modo chiaro e univoco dallo sviluppatore, che doveva anche stabilire come un modulo interagisse con gli altri moduli.
Come già menzionato, il punto di partenza per questo approccio fu l'ALGOL [10], seguito poi dal C [21] e dal Pascal [20]. È, però, con il Modula 2 [35] e con l'Ada [15] che la programmazione strutturata acquisisce la sua completa espressione. Nel seguito, di questa sezione si analizzano il Modula 2 e l'Ada. Si presenta, poi, una breve riflessione sulla nuove versioni del FORTRAN, il FORTRAN 77 [12] e il FORTRAN 90 [13].

4.1. Modula 2

L'intento del Modula 2 è duplice:
- da un lato, vuole favorire la suddivisione del lavoro in moduli indipendenti e comunicanti solamente tramite chiare interfacce;
- d'altro lato, vuole semplificare la scrittura del modulo nel suo complesso, separando la parte dichiarativa, in cui si specifica quello che un modulo fa, dalla parte implementativa, in cui si implementano funzioni finalizzate a ottenere quanto precedentemente dichiarato.

In questo senso, il Modula 2 sembrò essere il passo fondamentale per risolvere i problemi della crisi del software tramite l'applicazione dei modelli fordisti.
L'organizzazione in moduli garantiva la parcellizzazione del lavoro, assicurando la possibilità di scrivere sistemi sempre più grossi. La separazione dell'interfaccia dall'implementazione assicurava sia la specializzazione delle competenze (l'implementazione era fatta da esperti dei diversi domini applicativi, ma l'uso era aperto a chiunque potesse leggere le specifiche descritte nell'interfaccia), sia che eventuali modifiche implementative operate a un modulo, dovute, per esempio, alla già allora veloce evoluzione degli strumenti informatici, non provocassero effetti catastrofici per altri moduli che lo usavano.
Il successo di Modula 2 non è tanto testimoniato dalla diffusione del linguaggio, che rimase piuttosto limitata, quanto all'influenza che ebbe su linguaggi sia esistenti che di nuova concezione. Modula 2 provò, infatti, che le idee dei modelli "a cascata" potevano avere riflessi essenziali sui linguaggi di programmazione. Sia C che FORTRAN si aggiornarono per acquisire il maggior numero possibile di caratteristiche di Modula 2 come una migliore separazione tra interfaccia e implementazione, la tipizzazione stretta ecc. Ada ricalcò tutta la struttura a moduli nei propri *package*.

4.2. Ada

Ada nacque sotto l'egida del Dipartimento della Difesa statunitense con l'ambizione di voler diventare il linguaggio universale per i

sistemi *embedded, in primis*, e poi per tutta l'informatica.
Con tale sponsorizzazione, il successo sembrava garantito. Il linguaggio rassomigliava molto a Modula 2, a parte la nomenclatura: il modulo si chiamava package.
Una peculiarità di Ada che vale la pena rimarcare è la presenza di una ricca scelta di modalità di "passaggio parametri": per valore, per risultato, per valore-risultato e poi, con una forzatura, per riferimento via puntatori. Si può intravedere in questo il desiderio di formalizzare, non solo al livello alto del modulo ma anche al livello più basso delle singole funzioni, i protocolli di interazioni tra le diverse parti di programma.
In ogni caso, essendoci poco da aggiungere a Modula 2, in chiave di chiarezza operativa gli ideatori di Ada si concentrarono sull'estensione del linguaggio per gestire ambiti di programmazione utili in sistemi *embedded* ma non considerati esplicitamente da Modula 2, come la programmazione concorrente e la gestione delle eccezioni, sull'ampliamento dei tipi astratti tramite l'uso dei generici e, infine, sulla definizione di una semantica precisa.
Queste estensioni, però, resero difficile la definizione formale dello stesso linguaggio e causarono dei ritardi evidenti nello sviluppo di compilatori che ne permettessero il largo utilizzo e la diffusione. Fu, pertanto, solo alla fine degli anni '80 che Ada cominciò a prendere piede su larga scala, quando ormai i modelli "a cascata" si avviavano al tramonto.

5. MODELLI A V

All'inizio degli anni '80 il modello di sviluppo "a cascata" cominciò a evidenziare i propri limiti, soprattutto per quanto concerne l'incapacità di gestire la variabilità dei requisiti.
Come precedentemente menzionato, la soluzione fu quella di "cercare di essere più precisi" nello sviluppo del software, attribuendo alla scarsa precisione degli analisti, dei progettisti e degli implementatori, il mancato successo del linguaggio.
Si passò allora al modello di sviluppo "a V", in cui si associava a ogni fase di sviluppo (ovvero analisi, progetto e codifica), una batteria di test per verificare la sua correttezza. I test andavano eseguiti quando la batteria di test sottostante era stata completata. In pratica, il flusso delle attività era: analisi, progetto, codifica e poi test della codifica, test del progetto, test dell'analisi. Dato il movimento discendente (dal generale al particolare) nella fase di sviluppo e ascendente (dal particolare al generale) nella fase di test, si chiamò questo modello "a V".
Nello sviluppo del codice si ebbe un riflesso diretto del modello a V tramite le cosiddette "asserzioni".
In pratica, si definivano per ogni funzione pre-condizioni e post-condizioni che dovevano essere soddisfatte prima e dopo l'esecuzione di ogni funzione. Inoltre, si potevano inserire delle espressioni booleane in punti critici di una funzione, le asserzioni, che dovevano essere sempre valutate vere. Il problema passava allora alle definizioni di queste condizioni e asserzioni e alla loro valutazione durante l'esecuzione del programma [4]. Di fatto, nessuna versione di un linguaggio di programmazione effettivamente usato inglobò l'idea delle asserzioni, a parte le librerie di asserzioni del compilatore *gnu* del C. Essa rimase, quindi, nell'aria e ora si è praticamente tradotta nel concetto di *test first*, tanto caro agli sviluppatori utilizzanti le metodologie agili.

5.1. FORTRAN 77 e FORTRAN 90

L'evoluzione del FORTRAN verso la programmazione strutturata e di sistema avvenne in due passi: il FORTRAN 77 e il FORTRAN 90.
Il FORTRAN 77 venne definito con l'intento di eliminare gli aspetti chiaramente obsoleti delle versioni di FORTRAN ancora largamente in uso nei primi anni '80: il FORTRAN 4 e il FORTRAN 66.
In particolare, venne definito un costrutto per i cicli e si obbligò il programmatore a definire esplicitamente tutte le variabili, in modo da limitare le possibili sviste.
Le novità portate dal FORTRAN 90 furono molteplici: eliminò i vincoli di formattazione precedentemente descritti, permise la definizione di tipi strutturati e di moduli, aggiunse un insieme di costrutti di controllo equivalenti a quelli di Modula 2 e di Ada, tra cui

le tecniche di *passaggio parametro* tipiche di Ada, e incluse anche alcune tecniche primordiali per la definizioni di oggetti. Inoltre, aggiunse strumenti per la programmazione parallela, dato che gli sviluppatori in FORTRAN erano spesso scienziati interessati a calcoli su strutture di grandi dimensioni.

6. LA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

Verso la metà degli anni '80, l'informatica comprese che i problemi dei modelli di sviluppo *a cascata* nascevano dall'intrinseca incertezza che pervade il modo stesso di sviluppare software.
Questo cambiava radicalmente la possibile soluzione. Occorreva un modello, non tanto focalizzato a raggiungere una chimerica esattezza, quanto capace di gestire una congenita variabilità.
I modelli di divisione del lavoro diventavano, di colpo, inadeguati: se lo scopo di un sistema da sviluppare non era chiaro, una qualunque sua suddivisione rischiava di rendere ancora meno chiaro lo scopo del sistema.
Si pensò allora a uno sviluppo *a piccoli pezzi*. L'idea fu di prendere come punto di partenza l'ambito applicativo dei sistemi da sviluppare e di studiare a fondo tale ambito con lo scopo di capire quali elementi andavano sicuramente considerati nel sistema da sviluppare. Per esempio, se si trattava di sviluppare un sistema per la gestione di un aeroporto, occorreva considerare l'aspetto dei voli, dei passeggeri, dei biglietti e così via. Si poteva allora partire nello sviluppo dal modello di comportamento di tali entità.
Tali entità furono chiamate oggetti e, continuando la metafora, trasmissioni di *messaggi* tra diversi oggetti si sostituirono a *chiamate a funzioni*. Ogni oggetto aveva una serie di *metodi* per la risposta ai messaggi che rimpiazzarono il *corpo delle funzioni*[3].
L'uso degli oggetti facilitò anche le comunicazioni con il cliente, in quanto gli oggetti erano presi dal linguaggio del dominio applicativo noto al cliente stesso e, quindi, il flusso della computazione diventava spesso di più semplice comprensione.
Lo sviluppo a piccoli pezzi si concretizzò in diversi modelli di produzione, tra cui si ricordano i modelli incrementali, quelli prototipali e quelli a spirali. Tutti facevano esplicito riferimento alla programmazione orientata agli oggetti.

6.1. C++

Infatti, tra gli scopi del C++ ci fu quello di rendere facile la transizione alla programmazione orientata agli oggetti per la comunità degli sviluppatori di sistema in ambiente Unix, solita a programmare in C.
Il C++ è un linguaggio in questo senso controverso: i puristi lo considerarono, fin dall'inizio, non un vero linguaggio orientato agli oggetti, ma un ibrido e gli preferirono sistemi più integri, quali Smalltalk.
In questo articolo, non si vuole entrare nel dibattito su quale sia il migliore linguaggio orientato agli oggetti. Di fatto, però, C++ è il più diffuso e, quindi, rappresenta un punto di riferimento importante se si vuole studiare l'evoluzione dei linguaggi di programmazione.
In primis, si ha la presenza del codice dell'implementazione insieme a quello della dichiarazione, seppure con una separazione tra *zona pubblica* – accessibile a tutti, e *zona privata* – accessibile solo agli implementatori. Questo sembra contraddire l'idea della divisione e della specializzazione del lavoro. Ma se la dimensione del codice rimane contenuta, secondo l'idea dello sviluppo incrementale, essa serve a fornire una specifica operazionale al metodo. Chiaramente, se il codice è lungo e complesso, la sua presenza è dannosa.
L'idea di documentare il codice con il codice stesso ebbe molta fortuna e ora è largamente usata in Java, anche tramite una specifica strutturazione dei commenti, con un particolare strumento chiamato JavaDoc, che analizza il codice ed estrae parti di commenti formattati secondo una semplice e chiara sintassi per fornire in modo automatico una documentazione sintetica ma efficace del sistema.
Le definizioni di zona pubblica e zona privata

[3] Peraltro, i metodi sono anche chiamati *funzioni membro*.

diventano, pertanto, quasi un suggerimento all'utente della classe su dove concentrare, in modo prioritario, la propria attenzione, piuttosto che una separazione contrattuale di compiti[4].

6.2. FORTRAN 95 e FORTRAN 2000 e altri linguaggi

Il FORTRAN proseguì la propria evoluzione verso l'universo della programmazione a oggetti con due versioni: il FORTRAN 95 e il FORTRAN 2000. Esse vengono qui nel seguito brevemente tratteggiate.

Il FORTRAN 95 arricchì il linguaggio con una serie di costrutti di controllo più raffinati e con ulteriori specializzazioni del concetto di funzione: le funzioni pure e le funzioni di elementi. Le funzioni pure non hanno effetti collaterali, non agiscono cioè su variabili globali o di ambiente. Le funzioni di elementi hanno come argomenti e restituiscono solo valori scalari. Si definirono, inoltre, tecniche per una gestione più pulita dei puntatori, inclusa la loro inizializzazione e l'allocazione e deallocazione dinamica della memoria. Infine, si rese più robusta la gestione dei tipi derivati.

Il FORTRAN 2000 è attualmente in corso di definizione e sta compiendo un passo sostanziale nel rendere FORTRAN completamente orientato agli oggetti [28]. In particolare, si sta rafforzando la gestione dei generici estendendoli ai tipi derivati, l'uso dei puntatori a funzione e la gestione di oggetti polimorfi. Inoltre, sta migliorando l'interfacciamento con altri linguaggi orientati agli oggetti e non, per rendere più agevole l'interoperabilità tra sistemi, secondo i cardini dello sviluppo a piccoli pezzi.

L'approccio generale è più ridondante del C++. In particolare, c'è una marcata separazione tra interfaccia e implementazione che ricorda i linguaggi strutturati; questa interfaccia addirittura disaccoppia il nome del metodo dal corpo del metodo, richiedendo la presenza di un esplicito operatore per la connessione: l'operatore ⇒. La filosofia generale, però, è chiaramente quella ibrida a oggetti del C++.

7. LA PROGRAMMAZIONE DI SISTEMI COMPONIBILI DINAMICAMENTE

La programmazione orientata agli oggetti causò un miglioramento sostanziale dello sviluppo del codice. Di contro, però, richiese una competenza decisamente superiore da parte degli sviluppatori.

Si potrebbe quasi asserire che mentre i modelli *a cascata* volevano risolvere i problemi con la formalizzazione delle attività da svolgere e la specializzazione del lavoro, i modelli *a piccoli pezzi* si concentravano sulla definizione di un linguaggio ricco che permettesse di catturare nel modo più completo possibile i requisiti dei clienti, comunque organizzati in insiemi di dimensioni ridotte.

7.1. Limiti dell'approccio orientato agli oggetti

Due fattori tra loro connessi emersero, però, con effetti dirompenti. Il primo fu che ci si rese conto che l'uso del C++ o di un altro linguaggio di simili connotazioni richiedeva un'adeguata formazione e che la formazione del personale in un linguaggio complesso era ovviamente non semplice, soprattutto se tale personale proveniva da anni di uso di linguaggi di programmazione meno evoluti.

Il secondo fattore fu la domanda del mercato di prodotti in tempi brevi e il contestuale comparire sulla scena del web, che rendeva ancora più impellente tale domanda e aggiungeva la necessità di avere un linguaggio che si adattasse a combinare pezzi di codice che si muovessero sulla rete.

La combinazione di questi due fattori contribuì alla fortuna del linguaggio Java.

Le metodologie agili, poi, definirono un processo di sviluppo che sosteneva gli stessi obiettivi: limitare la complessità di sviluppo, favorire la consegna del prodotto all'utente in tempi brevi e certi anche se con funzionalità ridotte, garantire la qualità e eliminare gli sprechi dovuti a sviluppi di moduli inutili o ad attività sì connesse con lo sviluppo ma non direttamente produttive.

[4] Si noti che in Smalltalk, il linguaggio di programmazione agli oggetti considerato forse più *puro*, questa suddivisione non esiste.

7.2. Java

Nacque con l'intento di definire un linguaggio ad oggetti quanto più *completo e puro* possibile, utilizzando come base la sintassi del C e senza compiere gli errori dell'Objective C [5], un precedente linguaggio nato con i medesimi scopi ma naufragato per l'ambiguità di fondo di non avere né la pulizia di un linguaggio ad oggetti né la flessibilità del C.
Per semplificare l'apprendimento e la gestione dei programmi scritti nel linguaggio, e ridurre, quindi, lo sforzo per l'apprendimento di costrutti inutili, si eliminarono alcune caratteristiche particolarmente onerose del C++: la gestione della memoria dinamica lasciata all'utente, i generici, l'ereditarietà multipla delle classi e il passaggio parametri per riferimento.
Per facilitare l'uso del sistema in modo dinamico sulla rete si cambiò il meccanismo di generazione dell'oggetto e dell'eseguibile, si resero tutte le funzioni virtuali (con *binding* ritardato) e si definirono costrutti specifici per la gestione della concorrenza.
Dall'analisi del processo di compilazione ed esecuzione (Figura 2) si può capire l'aderenza delle specifiche del linguaggio agli scopi di Java precedentemente delineati e ai modelli di sviluppo agili.

Nel caso del C++, come nel caso dei linguaggi di alto livello[5] e dei linguaggi strutturati finora considerati, fu scelto il seguente processo per la generazione di un programma eseguibile:

- dapprima un modulo, chiamato "compilatore", provvede ad analizzare il *file sorgente* e a generare un file "oggetto", o "OBJ", che è indipendente dal linguaggio sorgente originario, ma dipendente dall'hardware e dal sistema operativo;
- quindi, un secondo modulo, chiamato *linker*, dipendente dallo specifico hardware e sistema operativo, ma spesso indipendente dal linguaggio sorgente, si occupa di cercare e unire i diversi moduli oggetto, sia quelli sviluppati dal singolo programmatore che quelli provenienti dal sistema operativo, per costruire il modulo eseguibile, quello che può essere caricato e fatto funzionare sul calcolatore da solo e in un unico blocco.

In alcuni testi e in alcuni ambienti di sviluppo le due fasi sono trattate in modo così consequenziale da essere chiamate genericamente entrambe *compilazione*.
Va da sé che in questo contesto non è assolutamente scontato che un modulo oggetto o un modulo eseguibile sviluppati per una data configurazione di hardware e sistema operativo funzionino su un'altra configurazione.

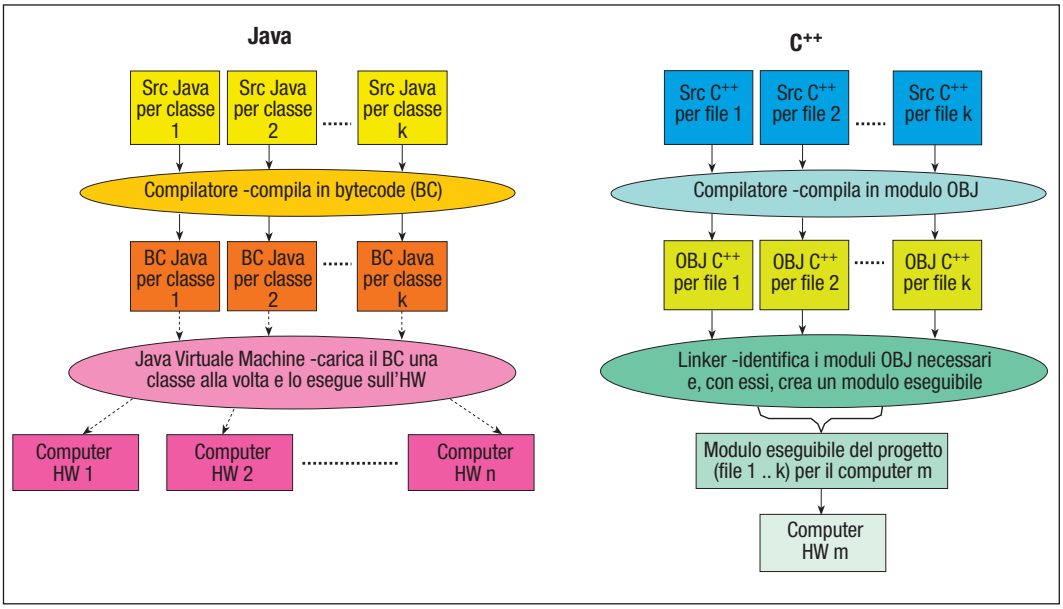


FIGURA 2
Analisi del processo di compilazione ed esecuzione in Java e in C++

[5] In realtà, il LISP ha un processo di interpretazione più simile a quello di Java che agli altri. Di fatto, però, il *programmatore medio* del LISP di solito non percepisce questa differenza, e quindi non sfrutta le potenzialità ad essa collegate.

È di conseguenza esclusa la possibilità che lo stesso codice eseguibile possa migrare di macchina in macchina continuando a funzionare senza nulla toccare.
Inoltre, occorre che tutto il sistema sia costruito in un solo colpo e che, al momento del link, si abbia a disposizione la versione finale di tutti i moduli oggetto necessari; tale versione non è più cambiabile[6].
In Java, fu preferito un approccio completamente differente.
Il codice è compilato, una classe alla volta, da un compilatore Java che lo trasforma in un modulo *simil-oggetto* chiamato di solito "bytecode". Per ogni classe pubblica (ovvero, ogni classe utilizzabile al di fuori del package di definizione) esiste uno e un solo file bytecode. Il bytecode è lo stesso per ogni compilatore Java, indipendentemente dalla configurazione di sistema operativo e hardware in uso.
Un altro modulo chiamato *virtual machine* o *macchina virtuale*[7] provvede, quindi, a caricare ed eseguire una ad una le classi effettuando la ricerca del bytecode delle classi necessarie dinamicamente nel corso dell'esecuzione del programma stesso.
La macchina virtuale disaccoppia il bytecode dal sistema operativo e dall'hardware sottostante, fornendo tutti quegli strumenti necessari per l'esecuzione del programma e occupandosi anche della gestione automatica della memoria dinamica e della concorrenza.
Il funzionamento delle macchine virtuali è, in pratica, lo stesso[8] per ogni sistema operativo e per ogni hardware, anche se, ovviamente, il codice eseguibile della macchina virtuale non è necessariamente lo stesso.
Ne risulta che il programma può tranquillamente migrare da un computer all'altro, purché ovviamente il calcolatore sia dotato della macchina virtuale adeguata allo scopo.
Il caricamento in memoria fatto per classi permette, inoltre, di limitare la quantità di codice da trasmettere sulla rete, in quanto si possono utilizzare le parti di classi di sistema presenti sul calcolatore destinazione. Inoltre, il caricamento parziale di codice permette di rendere maggiormente dinamici lo sviluppo e la successiva evoluzione del sistema.
Se si definiscono in modo chiaro le dipendenze mutue tra classi, cosa difficile ma non impossibile, soprattutto se si mantengono gli assunti della programmazione orientata agli oggetti, di rendere gli oggetti costantemente riferiti alle entità del dominio applicativo, è allora possibile che gli oggetti di versioni diverse della stessa classe rispondano agli stessi messaggi con risposte sicuramente non identiche, ma equipollenti dal punto di vista dello sviluppo del sistema, in modo che il sistema possa evolvere dinamicamente.
Un uso tipico di questo approccio dinamico sia per quanto concerne la migrazione del codice che la definizione di versioni multiple della stessa entità, si ha nella possibilità di caricare nelle pagine web frammenti di codice Java chiamati *applet* che vengono trasmessi dal *server* web al calcolatore che sta visualizzando la pagina web e vengono poi eseguiti sulla macchina virtuale residente sul calcolatore destinazione.
Java riconobbe l'importanza di *usare* la sintassi di un linguaggio affermato, il C++, in modo da invogliare i programmatori in tale linguaggio a considerarlo come una potenziale alternativa.

7.2. Autodocumentazione e test-first in Java

Per completare la panoramica su Java e approfondire il legame che lo lega con i modelli di sviluppo agile, è importante menzionare che Java fu concepito anche per fornire un supporto all'autodocumentazione del codice. Insieme al linguaggio fu costruito un *tool* per documentare la struttura in classi

[6] Si semplifica la realtà ignorando, volutamente, le librerie di sistema caricabili dinamicamente o quelle condivise: esse non sono state pensate per cambiare il comportamento a *run time* di un sistema quanto per poter ridurre le dimensioni dell'eseguibile e rendere più veloce il suo caricamento ed esecuzione, ovvero (come nel caso delle DLL sui vecchi sistemi operativi DOS) per renderlo eseguibile *in toto*.

[7] Spesso si usa anche in italiano il termine inglese.

[8] Si trascurano i problemi legati alle diverse versioni di macchina virtuale, poiché si tratta di un problema di gestione delle configurazioni e non di struttura del linguaggio di programmazione.

anche tramite l'identificazione e l'estrazione di commenti particolari che il programmatore può scrivere all'uopo.
I cultori delle metodologie agili estesero subito a Java l'approccio *test-first*, originariamente pensato per Smalltalk.
Test-first è un processo di scrittura del codice in cui dapprima si definisce per ogni metodo di una classe, non appena si sa di doverlo sviluppare e si conoscono i suoi parametri, uno o più test che ne verifichino il corretto comportamento e, quindi, si usano i test come guida per lo sviluppo della classe. La similitudine tra l'approccio test-first e le asserzioni precedentemente citate è evidente.
Test-first è particolarmente efficace in Java nello sviluppo di sistemi componibili dinamicamente, in quanto le moltitudini di test che vengono così via via create diventano un controllo molto efficiente contro possibili disallineamenti che si vengono a creare quando si costruisce il codice a pezzetti e a intervalli successivi, magari anche in luoghi distribuiti geograficamente.
L'approccio test-first fu consolidato in Java con jUnit, il *porting* di sUnit, lo strumento precedentemente usato in Smalltalk per poter eseguire automaticamente tutti i test in un colpo solo. In un contesto di sviluppo agile, jUnit rappresentò un passo essenziale per assicurarsi che i test fossero effettivamente sviluppati prima del codice e costantemente eseguiti.
In ultima analisi, si può dire che l'autodocumentazione e il test-first rappresentino un'ulteriore forte evidenza che lega processo di sviluppo agile ai linguaggi per i sistemi componibili dinamicamente.
A questo punto si potrebbe obiettare che Smalltalk fu proprio il precursore dei linguaggi per sistemi componibili dinamicamente e non tanto dei linguaggi a oggetti. Questa obiezione è, infatti, molto interessante: delinea, in modo significativo, l'importanza di tale linguaggio ed evidenzia l'ispirazione di chi lo pensò. Del resto, Smalltalk è legato anche al filone del LISP e dei linguaggi funzionali [30] e, come già menzionato, ci sono legami non ovvi ma profondi tra i modelli funzionali proposti dal LISP e i linguaggi per sistemi componibili dinamicamente.

8. LINGUAGGI, LINGUAGGI, LINGUAGGI, ...

In questa veloce carrellata sono stati evidenziati solo una minima parte dei linguaggi di programmazione usati.
In particolare, non è stata spesa neanche una parola sulla serie dei linguaggi Microsoft a partire dai primi BASIC adattati per il DOS fino a C# ecc.. Questo non vuole essere in alcun modo una censura nei confronti della più grande azienda software del mondo, che molti criticano ma che indubbiamente ha finanziato ricerche di altissima qualità che hanno contribuito alla crescita del mondo dell'informatica.
Bisogna riconoscere però che, quando ha operato direttamente e non tramite i finanziamenti alla ricerca erogati all'Università, Microsoft si è focalizzata nella direzione del perfezionamento di strutture linguistiche già esistenti e in quella degli ambienti di sviluppo per rendere quanto più efficiente possibile alcune tecniche di sviluppo considerate strategiche, piuttosto che nella creazione di nuove strutture linguistiche di interesse per questo lavoro.
Ci sono poi i linguaggi di *script* che hanno costituito la base della programmazione di sistema. Anche in questo caso, l'averli ignorati non significa averli sottovalutati. Il linguaggio della *C-shell* ha formato generazioni di sistemisti Unix, così come PERL successivamente ha permesso la creazione dei prototipi dei siti web dinamici, e quindi PHP[9] e così via.
Bisogna però ripetere quanto detto per Microsoft: sono linguaggi molto efficaci ma che sublimano alcune caratteristiche esistenti in un dato linguaggio per un dominio applicativo ben preciso: per esempio, la programmazione di pagine web, la creazione di portali, la programmazione *in the small* di sistema. Come tali, sono fuori dall'ottica di questo articolo.
Una menzione va poi ai linguaggi funzionali, al di là del LISP, e a quelli logici. Per gli altri,

[9] Il termine PHP è un acronimo ricorsivo che sta per *PHP Hypertext Processor*, ovvero, processore PHP di ipertesti. Questo acronimo ben evidenzia le caratteristiche di questo linguaggio, oggi diventato molto popolare.

anche se hanno creato strutture interessanti, non si può dire che si siano diffusi in modo significativo. Inoltre, e questa è una opinione molto personale dell'autore, hanno più usato e modellato teoricamente caratteristiche già precedentemente pensate piuttosto che gestito un sistema nuovo: i tipi generici sono nati prima di ML[10] (*Meta Language*); le gerarchie di classi sono nate molto prima di Haskell; l'analisi delle strutture "ad alberi" era presente in YACC prima del PROLOG; la programmazione *a clausole* era presente prima dei recenti linguaggi *a vincoli*.
Forse due aspetti sono originali a questi linguaggi: la valutazione *lazy* per la creazione di strutture infinite –propria di alcuni linguaggi funzionali quali LML e Haskell, e la possibilità di avere parametri che sono contestualmente di *input* o di *output*, come nel caso del PROLOG.
Ma, dopo l'entusiasmo iniziale, non pare che questi modelli abbiano originato sostanziali avanzamenti nella disciplina.
È opportuno sottolineare che questo non vuol dire affatto che tali linguaggi si siano rivelati inutili semplicemente, essi non si legarono mai ad alcun processo in quel binomio indivisibile di cui si discute. Con una metafora si può affermare che nessuno penserebbe di andare con un'auto di Formula 1 per i vicoli del centro storico di Genova, o con una bicicletta in autostrada, ma questo non significa che le macchine di Formula 1 o le biciclette siano inutili.

9. SI PUÒ TRARRE UNA CONCLUSIONE?

Si pensa che l'analisi effettuata confermi l'ipotesi di partenza sulla imprescindibile connessione tra linguaggio e processo.
È difficile ma interessante provare a predire il futuro dei linguaggi usando tale paradigma. Chiaramente non si possono che identificare tendenze passate e capire come queste tendenze possano evolversi.
Si può poi notare che i processi di produzione del software e poi i linguaggi hanno attraversato un'interessante parabola in termini di complessità (Figura 3).
Come già precedentemente ricordato, dapprima si cercò di rispondere ai bisogni dell'industria software con modelli di produzione sempre più complessi e in secondo momento, dato che i processi diventavano via via sempre più pesanti, i linguaggi li seguirono accrescendo le proprie funzionalità.
Finalmente, con i modelli incrementali, si arrivò a capire che rendere il processo ancora più complesso non avrebbe giovato: il sistema era diventato troppo complesso e occorreva trovare qualche altro approccio e semplificare il modello di sviluppo per sperare in ulteriori miglioramenti.
I linguaggi continuarono a crescere, in complessità, fino al proverbiale C++. Una divertente falsa intervista, con Stroustrup[11], l'inventore del C++, evidenzia come la comunità degli sviluppatori abbia percepito la complessità di questo linguaggio.
Alla fine, ci si rese conto che la complessità del linguaggio non era più sostenibile: semplicemente costava troppo formare sviluppatori per tali linguaggi o mantenere sistemi scritti in tali linguaggi. Con Java, iniziò la parabola discendente di complessità.
Quanto durerà la discesa? Chi scrive pensa che continuerà ancora. La diffusione dei processi agili è irreversibile in tutti i settori del mondo produttivo ed essa è solo agli inizi nell'industria informatica. Essa porta il bisogno di eliminare lo spreco nella programmazione.
L'eliminazione delle attività inutili è, quindi, un cardine delle future tendenze. Essa potrà realizzarsi come semplificazione dei linguaggi di programmazione, ovvero come l'uso di quei linguaggi che hanno sintassi ridotte al minimo, come Smalltalk.
Essa potrà anche apparire sottoforma di nuovi sistemi integrati per lo sviluppo, che automatizzino o semiautomatizzino le attività e i controlli del programmatore. Si pensi, per esempio, all'efficacia di JUnit in Java per il test e di Javadoc per la documentazione.
Si può, quindi, fare una semplice considera-

[10] ML fu il primo linguaggio funzionale ad avere una diffusione oltre la ristretta cerchia degli inventori.

[11] URL: http://www.nsbasic.com/newton/info/nsbasic/interview.shtml

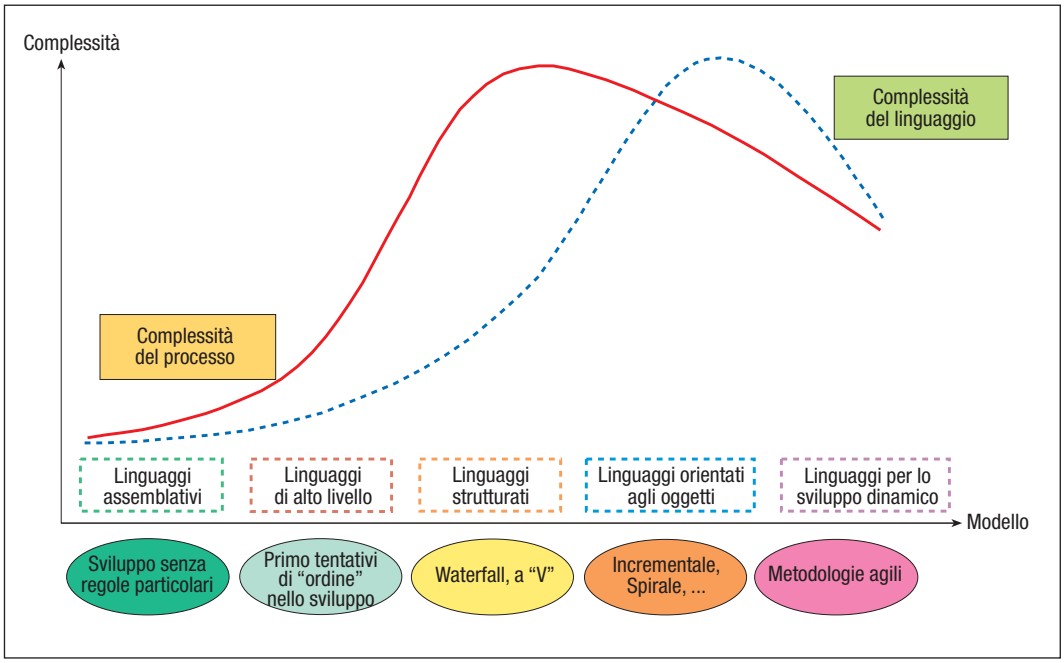


FIGURA 3
Evoluzione della complessità nel processo e nel linguaggio

zione, quasi scherzosa. Sicuramente, chiunque profetizzò le fortune del FORTRAN[12], o meglio, del suo nome e della struttura della sua sintassi, ebbe ragione. Questo è il mondo del FORTRAN e non si sarebbe affatto stupiti di trovarsi un FORTRAN 3000 "tra i piedi" tra qualche anno...
Di fatto, la storia dell'uso dei linguaggi di programmazione condiziona quelli che saranno i linguaggi in uso nel futuro. Gli economisti chiamano questo fenomeno *path dependency* [23].
Un esempio classico è il *layout* della tastiera della macchina da scrivere e del computer, la cui inefficienza è provata rispetto a soluzioni alternative, ma che risulta non conveniente cambiare in quanto il costo di transizione sarebbe maggiore del guadagno ottenuto cambiando.
La fortuna della famiglia dei linguaggi basati sul primo FORTRAN o di quelli basati sul C sta probabilmente in questo. Potrebbero esistere altri sistemi radicalmente diversi e superiori per scrivere codice. Per esempio, l'autore pensa che Smalltalk sia estremamente più semplice e compatto. Di fatto, la transizione a una sintassi diversa rende questo percorso di difficile praticabilità, forse impossibile.
Resta, dunque, sempre più attuale la strategia di *cambiare il linguaggio piuttosto che cambiare linguaggio*. C++ e Java si stanno muovendo in questa direzione.

Nell'articolo si è parlato in svariati punti del concetto di "*tipo di dato*".
I tipi di dati rispondono a due scopi.
Gli ingegneri del software hanno sempre nutrito il desiderio impossibile di poter controllare in modo statico la correttezza semantica di un programma. Di fatto, le uniche verifiche fattibili sono quelle sintattiche. In questo contesto si è cercato di raffinare sempre più ciò che è sintatticamente esprimibile, al fine di massimizzare quanto sia controllabile prima dell'esecuzione di un programma.
Il primo scopo dei "tipi di dato" è proprio quello di aumentare le parti controllabili tramite la definizione di vincoli sulla operazioni fattibili.
Così come, ad esempio, in matematica per ogni tipo di numero (naturale, intero, razionale ecc.) esistono un insieme di operazioni ammissibili, un dominio per tali operazioni e un co-dominio, con i tipi di dati si vuole fornire all'utente la possibilità di definire tipi come quelli matematici e operazioni ammissibili, in modo tale che, laddove si compia una operazione non ammissibile, un controllo sintattico possa tempestivamente informare lo sviluppatore.
I tipi di dati utilizzabili in un programma si sono evoluti da semplice combinazioni di più valori elementari, come nel primo FORTRAN, a complesse algebre di ordine superiore con relazioni funzionali e definizioni anche ricorsive, come in C++.
Anche nel caso dei tipi di dati, passando da C++ a Java si è assistito a una progressiva semplificazione dei tipi ammissibili.
Il secondo scopo dei tipi è stato quello di fornire un supporto alla modularità, tramite l'astrazione dei dati. Di questo si discute anche nella parte finale dell'articolo.

[12] Si fa riferimento alla citazione sulle fortune del nome FORTRAN, il cui autore, come detto in nota 2, non è determinato con certezza.

Bibliografia

[1] Backus J.W.: *Automatic Programming: Properties and Performances of FORTRAN Systems I and II*. Atti del Symposium of Mechanisation of Thought Processes, Teddington, UK, 1958.

[2] Backus J.W.: The History of FORTRAN I, II, and III. In: *IEEE Annals of the History of Computing*, Vol. 20, n. 4, 1998, p. 68-78.

[3] Banham R., Newman P.: *The Ford Century: Ford Motor Company and the Innovations That Shaped the World*. Artisan Sales, 2002.

[4] Broy M., Krieg-Brückner B.: Derivation of Invariant Assertions During Program Development by Transformation. *Transactions on Programming Languages and Systems*, Vol. 2, n. 3, 1980, p. 321-337.

[5] Budd T.: *An Introduction to Object-Oriented Programming*. Addison Wesley, 1991.

[6] Cox B.: *Object Oriented Programming – An Evolutionary Approach*. Addison Wesley, 1986.

[7] Chidamber S.R., Kemerer C.F.: A metrics suite for object-oriented design. *IEEE Transactions on Software Engineering*, Vol. 20, n. 6, June 1994, p. 476-493.

[8] Church A.: *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jersey, 1941.

[9] Dahl O.-J., Nygaard K.: SIMULA –An Algol Based Simulation Language. *Communications of the ACM*, Vol. 9, n. 9, 1966, p. 671-678.

[10] Dijkstra E.W.: *Primer of Algol 60 Programming*. Academic Press, 1962.

[11] Dodrill G.: *Coronado Enterprises Modula-2 Tutor*. 1987, URL: http://www.modula2.org/tutor/

[12] Ellis T., Miles R.: *A Structured Approach to FORTRAN 77 Programming*. Addison-Wesley Publishing, 1983.

[13] Ellis T., Miles R.: *FORTRAN 90 Programming*. Addison-Wesley Publishing, 1994.

[14] Feuer A.R., Gehani N.H.: Comparison of the Programming Languages C and Pascal. *ACM Computing Surveys*, January, 1982.

[15] Gehani N.H.: *Unix Ada Programming*. Prentice Hall, 1985.

[16] Goldberg A., Robson D.: *Smalltalk-80: The Language and Its Implementation*. Addison Wesley, 1983.

[17] Graham P.: *History of FORTRAN*, URL: http://www.paulgraham.com/history.html

[18] Hewitt C., Bishop P., Steiger R.: *A Universal Modular Actor Formalism*. Atti della 3rd International Joint Conference on Artificial Intelligence, Stanford, CA, 1973.

[19] Hansen P.B.: The Programming Language Concurrent Pascal. *IEEE Transactions on Software Engineering*, Vol. 1, n. 2, 1975, p; 199-207.

[20] Jensen K., Wirth N.: *PASCAL User Manual and Report*. Springer, 1975.

[21] Kernighan B.W., Ritchie D: *The C Programming Language*. Prentice Hall, 1978.

[22] Kruchten P.: *The Rational Unified Process: An Introduction*. Addison Wesley, 1998.

[23] Liebowitz S.J., StMargolis S.E.: Path Dependence, Lock-in, and History. *Journal of Law, Economics, and Organization*; 1995, anche disponibile on-line all'URL: http://wwwpub.utdallas.edu/~liebowit/paths.html

[24] McCarthy J.: *A Revised Version of MAPLIST*. MIT AI Lab., AI Memo n. 2, Cambridge, September 1958.

[25] McCracken D.D.: *A Guide to Cobol Programming*. Wiley, 1963.

[26] Meyer B.: *Eiffel: The Language*. Prentice Hall, 1992.

[27] Murtagh J.L., Hamilton J.A.: A comparison of Ada and Pascal in an introductory computer science course. *ACM SIGAda Ada Letters*, Vol. 18, n. 6, 1998.

[28] Reid J.: *The New Features of Fortran 2000*. URL: www.ukhec.ac.uk/publications/reports/N1507.pdf

[29] Rumbaugh J., Jacobson I., Booch G.: *The UML Reference Manual*. Addison Wesley, 1998.

[30] Sethi R.: *Programming Languages: Concepts and Constructs*. Addison Wesley, 1996.

[31] Stroustrup B.: *The C++ Programming Language*. Addison Wesley, 1996.

[32] Tsaptsinos D., Davies R., Rea A.: *Fortran 90 – An introduction to the language for beginners*. URL: http://www.pcc.qub.ac.uk/tec/courses/f90/ohp/header_ohMIF_1.html

[33] US Department of Defence: *Department of Defense Requirements for High Order Computer Programming Languages: Steelman*. 1978.

[34] Wheeler D.A.: *Ada, C, C++, and Java vs. The Steelman*. Ada Letters, July/August, 1997.

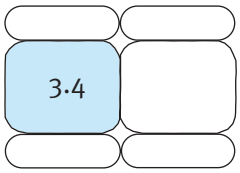
[35] Wirth N.: *Programming in Modula-2*. Springer Verlag, 1982.

GIANCARLO SUCCI è ordinario di Informatica e direttore del Center for Applied Software Engineering presso la Libera Università di Bolzano, dove si occupa di metodologie agili per lo sviluppo software, metriche software, ingegneria del software empirica, strumenti di e-learning nell'industria software, linee di prodotto software. È stato ricercatore presso l'Università di Trento, professore associato alla University of Calgary e quindi professore ordinario alla University of Alberta, dove ha co-diretto l'Alberta Software Engineering Research Consortium. È autore di più di 150 articoli scientifici e di 5 libri.
e-mail: giancarlo.succi@unibz.it

LIMITI DI VELOCITÀ SULLE AUTOSTRADE DELL'INFORMAZIONE

Ezio Biglieri
Guido Vannucchi

La storia delle comunicazioni numeriche moderne è la storia dei tentativi di mettere in pratica quanto enunciato teoricamente da Shannon nel 1948, a cui si deve il calcolo dei limiti massimi di velocità a cui è possibile inviare informazioni in modo affidabile su un canale di trasmissione. Questo obiettivo ha stimolato i progettisti ad avvicinarsi sempre più a questi limiti, fino a raggiungerli nella pratica, solo pochi anni fa, per alcuni tipi di canale: nell'articolo si esaminerà l'itinerario seguito per avvicinarsi ai limiti teorici di Shannon.



1. PREMESSA

Grazie al genio di Claude E. Shannon (1916-2001), il 1948 fu per le telecomunicazioni l'anno della rivoluzione. In effetti, tale anno segnò un radicale mutamento nell'impostazione teorica dei relativi studi, sia per una precisa e chiara modellizzazione dei sistemi sia per i metodi matematici nell'analisi del funzionamento dei sistemi di comunicazione, temi affrontati in un famoso articolo dal titolo "A Mathematical Theory of Communication" pubblicato da Shannon sul *Bell System Technical Journal*.

La motivazione dell'articolo era pratica: come trasferire al meglio a un *utente informazioni* generate da una sorgente lontana nello spazio (è il problema classico della trasmissione a distanza) o nel tempo (immagazzinamento dell'informazione). Il trasferimento avviene grazie alla presenza di un *canale di comunicazione*, affetto da disturbi (chiamati genericamente *rumore*) che tendono ad alterare la qualità della trasmissione. Per impostare questo problema nel modo piú appropriato, Shannon intuì che avrebbe dovuto rifarsi a una definizione precisa di "informazione", concetto fino ad allora alquanto vago: in particolare, egli mise in luce che il *contenuto di informazione di un messaggio* non ha niente a che fare con la rappresentazione cui si è abituati, ma semplicemente con la quantità di simboli binari ("1" e "0") che sono sufficienti per trasmetterlo. A lui si deve anche l'introduzione della parola "*bit*" (contrazione di *binary digit*, o cifra binaria) per definire l'unità elementare di informazione: "1 bit" è per esempio il contenuto di informazione convogliato dal verificarsi, nel lancio di una moneta, dell'evento "testa" o "croce". Oggi, questo concetto è abbastanza familiare a tutti, come documenta la parola "bit" ormai entrata nell'uso comune, ma per l'epoca in cui fu introdotto esso rappresentò una radicale innovazione per un'intera generazione di ingegneri abituati a ragionare con riferimento alla natura specifica di ciascun tipo di informazione. La schematizzazione di un sistema completo di comunicazione secondo Shannon è illustrata nella figura 1.

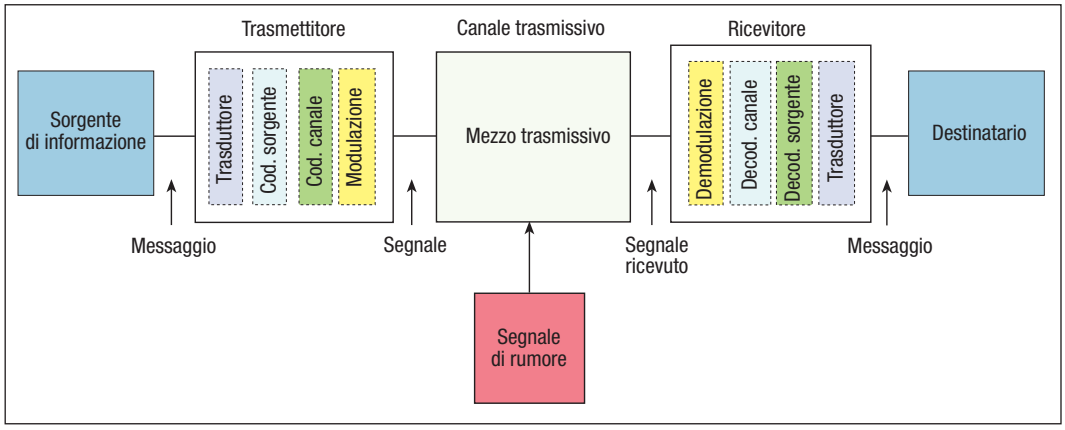


FIGURA 1
Schema di un canale di comunicazione secondo Shannon

Nella figura 1 il *trasmettitore* include tutte le operazioni che servono a trasformare la sorgente primaria di informazione in un appropriato formato elettronico che possa alimentare il canale di trasmissione, ivi compresa l'eventuale eliminazione della ridondanza naturale del segnale elettrico che rappresenta la sorgente.
Il *trasduttore* trasforma la sorgente primaria in segnale elettronico. La riduzione di ridondanza (compressione) si può conseguire attraverso un'opportuna *codifica di sorgente*: per esempio, nel caso televisivo, essa è data dalla codifica MPEG, dopo la "cattura" elettronica dell'immagine reale, per ridurre l'occupazione spettrale nella diffusione televisiva, e, nel caso telefonico, dalla compressione della voce ad 8 kbit/s per la trasmissione su reti radiomobili cellulari. All'altro estremo, il *ricevitore* compie le funzioni inverse del trasmettitore, consentendo la restituzione dell'informazione originaria all'utente.
Per quanto la codifica di sorgente sia utilizzata ampiamente nei moderni sistemi di comunicazione per renderli più efficienti, essa è tuttavia estranea alle considerazioni di ottimizzazione illustrate in questo articolo, dedicato principalmente al miglioramento delle prestazioni ottenibili sul canale trasmissivo propriamente detto con riguardo al suo specifico segnale d'ingresso (anche se quest'ultimo ha già incluso altre ottimizzazioni).
Il *canale di trasmissione*, impiegato per convogliare l'informazione dal trasmettitore al ricevitore, è costituito dal particolare mezzo trasmissivo (coppia d'utente, fibra ottica, etere ecc.). Per adattare al canale di trasmissione l'informazione, all'uscita dal codificatore di sorgente, si introducono il *codificatore di canale* e il *modulatore*. Il primo introduce una *ridondanza artificiale* all'informazione, allo scopo di proteggerla dagli effetti del rumore migliorando così le prestazioni qualitative del canale grazie al controllo, più o meno accurato, degli errori che il rumore può causare; il secondo trasforma l'informazione, generalmente rappresentata da una successione di simboli binari (i bit), in un segnale elettrico nella forma (una tensione o un'onda elettromagnetica ecc.) adatta al mezzo trasmissivo impiegato.
Naturalmente, sul fronte di uscita del canale di trasmissione, ossia al ricevitore, verranno compiute le operazioni inverse di demodulazione, decodifica di canale e decodifica di sorgente per ritrasformare il segnale d'ingresso in un flusso informativo che parli il linguaggio dell'utente.
Elemento essenziale della schematizzazione della figura 1 è la sorgente di *rumore*, che deteriora le prestazioni del canale di trasmissione. Il termine "rumore" è generico e caratterizza *tutti i tipi di disturbi* provocati sul mezzo trasmissivo dal rumore termico dei circuiti elettronici, dall'interferenza intersimbolo, dall'accoppiamento di altri sistemi trasmissivi vicini ecc. Per quanto riguarda il funzionamento del codificatore di canale, Shannon dimostrò che se un numero sufficiente di "bit ridondanti", cioè privi di un loro contenuto informativo, viene aggiunto al messaggio utile d'ingresso, è possibile proteggere integralmente l'informazione trasmessa dall'effetto dei disturbi sopra accennati. Per fare questo, Shannon definì matematicamente un parametro del canale, chiamato "*capacità*" (ter-

mine che nella sua definizione si identificava con il limite massimo di velocità trasmissibile sul canale) e dimostrò che per ogni velocità di trasmissione inferiore alla "capacità" *esiste un codice che permette di ottenere una probabilità di errore arbitrariamente bassa* e quindi può rendere la trasmissione sul canale perfettamente affidabile (*teorema di Shannon*). La "*capacità*" o velocità (*bit-rate*) massima del canale, espressa in *bit al secondo*, è ovviamente una funzione che dipende dalle caratteristiche (attenuazione, banda ecc.) del particolare mezzo trasmissivo impiegato, dalla potenza del segnale trasmesso e dalla natura e intensità del rumore che influenza il canale di trasmissione. Definita dagli addetti ai lavori come "*limite di Shannon*" di un canale di trasmissione, la capacità può essere calcolata per ogni specifico mezzo trasmissivo. Tuttavia, la dimostrazione del teorema non è costruttiva, nel senso che non spiega come costruire il codice che permette di raggiungere questo limite con perfetta affidabilità della trasmissione: per questo, i progettisti dei sistemi di comunicazione presero a scoprire codici a controllo di errore sempre più sofisticati, assicurando gradi via via crescenti di integrità dei dati trasmessi. Il progresso nel campo dei *codici* ha sfruttato con notevole profitto la progressiva diminuzione del costo realizzativo dei circuiti numerici, dovuto all'integrazione. Per molti anni dopo il 1948, pur avendo ben acquisiti i principi e le teorie introdotte nel famoso articolo, i sistemi di trasmissione progettati erano in grado di trasmettere ad una velocità effettiva così lontana dal "limite di Shannon" che tale confine fu considerato per molto tempo un'esercitazione puramente teorica, troppo lontana dalle prestazioni reali che un sistema trasmissivo era effettivamente in grado di realizzare. Scopo del presente articolo è essenzialmente quello di illustrare e richiamare l'attenzione su come - a seguito dell'introduzione di nuovi codici di canale e di nuove modulazioni, o di una combinazione di entrambi - sia stato possibile, in questi ultimi anni avvicinarsi in modo sostanziale a questo limite, confermando nella pratica realizzativa la fondamentale importanza delle profonde intuizioni e degli studi teorici di Shannon. Essenziali, a tale proposito, sono state anche le realizzazioni circuitali rese possibili, in modo compatto ed economico, dai progressi dell'integrazione microelettronica.

2. IL PROGETTO DI UN SISTEMA DI TRASMISSIONE: EFFICIENZA DI BANDA ED EFFICIENZA DI POTENZA

Il problema centrale del *progetto di un sistema di trasmissione numerica* si può riassumere nei termini seguenti: in base agli obiettivi primari che ci si prefigge (per esempio, alta utilizzazione dello spettro di frequenza disponibile, ovvero ridotta potenza di trasmissione, o una opportuna combinazione delle due cose), trovare le tecniche che rappresentano il miglior *compromesso tecnico-economico* per comunicare messaggi in modo affidabile e veloce su un prefissato mezzo trasmissivo.
Il mezzo trasmissivo (impiegato con modalità bidirezionale o monodirezionale) può essere costituito da una grande varietà di portanti: doppino telefonico, radio punto-punto, cavo coassiale, fibra ottica, etere per collegamenti punto-punto o punto-multipunto (terrestri o satellitari) ecc..
Il canale di trasmissione include, come già detto, una quantità di disturbi che ne degradano le prestazioni di qualità: rumore termico, interferenza intersimbolica (causata dalla finitezza della banda di frequenza a disposizione), cammini multipli di propagazione, interferenza proveniente da altri utenti che condividono le medesime risorse trasmissive ecc..
Ai fini del progetto del sistema di comunicazione adottato, si tratta di usare nel modo più accorto possibile le risorse disponibili, ossia banda e potenza di trasmissione del segnale, in modo da raggiungere, al minore costo e senza eccessiva complessità, la desiderata *velocità di trasmissione* (bit-rate espressa in bit al secondo, o bit/s) e l'obiettivo imposto di q*ualità di servizio*. Questa qualità di servizio è espressa tipicamente in termini di *probabilità di errore* (*error rate*), la quale a sua volta è una funzione decrescente del rapporto tra la potenza del segnale e la potenza di rumore (il *rapporto segnale-rumore* S/N) all'ingresso dei circuiti del ricevitore.

Va anche tenuto presente che, per determinati servizi, l'occupazione di una banda prefissata, o l'impossibilità di superare certi limiti di potenza, o una combinazione delle due, rappresenta un ben determinato vincolo di progetto di cui occorre tener conto. A tale proposito, a titolo di esempio, si ricordano alcuni tipici sistemi trasmissivi moderni e i relativi vincoli: per un sistema ADSL che opera su un portante a banda stretta quale il doppino d'utente, il massimo sfruttamento della banda è certamente prioritario rispetto agli altri parametri; per una trasmissione televisiva da satellite che impiega bande con ampio spettro, la limitazione di potenza diventa, al contrario, il vincolo prioritario per la difficoltà dei satelliti a erogare potenze trasmissive troppo elevate, in connessione con la necessità di non superare a terra un determinato diametro per le parabole; per un servizio radiomobile cellulare ambedue i parametri di ottimizzazione (banda stretta, bassa potenza) sono essenziali nel progetto, in quanto la banda di tali sistemi è una risorsa estremamente costosa, mentre la potenza massima utilizzabile nella trasmissione da un terminale mobile è chiaramente limitata dal peso del terminale, dalla necessità di avere batterie a lunga durata di carica e dall'opportunità di non fare generare campi elettromagnetici troppo intensi da un terminale telefonico usato a piccola distanza dal capo.

A pari qualità di servizio, si possono, pertanto, considerare due possibilità estreme:

a. un primo caso è quello in cui obiettivo prioritario è l'*efficienza spettrale* (*o di banda*): questa è definita come il rapporto R/B (espresso in *bit utili* al secondo per Hz di banda) tra la velocità R a cui intendo trasmettere il carico utile di informazione e la banda B occupata dal sistema;

b. il caso all'altro estremo è quello in cui obiettivo essenziale è l'*efficienza di potenza* per ridurre al minimo (a pari qualità) la potenza di trasmissione, anche a scapito di una ridotta occupazione spettrale. Il parametro che caratterizza l'efficienza di potenza di un sistema è definito come indicato nel seguito.

Per elevati rapporti segnale-rumore S/N, la probabilità di errore sul canale può essere approssimata con buona precisione da una funzione decrescente del tipo:

$$Pe = f(\gamma E_b/N_0) \qquad (1)$$

il cui argomento è proporzionale al rapporto segnale/rumore, definito qui come il rapporto tra l'energia E_b necessaria a trasmettere un bit di informazione e la densità spettrale (unilatera) di potenza N_0 dei disturbi ("rumore" di potenza N) che si aggiungono al segnale utile ($N_0 = N/B$). Il fattore di proporzionalità, indicato con γ - che esprime l'efficienza con cui lo schema di trasmissione utilizza l'energia a disposizione per assegnare una certa probabilità di errore - è appunto il parametro che si definisce efficienza di potenza. Tanto maggiore è γ, tanto minore è la potenza di segnale richiesta per consentire la stessa probabilità di errore.

In un *regime limitato in banda*, il compito di incrementare l'efficienza spettrale spetta all'impiego di *modulazioni multilivello*, le quali, per una varietà di mezzi trasmissivi, si prestano bene a risolvere il problema del risparmio di banda. In tale regime, se vengono usati codici a controllo di errore, questi devono essere a bassa ridondanza (e, quindi, non particolarmente efficaci nel miglioramento delle prestazioni) perché il loro utilizzo non deve aumentare in modo considerevole la velocità di trasmissione, e di conseguenza la banda.

Al contrario, nel caso di *sistemi limitati in potenza*, i criteri di progetto suggeriscono l'utilizzo di potenti *codici a controllo di errore*, il cui obiettivo è incrementare l'efficienza nell'uso della potenza. I bit ridondanti che devono essere trasmessi costringono a incrementare la velocità di trasmissione, quindi costringono a un'espansione della banda necessaria (il che non costituisce un grave problema se il sistema non ha pesanti limiti di utilizzo dello spettro). In compenso, essi consentono un risparmio di potenza (o, se si preferisce, la necessità di un minore rapporto S/N in ricezione) in quanto permettono di ottenere un miglioramento della qualità di trasmissione senza un corrispettivo incremento della potenza trasmessa.

In termini qualitativi, si dice che si lavora in un regime "*limitato in banda*" se i vincoli del progetto forzano a operare con un'efficienza

spettrale significativamente maggiore di 1, e in un regime "*limitato in potenza*" se avviene l'opposto.
Nel caso più generale, ossia quando può essere utile mirare ad ambedue gli obiettivi, occorre perseguire il miglior compromesso, compatibilmente con i vincoli di progetto, sempre tenendo presente che, per garantire la qualità di servizio richiesta, un aumento di efficienza spettrale si ottiene a prezzo di un corrispondente decremento dell'efficienza di potenza. Viceversa, un aumento di efficienza di potenza richiede una diminuzione di efficienza spettrale, per la necessità di aumentare la ridondanza richiesta da codici più complessi. In queste situazioni, l'abbinare in cascata *codifica e modulazione* con opportuni parametri può aumentare l'efficienza sia di potenza che di banda per meglio raggiungere il compromesso desiderato. Questa soluzione verrà esaminata nel seguito.

3. IL CONTRIBUTO DI SHANNON E L'EVOLUZIONE DEGLI STUDI SU MODULAZIONI E CODICI

Una delle novità del più volte citato lavoro di Shannon consiste nell'osservazione che il compromesso fondamentale del progetto di un sistema di trasmissione numerica non si fonda soltanto nello *scambio tra i due parametri di potenza e banda*, descritti precedentemente. Questo si era creduto fino al 1948: ma il lavoro di Shannon introdusse un terzo parametro, costituito dal *ritardo* causato dalla presenza di un codice potente, quindi dotato di numerosi bit ridondanti e dunque *lungo*. In effetti, per rendere arbitrariamente piccola la probabilità di errori in trasmissione quando si trasmetta a velocità prossime alla "*capacità*", occorre accettare un codice complesso e, dunque, un ritardo nel trasferimento dell'informazione cha aumenta a mano a mano che ci si avvicina alla "*capacità*".
In situazioni particolari di rumore, quali ad esempio quelle che si verificano sui canali radio-mobili, è anche possibile facilitare i compiti dei normali codici correttori attraverso la tecnica del cosiddetto *interlacciamento temporale* (*time interleaving*) che permette che errori che avverrebbero in bit contigui (per esempio, a causa di rumori a struttura impulsiva) siano viceversa distribuiti nel tempo a opportune distanze. In tal modo, un rumore impulsivo torna ad essere assimilabile a un rumore totalmente casuale e può essere tenuto sotto controllo piú facilmente. L'interlacciatore temporale è ormai comunemente associato ai normali codici: esso opera senza introdurre ulteriore ridondanza, ma la sua funzione viene attuata a spese di un ritardo nella trasmissione del messaggio che si aggiunge a quella creata dalla ridondanza del normale codice di canale. Per un sistema di trasmissione, le scelte di uno schema di modulazione e codifica sono fortemente influenzate dal tipo di canale. Come si è detto, a partire dal 1948, gli ingegneri di telecomunicazioni si sono sforzati di sviluppare sistemi di modulazione e codifica realizzabili nella pratica, ma sempre con l'intento di avvicinare al meglio le prestazioni ideali, ossia il "limite di Shannon". Nonostante una robusta dose di pessimismo, che permeava l'ambiente soprattutto nei primi anni (si riteneva, infatti, che i codici "ottimi" fossero talmente complicati da risultare assolutamente irrealizzabili in pratica), il problema fu progressivamente risolto a partire da una decina di anni fa, almeno per un canale particolare (ma importante), il cosiddetto *canale gaussiano lineare* in cui la distribuzione statistica dei disturbi si identifica con quella del rumore termico.
Il canale trasmissivo gaussiano propriamente detto, esclusivamente caratterizzato - oltre che da una banda e un'attenuazione nota - dall'aggiunta di rumore termico, è stato il primo ad essere studiato e il primo a sollecitare studi sul modo di raggiungerne la capacità. Nei suoi lavori, Shannon ha dimostrato la celebre formula che individua appunto, in tale situazione, il "limite" di capacità di un canale di trasmissione dato da:

$$C = B \log (1+ S/N) \qquad (2)$$

In essa, il logaritmo è calcolato in base 2 e C rappresenta il massimo numero di bit al secondo trasmissibile in modo affidabile nel canale considerato. La capacità dipende, pertanto, dalla banda B che il canale è in grado di mettere a disposizione e dal rapporto S/N tra potenza del segnale e potenza del rumore

in ricezione. Una volta che questi parametri siano determinati, il teorema di Shannon afferma anche, in aggiunta a quanto detto nel paragrafo 1, che è *impossibile, anche con l'utilizzo di codici comunque complessi, trasmettere in modo affidabile a una velocità superiore a C*.
L'efficienza spettrale C/B del canale in corrispondenza del limite di Shannon è facilmente deducibile dalla stessa formula (2).
Si noti, inoltre, che, nella (2), la banda B compare anche nella valutazione del rumore N espresso da:

$$N = N_0 B \qquad (3)$$

essendo N_0 è la densità spettrale (unilatera) di potenza del rumore. Per un sistema di trasmissione a banda ipoteticamente infinita, è pertanto possibile definire ancora il limite di Shannon come limite per B che tende all'infinito dell'espressione (2), limite dato da:

$$C = (1/\ln 2)\,(S/N_0) \qquad (4)$$

Questa espressione esprime la velocità massima raggiungibile su un canale gaussiano in cui non vengano imposti limiti alla larghezza di banda utilizzabile.
Una esemplificazione del miglioramento di efficienza di potenza che l'uso di codici correttori può comportare è evidenziata nella Figura 2 per un canale affetto solo da rumore termico.

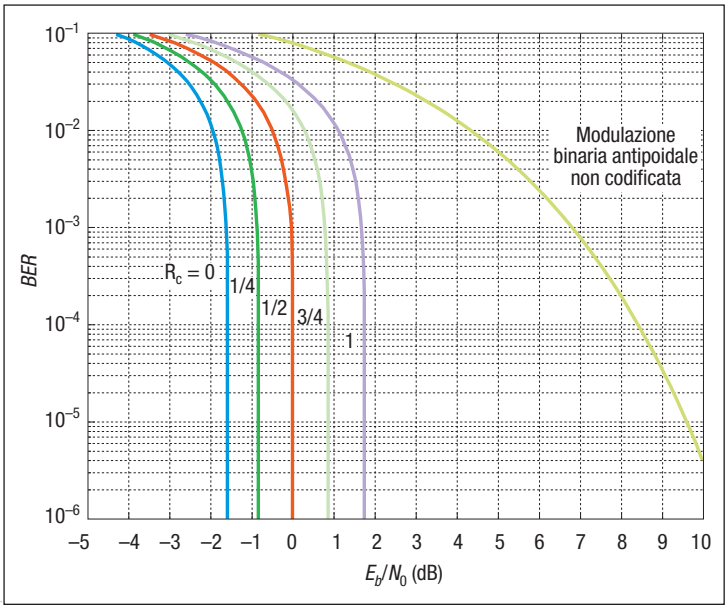


FIGURA 2
Miglioramento teorico (con decodifica soft) di efficienza di potenza, a diverse probabilità di errore

Nel diagramma il parametro R_c (detto *tasso di ridondanza del codice*) rappresenta la frazione di bit utili rispetto al numero totale di bit trasmessi; l'ordinata è la probabilità di errore (*Bit Error Rate*, *BER*) che non può essere superata, mentre l'ascissa riporta il rapporto segnale/rumore E_b/N_0 formula (1), espresso in decibel (dB). Codici piú potenti hanno un tasso minore, e dunque richiedono una maggiore larghezza di banda per la trasmissione. Al limite, un codice potentissimo trasmette una frazione trascurabile di bit utili, e dunque il suo tasso tende a zero: le prestazioni in queste condizioni (curva con $R_c = 0$) rappresentano una specie di "limite termodinamico" alle possibilità di trasmettere su un dato canale e designano quindi il "*limite assoluto di Shannon*", definito capacità del canale, ossia senza vincoli di efficienza spettrale e di tasso di ridondanza del codice. *Si può, in senso estensivo, definire limiti di Shannon anche quelli sotto il vincolo di una determinata efficienza spettrale dati dalle varie curve di figura 2 per differenti* R_c.
Per esempio, usando una modulazione binaria antipodale (cioè formata da due segnali di forma uguale ma in opposizione di fase: per esempio, un PSK binario) non codificata, per ottenere una probabilità di errore sul bit di 10^{-5} servono circa 9,6 dB di rapporto segnale/rumore. Viceversa, introducendo un codice con tasso di ridondanza R_c pari a $^1/_2$, la figura mostra che è sufficiente un rapporto S/N di 0 dB. In tal caso, i 9,6 dB costituiscono la distanza tra il sistema senza codici e il sistema con il migliore codice possibile a tasso $^1/_2$, distanza che le realizzazioni dei vari codici hanno progressivamente diminuito, avvicinandosi sempre di più al corrispondente valore teorico per tale tasso.
Il teorema di Shannon assume implicitamente che sul canale venga trasmesso un segnale la cui struttura è ottimale: nel caso del canale gaussiano, il segnale ottimo ha ampiezze che sono variabili casuali gaussiane. Nella pratica, verranno usati segnali (in termini tecnici, "schemi di modulazione") con un numero limitato di livelli. Nel campo delle modulazioni, le evoluzioni piú significative si sono avute con l'incremento progressivo del numero dei livelli, il che permette di migliorare l'efficienza spettrale, a scapito del rapporto

S/N necessario per un'assegnata probabilità di errore (Figura 3). Tali modulazioni, nella pratica degli ultimi anni, sono sempre associate a codici a controllo di errore, i quali permettono di avvicinarsi al limite di Shannon anche nei sistemi in cui prevale maggiormente l'esigenza di efficienza spettrale. La figura 3 descrive questo fatto illustrando i limiti che si possono teoricamente raggiungere *per una data efficienza spettrale R/B e un dato rapporto segnale/rumore E_b/N_0*: qui, la curva indicata "limite di Shannon" descrive il *compromesso ottimale tra efficienza in banda e rapporto segnale/rumore*, nel senso che nessun sistema consente di operare alla sua sinistra se viene richiesta una bassa probabilità di errore. Tale curva però prevede l'utilizzo di una struttura di segnali ottimale; ora, nella pratica, verranno usate costellazioni di segnali a struttura semplice (2PSK, 4PSK ecc.), le cui prestazioni si discostano da quelle ottimali come descritto nella figura 3 (per esempio, usando un 4PSK non si potrà mai superare $R/B = 2$, qualunque sia il rapporto segnale/rumore). Dunque, il limite di Shannon può essere utilizzato (come si farà nelle Figure 4 e 5) , non come limite assoluto, ma per descrivere le prestazioni ottimali di un sistema *vincolato* a utilizzare una determinata costellazione con un determinato tasso di codice (e, dunque, una determinata efficienza spettrale).

Per esempio, se si intende usare un codice con $R_c = 1/2$ e una modulazione 2PSK, il relativo limite di Shannon, in questo caso, sarà ottenuto dalla curva rossa di figura 2 come pari a circa 0 dB. Se si intende usare codici correttori a bassa ridondanza (R_c poco inferiore ad uno come avviene in varie applicazioni) al fine di aumentare l'efficienza spettrale, in tal caso, per sistemi binari, il relativo limite di Shannon è attorno a 1,5 dB.

Soltanto all'inizio degli anni '80 (anche se a livello teorico) l'introduzione del concetto di *riunificazione delle funzioni di codifica e modulazione* ha permesso un notevole miglioramento di efficacia dei codici. La teoria di quelli che venivano chiamati (e vengono chiamati ancora oggi, con una terminologia un po' obsoleta) "codici a correzione di errore" era stata motivata dall'idea di migliorare le prestazioni di un *modem* usando bit supplementari per compensare (e cioè correggere) gli errori introdotti dal demodulatore nella fase di decisione a causa di un insufficiente rapporto S/N. In questo contesto, il demodulatore prende una decisione su ognuno dei segnali che gli viene presentato, e passa il risultato della sua decisione al decodificatore di canale. Quest'ultimo utilizza la struttura del codice adottato per tentare di correggere gli eventuali errori commessi nelle decisioni prese dal demodulatore. Questo procedimento, detto di "decodifica algebrica", non è ottimale, perché per ogni decisione presa dal demodulatore questo scarta una parte dell'informazione, che potrebbe ancora essere utilizzata per la decodifica e uno degli insegnamenti fondamentali della teoria della comunicazione è che non bisogna mai scartare dell'informazione che potrebbe ancora essere utilizzata nel prendere la decisione finale.

Il salto concettuale consiste nell'amalgamare modulazione e codifica, discipline che fino agli anni '80 si erano evolute indipendentemente. In una visione integrata di modulazione e codifica, il demodulatore non commette errori che il decodificatore poi deve preoccuparsi di correggere, ma genera "decisioni

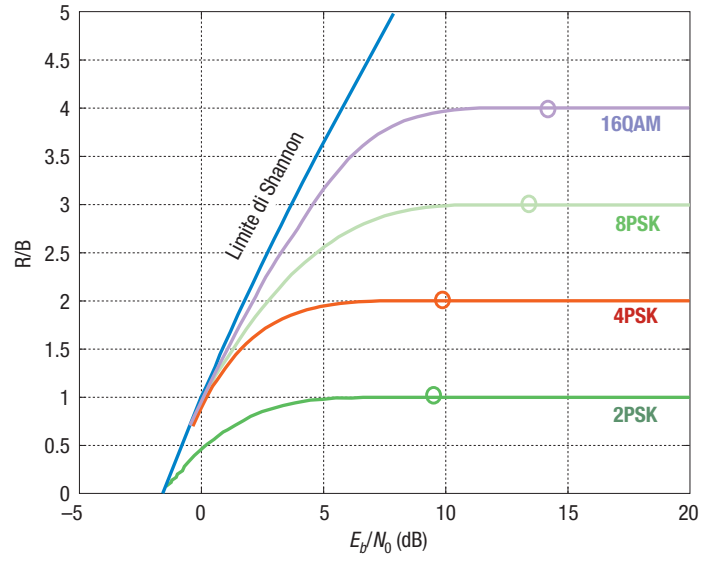


FIGURA 3

Efficienza spettrale R/B di informazione utile, in funzione del rapporto segnale-rumore E_b/N_0 per diverse modulazioni binarie e multilivello in assenza di codifica a controllo di errore (i cerchietti sulle curve della figura corrispondono ai punti con probabilità di errore di 10^{-5})

provvisorie" (indicate nel gergo come decisioni "soft") che non scartano alcuna informazione rilevante e che vengono utilizzate dal decodificatore per una decisione ottima. Questa decodifica soft può dare un miglioramento considerevole delle prestazioni: si suole affermare, con qualche approssimazione, che il miglioramento consentito è dell'ordine di 2 dB. Questa sinergia tra il demodulatore e il decodificatore fa sí che sia piú appropriato parlare di codici a *controllo di errore* piuttosto che di codici a *correzione di errore*. In questi casi, si dice anche che la codifica avviene nello spazio dei segnali.

La soluzione di combinare modulazione e codifica (generando così le cosiddette modulazioni codificate a traliccio, *Trellis Coded Modulation* (TCM), introdotte da Gottfried Ungerboeck, allora all'IBM di Zurigo, prendeva così il sopravvento nell'utilizzo delle modulazioni multivello. Per ottenere valori sempre più alti di efficienza spettrale venivano utilizzati segnali via via piú sofisticati quali le modulazioni di ampiezza e fase (QAM) con elevato numero di livelli (Figura 3), ovvero costellazioni "a reticolo", o ancora sistemi di modulazione opportunamente studiati per canali con forti limitazioni di banda (modulazioni a fase continua, *Continous Phase Modulation*, CPM). La richiesta di elevate efficienze in potenza veniva soddisfatta usando codici potenti: ma ora la ridondanza necessaria al controllo di errore veniva fornita, anziché da un incremento della banda, da un incremento del numero di livelli di segnale utilizzati per la trasmissione. Per esempio, se con un PSK binario è possibile trasmettere un bit al simbolo, usando un PSK quaternario ogni simbolo porta 2 bit: dunque si può usare uno di questi bit per portare informazione utile, e l'altro per la protezione dagli errori. L'introduzione del TCM, avvenuta all'inizio degli anni '80, chiarì, una volta per tutte, che la separazione di modulazione e codifica era dannosa all'efficienza del sistema, e che la migliore soluzione di progetto doveva consistere in un'integrazione delle due funzioni. Il TCM consente di ottenere guadagni di codifica senza un'espansione della banda. La figura 3 mostra, tra l'altro, il miglioramento di prestazioni ottenibile, nelle modulazioni multilivello, con integrazione TCM che porta ad avvicinarsi a poco più di 1.5 dal limite di Shannon. Un'ultima considerazione di carattere generale. L'uso del teorema di Shannon presuppone che, per una desiderata efficienza spettrale, il progettista sia libero di scegliere lo schema di modulazione che dia le prestazioni migliori. È importante, tuttavia, osservare che nei sistemi di comunicazione reali nascono considerazioni pratiche che spesso vincolano la scelta della modulazione. Per esempio, i trasmettitori radio che usano amplificatori di potenza non lineari sono agevolati nel loro compito se i segnali che vengono trasmessi hanno inviluppo costante, quali ad esempio quelli forniti dalle modulazioni PSK o dalla modulazione di fase continua. Se si deve operare in un ambiente radio in cui numerosi utenti condividono lo spettro, oltre ad avere elevata efficienza spettrale lo schema di modulazione deve anche essere robusto all'interferenza co-canale e, quindi, avere uno spettro compatto, con un lobo principale stretto e lobi secondari che si affievoliscono rapidamente quando la frequenza si discosta da quella della portante. Inoltre, per la trasmissione di dati sulla linea telefonica, le limitazioni di banda costringono all'uso di schemi di modulazione a molti livelli: questi impiegano una combinazione di modulazione di ampiezza e di fase, e permettono di raggiungere elevate velocità di trasmissione. Altre famiglie di modulazioni affrontano problematiche specifiche dei canali di trasmissione, quali ad esempio la presenza di disturbi concentrati su una porzione della banda disponibile e in continuo spostamento su di essa (è il caso dei raggi riflessi, dei cammini multipli e della diafonia). Per la sua importanza vale la pena, a tale proposito, citare la modulazione OFDM - impiegata nelle applicazioni dei sistemi digitali radio e televisivi di diffusione terrestre - e quella DMT, da essa derivata, adottata nei sistemi ADSL di cui si darà un ampio cenno nel seguito.

Va osservato che nella figura 3 l'efficienza spettrale in corrispondenza del "limite di Shannon" è riportata per un canale gaussiano e descrive il miglior compromesso fra potenza (in ascissa il rapporto E_b/N_0) ed efficienza spettrale (in ordinata il rapporto R/B) ottenibile usando una modulazione ideale. Il limite in questione definisce pertanto il mini-

mo S/N necessario per ottenere una prefissata efficienza spettrale. Tutte le altre curve della figura fanno riferimento ai valori ottenibili per un prefissato sistema di modulazione. Si noti, per esempio, che con una modulazione a 4 livelli (4PSK) la massima efficienza spettrale ottenibile (per un rapporto teorico E_b/N_0 teoricamente infinito) è di 2 bit/s/Hz. I cerchietti (o) su ogni curva indicano il punto in cui la probabilità di errore sul bit (*BER*) di un sistema senza codifica vale 10^{-5} (per esempio, usando un PSK quaternario 4PSK senza codifica, sono necessari 9,6 dB di rapporto E_b/N_0 per ottenere $BER = 10^{-5}$). La stessa efficienza spettrale $R/B = 2$ si può ottenere con un *codice a controllo di errore ottimale* per un rapporto E_b/N_0 di 1,8 dB: dunque l'utilizzo di un codice "opportuno" permette potenzialmente una trasmissione, a pari tasso di errore di 10^{-5}, con un risparmio di potenza di 7,8 dB (il cosiddetto *guadagno di codifica*).

4. METTERE IN PRATICA IL TEOREMA DI SHANNON NEI SISTEMI REALI

4.1. Generalità

Nel campo di cui si sta parlando, occorre sempre essere molto accorti a distinguere tra studi teorici di nuovi codici e realizzazioni pratiche: infatti, anche se l'integrazione ha dimostrato di consentire complessità assai alte, possono intercorrere molti anni tra uno studio teorico (o addirittura una dimostrazione di fattibilità) e l'uscita di un dispositivo per il mercato (è questo, ad esempio, il caso dell'ADSL, in cui sono occorsi più di dieci anni per realizzare i relativi dispositivi).
Nel campo dello studio teorico, i primi lavori sui codici sono stati soprattutto dedicati a sistemi limitati in potenza. In questo ambito, codici combinati con un sistema di modulazione binario possono risultare adeguati. Fin dagli anni '60 era noto che, usando codici convoluzionali accoppiati a un metodo di decodifica detto "sequenziale", si poteva avvicinare il limite di Shannon fino a circa 3 dB. In questo periodo, molti ritenevano che non si potesse fare meglio di così, a meno di non incappare in complessità inaccettabili nel decodificatore, e che, quindi, il problema di raggiungere la capacità fosse "sostanzialmente" risolto.
Tra le famiglie di codici che godevano di una vasta popolarità teorica grazie alla loro elegante struttura algebrica, un ruolo particolarmente importante ebbero i codici di Reed-Solomon, grazie a due importanti applicazioni. La prima nacque negli anni '80, quando si osservò come la concatenazione di un codice di Reed-Solomon con un codice convoluzionale desse le migliori prestazioni note, quantunque al prezzo di una complessità notevole: alla probabilità di errore di 10^{-5} questo sistema si discostava dal limite di Shannon di circa 2,3 dB. Questi risultati furono fondamentali per le applicazioni pratiche sui satelliti di televisione numerica per le trasmissioni iniziate dopo la metà degli anni '90. La seconda (l'impiego nei lettori semiprofessionali di *Compact Disc*) dimostrò che la complessità di questi codici, prima ritenuta troppo elevata, poteva, in realtà, essere tollerata quando una spinta prettamente commerciale ne promosse la spinta realizzativa per le interessanti quantità in gioco. Tra le ulteriori applicazioni professionali dei codici nel campo delle telecomunicazioni, un'importanza particolare ebbe il progetto dei sistemi per fibra ottica sottomarina, a causa della estrema lunghezza delle tratte e, quindi, della necessità di ridurre la potenza necessaria per l'assegnata qualità di servizio.
Un nuovo, spettacolare impulso alla teoria dei codici venne negli anni '90, quando si dimostrò che il limite di Shannon poteva veramente essere raggiunto: i "turbo codici", inventati in Francia da Claude Berrou e Alain Glavieux e presentati per la prima volta alla comunità scientifica in un congresso a Ginevra nel 1993, avvicinano il limite di Shannon fino a una frazione di dB. In particolare, i turbo codici hanno eccellenti prestazioni per probabilità di errore non migliori di 10^{-4} o 10^{-5}. Per probabilità di errore inferiori la loro qualità è meno buona. Inoltre, a causa della loro complessità di decodifica la presenza di un turbo codice causa un notevole ritardo, il che può limitare il campo della loro applicabilità a sistemi in cui i ritardi di elaborazione non siano particolarmente dannosi (televisione, trasmissione di dati ecc.).
L'invenzione dei turbo codici convinse anche i più scettici sulla possibilità pratica di avvicinarsi alla capacità di un canale di trasmis-

sione: sono state, quindi, riprese vecchie idee e riesaminate alla luce di questi nuovi risultati. La piú interessante di queste riesumazioni è stata indubbiamente quella relativa ai codici con bassa densità di controlli di parità (*Low-Density Parity-Check Codes*, codici LDPC), studiati da Robert Gallager al MIT (*Massachusetts Institute of Technology*) intorno al 1960, ma le cui prestazioni effettive non si potevano esaminare a quel tempo per mancanza di mezzi di calcolo sufficientemente potenti. Un'analisi accurata degli LDPC ha dimostrato che essi sono almeno altrettanto potenti dei turbo codici. Lo stato attuale della ricerca genera codici LDPC che si scostano dal limite di Shannon di 4 centesimi di dB! Ovviamente, ambedue gli aspetti, teorici e realizzativi, sono molto importanti. Negli esempi successivi, si intende dare una breve rassegna di vari sistemi - e della relativa storia di avvicinamento al limite di Shannon - solo con riferimento a effettive realizzazioni di carattere industriale.

4.2. Sistemi su canale telefonico analogico

Una storia di successo per quanto riguarda la ricerca delle strade per avvicinarsi al limite di Shannon riguarda il canale telefonico analogico, o, piú precisamente, il mezzo trasmissivo costituito dalla linea telefonica commutata. Questo mezzo era stato progettato per la trasmissione della voce, ma la necessità di trasmettere dati, aggiunta al fatto che la rete telefonica è estremamente capillare, stimolarono una intensa attività di ricerca dedicata alla trasmissione di dati su questo mezzo. La rete telefonica convenzionale utilizza una banda di circa 3000 Hz. La capacità teorica del canale telefonico si può calcolare, scoprendo che essa è, per un rapporto S/N di 33 dB (considerato tipico) di circa 36 kbit al secondo. L'utilizzo di tecniche di codifica e modulazione congiunta portarono, dopo circa 30 anni di sviluppo e ricerca sui modem, a raggiungere essenzialmente la capacità grazie allo standard V.34 bis. I primi modem per linea telefonica commutata potevano trasmettere dati alla velocità di 300 bit/s, secondo uno standard chiamato Bell 103 .

Alcuni standard per i modem operanti sulle linee telefoniche sono indicate con una "V" seguita da un numero che ne descrive velocità e altre caratteristiche (Tabella 1). Per esempio, i modem V32 bis trasmettono fino a 14.4 kbit/s, i V34 fino a 28.8 kbit/s. È importante notare che, nella pratica, la possibilità di superare con affidabilità i 2400 bit/s sulle linee commutate, si rese possibile solo dopo l'introduzione della commutazione elettronica: infatti, la commutazione telefonica elettromeccanica introduceva disturbi impulsivi che ostacolavano velocità superiori.

Raggiunta sostanzialmente la capacità del canale telefonico, si osservò che era possibile fare ancora meglio. Infatti, la vecchia "telefonia analogica" è stata oggi soppiantata da un nuovo sistema in cui il segnale telefonico viene trasmesso, con tecniche PCM, in forma numerica. Il canale numerico così generato può allora essere usato direttamente per la trasmissione di dati, evitando di trasformare questi ultimi in un segnale analogico per fare loro attraversare un canale progettato, invece, per la voce. Questa idea ha permesso di progettare modem che operano a 56 kbit/s (standard V.90).

Velocità di trasmissione	Standard
300 baud	Bell 103
300 baud	V.21
1,200 bit/s	Bell 212A
1,200 bit/s	V.22
1.2 or 2.4 kbit/s	V.22bis
4.8, 7.2 o 9.6 kbit/s	V.29
9,600 bit/s	V.32
14,400 bit/s	V.32 bis
28,800 bit/s	V.34
36,000 bit/s	V.34 bis
56,000 bit/s	V.90

TABELLA 1
L'evoluzione dei modem dati in banda vocale

4.3. Sistemi su doppino di utente

Raggiunto l'obiettivo di cui si è parlato nel precedente paragrafo, l'attenzione si fissò al supporto fisico (il doppino telefonico, cioè la coppia di fili di rame intrecciati per l'accesso

all'utente domestico) che porta il segnale telefonico dal terminale di utente alla centrale di commutazione. Poiché questo è un canale la cui capacità è ben maggiore di 56 kbit/s, venne naturale tentare la sua utilizzazione per trasmettervi dati a piú alte velocità.
Un effetto che limita la banda e la potenza del segnale che transita sul doppino, e dunque la sua capacità, è la forte attenuazione cui è sottoposto il segnale nel suo tragitto lungo la linea, con le alte frequenze affette maggiormente. Si possono, quindi, ottenere capacità piú elevate se le linee sono corte. Joseph W. Lechleider, un ingegnere della Bellcore (oggi Telcordia) nel 1980 propose di usare una linea telefonica ordinaria come canale a elevata larghezza di banda, sul breve tragitto che separa l'utente dalla centrale telefonica. Questa tecnica di trasmissione fu chiamata "linea numerica di utente" (*Digital Subscriber Line*, DSL). Nei primi anni '90 alcune ditte svilupparono sistemi (noti come HDSL, con "*H*" che sta per *High-Bit-Rate*) che potevano trasmettere quasi 800 kbit/s a una distanza di 4 km. Contemporaneamente allo sviluppo dell'HDSL, John Cioffi dell'Università Stanford dimostrò agli inizi degli anni '90 una tecnica di codifica del segnale chiamata *Discrete Multitone*, usandola per trasmettere piú di 8 Mbit/s su una coppia telefonica lunga oltre 1,6 km. Questa tecnica suddivide la banda totale di 1 MHz in 256 sottocanali di circa 4 kHz ciascuno. In sostanza, vengono creati 256 modem virtuali sulla stessa linea.
Originariamente, l'approccio di Cioffi era pensato per inviare segnali video sulle linee telefoniche. Poiché questa applicazione si affida principalmente a una trasmissione unidirezionale, la maggioranza dei sottocanali erano dedicati al segnale che transita verso l'utente, che portava circa 6 Mbit/s, con circa 0,6 Mbit/s disponibili nella direzione opposta. Questa forma di DSL asimmetrica, nota come ADSL[1] è ora uno standard mondiale. Quantunque l'applicazione ai segnali video non abbia dato frutti, la trasmissione asimmetrica si è verificata estremamente utile per le applicazioni collegate a Internet, dove il flusso di dati dal singolo utente verso la periferia è molto inferiore al flusso entrante. ADSL viene installata nelle reti telefoniche per accesso a Internet *always on*, operando tipicamente a velocità di centinaia di kbit/s o superiori su linee telefoniche lunghe oltre 5 km. Poiché i dati possono usare canali che operano sopra le frequenze vocali, un unico terminale può trasmettere simultaneamente sia la voce che i dati ad alta velocità.
La modulazione DMT rappresenta un ottimo esempio di incremento di efficienza spettrale per un particolare canale di trasmissione che va protetto da vari tipi di disturbi; inoltre, l'intero sistema è un modello esemplare di combinazione tra modulazione e codici correttori (a tal proposito si veda il riquadro a p. 36).

4.4. Sistemi su fibra ottica nell'area di trasporto

Tra i vari tipi di degrado trasmissivo che impattano sui sistemi di trasmissione su Fibra Ottica per lunghe distanze si possono elencare le non linearità ottiche, la dispersione cromatica, la dispersione di polarizzazione ed il rumore degli amplificatori ottici. Al crescere della velocità di trasmissione molti di questi effetti diventano più pronunciati.
In particolare, i nuovi sistemi di trasmissione a 10 e 40 Gbit/s avrebbero richiesto una significativa riduzione della tratta di ripetizione se fossero stati realizzati con le tecnologie tradizionali, con conseguenze costose e complesse sugli impianti già posati. Per fortuna l'applicazione di nuove soluzioni tecnologiche quali l'amplificazione Raman e l'impiego di potenti FEC (*Forward Error Correction*) hanno consentito di mitigare largamente il problema.
Il FEC è stato inizialmente introdotto durante i primi anni '90 nei sistemi di trasmissione sottomarini, nei quali la necessità di limitare al minimo il numero di punti di amplificazione intermedia è fondamentale. In seguito l'incremento della capacità di trasmissione delle grandi dorsali terrestri e la necessità di contenere i costi, ha suggerito l'adozione del FEC (peraltro già realizzati in forma integrata) anche per le tratte terrestri.

[1] Un'illustrazione completa dei sistemi ADSL è stata pubblicata sul numero 6 di questa stessa Rivista.

Progredendo in senso storico, le prime applicazioni di FEC nelle comunicazioni si verificano negli anni compresi tra il 1985 e il 1987. In questo periodo venne sviluppato un sistema pionieristico per la digitalizzazione del segnale TV normale che permettesse al tempo stesso di ridurre la forte ridondanza di sorgente di tale segnale. Il sistema impiegava per la prima volta nella pratica un codice di Bose-Chaudhuri-Hocquenghem BCH (255, 239) per la digitalizzazione del segnale televisivo standard a 34 Mbit/s con codifica ADPCM (*Adaptive Differential Pulse Code Modulation*). A questo seguì, nel 1989, conseguente a una cooperazione tutta italiana tra ricercatori Rai e Telettra, il corrispondente problema per la codifica di un segnale TV ad alta definizione con un sistema antesignano dell'attuale MPEG che utilizzava per la trasmissione un codice Reed-Solomon, RS (255, 239), più efficace del BCH e realizzato su un singolo chip che lavorava alla velocità limite di quel tempo di 320 Mbit/s.
Fu proprio questo chip che diede l'avvio agli sviluppi per l'impiego di FEC nel primo sistema in Fibra Ottica a 565 Mbit/s, progettato per tratte di rigenerazione lunghe fino a 200 km, e realizzato in Italia nel 1992 per la posa sottomarina della fibra nel sistema cosiddetto dei "Festoni" (collegamento delle città vicino alla costa con sistemi in fibra ottica sottomarini).
Al risultato particolarmente lusinghiero di questa prima applicazione sono seguite nuove e sempre più efficaci soluzioni FEC impiegate nei sistemi di trasmissione su Fibra Ottica sottomarini e terrestri. Nel frattempo, il moltiplicarsi delle applicazioni ha continuamente sospinto schiere di ricercatori a sviluppare nuovi studi per codifiche FEC più efficaci. Oggi due standardizzazioni diverse sono sviluppate in ITU-T e ANSI per soluzioni di codifica FEC. La prima è una soluzione SONET/SDH *in-band* documentata nella Raccomandazione G707 di ITU. La seconda *out of band* è il cosiddetto *digital wrapper* documentato nella Raccomandazione G709.

In-band FEC

Il concetto di quello che in gergo viene definito "*in-band FEC*" è di non modificare la velocità di linea già adottata in precedenza, utilizzando per il FEC i soli bit liberi contemplati nella trama del segnale mantenendo in tal modo una bit-rate di linea esattamente coincidente con quella prevista dagli standard già esistenti (STM-N).
L'aggiungere il FEC agli standard SONET/SDH presenta una quantità di vincoli, per primo quello della trama fissata in 125:s. Normalmente, infatti, un codice è studiato su una determinata sequenza di bit o simboli, ma avendo fissato il tempo di trama, questa cambia al cambiare del livello gerarchico considerato.
Un secondo vincolo deriva dal limitato numero di byte disponibili in trama per allocare la ridondanza di codifica. In particolare, nei più bassi livelli gerarchici di SONET/SDH non ci sono praticamente byte disponibili per il FEC. La conseguenza è che l'*in-band* FEC della norma G707 si applica solamente ai sistemi a 10 e 40 Gbit/s, per i quali è proposta una codifica BCH.
L'unico vantaggio, peraltro non trascurabile, dei sistemi con *in-band FEC* è la compatibilità con i sistemi precedenti già esistenti.

Out of band FEC

L'"*out of band FEC*", definito nella Raccomandazione G709, impiega, invece, una diversa soluzione essendo basato su una lunghezza di trama fissa: stabilisce cioè il numero di bit invece del tempo. Adotta, inoltre, una nuova struttura di trama, nella quale il *pay-load* è esattamente lo standard SONET/SDH a cui viene aggiunto un *overhead* per la codifica. Il codice utilizzato è un Reed –Solomon RS (255, 239). In altre parole viene ad essere un applicazione classica di codice a controllo di errore con introduzione di una certa ridondanza per raggiungere gli obiettivi di qualità che ci si prefigge.
Altre soluzioni FEC più efficienti sono possibili se compatibili con la struttura di supertrama proposta. Le soluzioni più avanzate di FEC oggi utilizzate fanno uso di codici concatenati quali RS-RS, RS-BCH, BCH-BCH, con decodifica soft ed interattiva.
La figura 4 indica i miglioramenti di prestazioni ottenuti adottando diversi tipi di FEC. Per limite di Shannon in questa figura si intende quello riferito alla decodifica soft ma con un tasso di ridondanza del codice di poco inferiore ad 1. Il vantaggio va pertanto con-

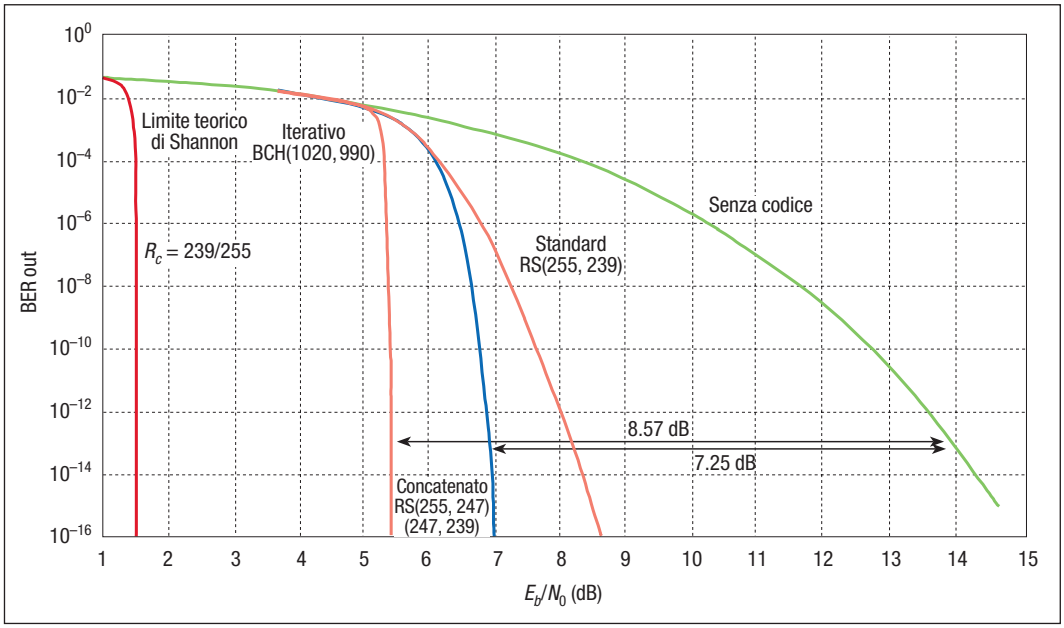


FIGURA 4

Miglioramento di prestazioni (Guadagno di codice) ottenuto con diversi tipi di FEC (Forward Error Correction) con decodifica hard in un sistema in fibra ottica con modulazione binaria. (Fonte: Alcatel Italia SpA)

frontato con quello della figura 2 per R_c prossimo ad 1.
È importante ricordare che I primi codici correttori RS (*Reed-Solomon*) per trasmissioni su fibra furono inseriti nel progetto dei cavi sottomarini dove era essenziale superare distanze molto alte senza amplificazione. Peraltro, dopo la progettazione dell'ASIC che integrava il codice, i costruttori di apparati li hanno utilizzati anche per i collegamenti terrestri ottenendo un vantaggio di 5-6 dB sulla sensibilità del ricevitore o sulla potenza in uscita, (od una combinazione delle due), prestazione tutt'altro che da disprezzare anche sui collegamenti terrestri in fibra ottica, per quanto meno esasperati di quelli sottomarini.

4.5. Sistemi via satellite e terrestri per diffusione televisiva

Nel campo della trasmissione televisiva, è interessante richiamare la storia della nascita della televisione digitale da satellite e, più recentemente quella della televisione digitale terrestre.
Alla metà degli anni '90 si è diffusa con grande rapidità la televisione digitale da satellite, grazie soprattutto delle forti economie di trasporto che era in grado di realizzare. In particolare, mantenendo gli stessi tipi di satelliti già usati per la diffusione analogica, si è riusciti ad aumentare di 5-10 volte la velocità di trasmissione in programmi (a seconda della qualità desiderata), con un proporzionale abbattimento dei costi di diffusione.
Per quanto riguarda questo straordinario miglioramento con un segnale digitale, il primo merito va alla codifica di sorgente, ossia all'impiego della compressione MPEG-2 che elimina molta ridondanza del segnale. Per quanto riguarda il canale trasmissivo costituito dal satellite, si hanno, peraltro, altri meriti fondamentali derivati dall'impiego di opportuni codici a controllo di errore.
Un satellite televisivo è normalmente organizzato con 9-18 *transponder* che costituiscono a tutti gli effetti degli amplificatori che lavorano in saturazione (ciò è dovuto alle caratteristiche dei tubi ad onda progressiva che operano come amplificatori a microonde nella gamma di frequenza dei 12 GHz). Ogni trasponder era stato concepito per inviare un solo canale televisivo analogico in modulazione di frequenza, necessaria, grazie al suo inviluppo costante, ad evitare gli effetti di non linearità nell'amplifica-

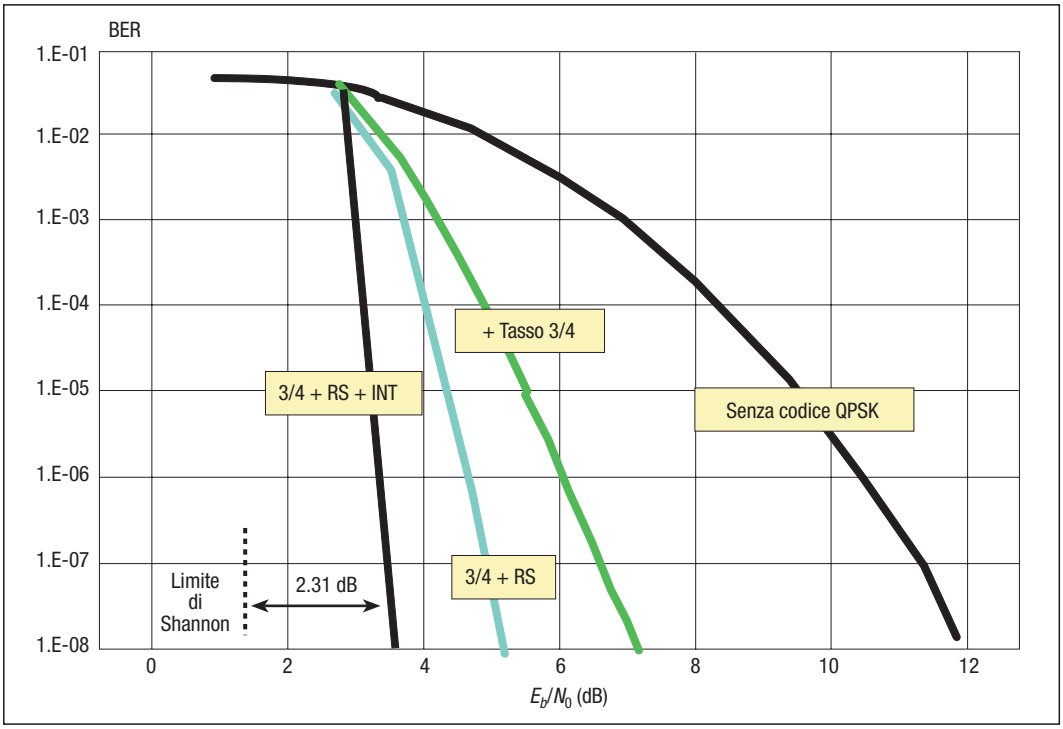


FIGURA 5

Riduzioni ottenibili, a pari probabilità di errore, del rapporto E_b/N_0 al ricevitore in una trasmissione satellitare televisiva con standard DVB-S che prevede l'impiego di due codici correttori in cascata (convoluzionale $^3/_4$ e Reed-Solomon a tasso 7/8 nonché la funzione di "interleaver") e modulazione QPSK (coincidente con 4PSK). Il limite di Shannon indicato in questa figura è quello relativo al tipo di modulazione ed al tasso di ridondanza della cascata di codici. *(Fonte: "Centro Ricerca ed Innovazione Tecnologica" della RAI)*

zione fornita. Un transponder ha tipicamente per ragioni storiche 36 MHz di banda: in tale banda, senza alcuna modifica del satellite, è stato possibile introdurre un segnale digitale operando con una modulazione 4PSK (il numero basso di livelli e l'inviluppo costante sono vincoli imposti dagli effetti di non linearità dell'amplificazione). Si trattava di trovare il miglior compromesso tra carico utile assegnabile alla capacità digitale trasmettibile e la necessaria ridondanza che si doveva creare per facilitare la trasmissione da una distanza di 38.000 km, sia agli effetti della potenza trasmessa dal satellite (che doveva rimanere identica al caso dell'analogico) sia per ridurre al minimo il diametro delle parabole di ricezione.

Il compromesso fu trovato, tenuto conto del vincolo della modulazione 4PSK, attraverso l'impiego di due codici "in cascata" e dell'interlacciamento temporale: il primo è un Reed-Solomon di tasso 7/8 (7 bit utili su ogni 8 inviati), il secondo è un codice convoluzionale (lasciato libero dallo standard ma generalmente impiegato con tasso $^3/_4$) con decodifica "soft" (consentita dall'*algoritmo di Viterbi*). In questa configurazione, la ridondanza introdotta era ammissibile tenuto conto della larga banda disponibile nel transponder.

Il complesso dei parametri sopra accennati ha dato origine allo standard DVB-S (*Digital Video Broadcasting-Satellite*) che permette di introdurre in un transponder 32 Mbit/s di velocità utile corrispondente appunto a 5-10 programmi televisivi digitali codificati in MPEG-2.

La figura 5 mostra, per diverse condizioni di rapporto S/N e di probabilità di errore, i miglioramenti in efficienza di potenza ottenibili con l'impiego dei codici a controllo di errore e dell'interallacciamento temporale, sistemi tutti adottati nello standard DVB-S.

Nella figura si può notare come, per una probabilità di errore di 10^{-5}, si ha un miglioramento introdotto dal solo codice convoluzionale (con rate $^3/_4$) di circa 4 dB, un migliora-

mento di 5.5 se il codice convoluzionale è combinato con un codice Reed-Solomon (a tasso 7/8) ed infine un miglioramento complessivo, come stabilito dallo standard DVB-S, di 7 dB se si aggiunge anche l'interlacciamento temporale
È stato da poco completato lo studio di un nuovo standard DVB-S2 che, con modulazioni 8PSK e 16PSK e l'introduzione di potenti codici a controllo di errore, si propone di aumentare del 30% la capacità e offrire prestazioni migliori per comunicazioni a banda larga. In queste evoluzioni occorre, però, fare i conti con la presenza di uno standard già esistente e i problemi di compatibilità diventano essenziali.
Per analogia vale la pena di ricordare anche i grandi progressi nell'esplorazione dello spazio profondo con sonde dove, non esistendo problemi di compatibilità, si è riusciti a migliorare le prestazioni di ordini di grandezza, per di più inviando i nuovi codici in formato software a sonde partite molti anni fa! Qui il problema della complessità di decodifica (e del conseguente ritardo) non è cosi cruciale: infatti, i segnali ricevuti non devono necessariamente essere decodificati in tempo reale! Uno dei primi codici usati per questa applicazione (sonda Mariner 5, 1969) aveva ancora prestazioni distanti 7,5 dB dal limite, mentre il codice piú moderno e potente usato per la sonda Pioneer 12 negli anni '70 arrivava a 2,5 dB. Per le missioni odierne si stanno considerando, naturalmente, i turbo codici.
Infine, vale la pena di accennare che i sistemi televisivi digitali terrestri (di cui tanto si parla in questo momento) adottano le stesse strategie di codici del satellite, ma impiegano la modulazione OFDM (si veda il riquadro a p. 36), sia perché la saturazione non costituisce un problema, sia perché garantisce una migliore efficienza spettrale e difende bene dagli effetti negativi degli echi e dei cammini multipli di propagazione.

4.6. Sistemi wireless di quarta generazione per accesso a banda larga

I sistemi *wireless* più recenti proposti in questo campo (Hiperaccess e standard 802.16 IEEE) - che estendono le prestazione dei noti standard Wi-Fi - portano all'estremo limite la ricerca dell'efficienza spettrale con tutta una serie di soluzioni (codici e modulazioni) assai sofisticate.
Senza entrare in troppi dettagli tecnici si dà solo un brevissimo cenno allo studio di questi nuovi sistemi BWA (*Broadband Wireless Access*) in cui l'approssimare il più possibile il limite di Shannon viene considerata un'esigenza primaria - più stringente dei sistemi di accesso su portante fisico – per realizzare al meglio trasferimenti di grossi *file* di dati, servizi multimediali (audio, musica, videoconferenze), sistemi di *Video on Demand* e accesso Internet ad alta velocità in bande assai strette. Tutti questi servizi richiedono un'alta velocità di trasferimento con bassi ritardi e basso *BER* (*Bit Error Rate*).
Ottenere queste prestazioni è molto difficile, dal momento che per questi tipi di accesso radio si è resa necessaria la scelta di frequenze non millimetriche, al fine di avere collegamenti non on *sight* (2.5 – 10 GHz), frequenze per cui la disponibilità di banda è notevolmente limitata. Inoltre, l'esistenza dei cammini multipli porta a una degradazione del rapporto S/N (e quindi del *BER*) continuamente e rapidamente variabile nel tempo. A causa di tutti questi problemi, la sfida tecnologica è molto più alta nei sistemi wireless rispetto ai sistemi cablati e, per sistemi di accesso wireless a banda larga (canali BAW), occorre dinamicamente ridurre la velocità massima - in caso di degradazione del canale - per mantenere la qualità di servizio desiderata.
Per questi tipi di sistemi, l'efficienza spettrale effettiva è definita come il valor medio di velocità (*bit-rate*) di informazione per tutti i rapporti S/N che si hanno nella banda del canale considerato, tenendo in conto le degradazioni statistiche che il canale subisce. Analogamente, per questi tipi di applicazione, si suole introdurre un parametro detto MASE (*Maximum Average Spectral Efficiency*) che rappresenta il rapporto tra il *valor medio* (Cav) della "capacità" C di Shannon per tutti i possibili rapporti S/N ottenibili sul canale e la banda B del canale (MASE = Cav/B).
Per far sì che per questi canali, fortemente dispersivi, l'efficienza spettrale effettiva sia il più possibile prossima al MASE, si stanno studiando nuove soluzioni assai sofisticate

Schema trasmissivo e tipo di modulazione del sistema ADSL

Lo schema trasmissivo del sistema ADSL (Figura **A**) è particolarmente innovativo e realizza, per il doppino di utente, un sistema che approssima sensibilmente i limiti teorici del teorema di Shannon. Si fa notare che, anche dopo l'impiego dell'ISDN, il doppino rimaneva alquanto distante da tale limite.

Per quanto riguarda la modulazione, il sistema ADSL impiega un tipo di modulazione che può considerarsi un perfezionamento della modulazione OFDM (largamente usata nei sistemi diffusivi digitale terrestri per radio e televisione con analogo schema trasmissivo della figura 1) resa possibile dalla presenza di un canale di ritorno. Sia la modulazione OFDM che quella DMT, oltre ad consentire un'ottima utilizzazione dello spettro, sono state studiate per essere particolarmente robuste ai disturbi provocati da interferenze.

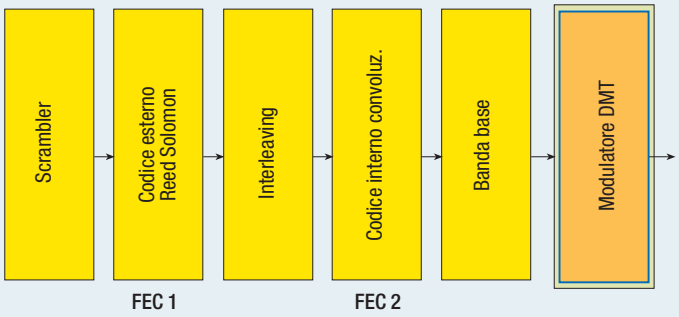


Figura A *Schema del sistema di trasmissione ADSL*

La funzione FEC (*Forward Error Correction*) è realizzata con l'insieme di due codici correttori. Il primo è il *codice Reed-Solomon* (FEC 1), detto anche codice esterno, che introduce nella trama del segnale una ridondanza contenente un algoritmo predefinito per la migliore correzione degli errori. In particolare, sono utilizzati a questo scopo 16 bytes della trama complessiva costituita da 255 bytes (6.3%).

Il FEC 1 è seguito, in trasmissione, dalla funzione di "*time interleaving*" (il circuito che la realizza è denominato "*outer interleaver*") che consiste nel distribuire (attraverso opportune memorie a *buffer*) la successione temporale dei bit del segnale d'ingresso su un campo molto più vasto e secondo una strategia opportuna. In ricezione si opera l'operazione inversa con un riallineamento dei bit secondo la sequenza originaria.

Il *time interleaving* è una tecnica semplice che non ha necessità di introdurre ridondanza ed ha il solo problema di aumentare il ritardo: tuttavia, è molto importante poiché riesce, in combinazione col FEC 1, a eliminare gli errori conseguenti a disturbi di tipo impulsivo che possono distruggere lunghe sequenze consequenziali di bit che non avrebbero possibilità di essere protette dai normali codici di canale. Con l'*interleaving*, invece, un disturbo di carattere impulsivo si limita, nella realtà, a distruggere bit non sequenziali nel flusso del segnale originario, bit che possono essere ricostruiti impiegando semplici codici correttori quali il Reed-Solomon. In caso di sistemi ADSL che non soffrano di disturbi di carattere impulsivo, è possibile *disabilitare l'interleaving per migliorare la prestazione del sistema in termini di tempo di ritardo*.

Un successivo circuito di FEC 2 è realizzato attraverso l'impiego di un codice a traliccio (*trellis code*), che costituisce un ulteriore meccanismo di rivelazione e correzione di errore ed è opzionale per l'ADSL. Il principio del *trellis code* si basa sull'osservazione di lunghe sequenze formate da più trame ADSL. La realizzazione di questo codice richiede un'aggiunta di quattro bit e mezzo a ogni trama di 255 bytes: quattro bytes sono usati per il codice e mezzo per l'allineamento. La struttura di trama complessiva è riportata nella figura **B**.

Il vantaggio conseguente alla codifica, misurato in termini di riduzione del rapporto segnale-rumore necessario per un tasso di errore di 1 per 10-7, è di circa 5.5 dB per la sola codifica *Reed-Solomon* (RS), e diventa di 9 dB con l'impiego congiunto di RS e trellis. Va notato, inoltre, che alcuni servizi trasmessi su ADSL, possono avere ulteriori propri provvedimenti di protezione contro gli errori di trasmissione. Per esempio, nel VoD (*Video on Demand*), lo schema di compressione video (ad esempio con MPEG 2), può includere il proprio mascheramento di errore.

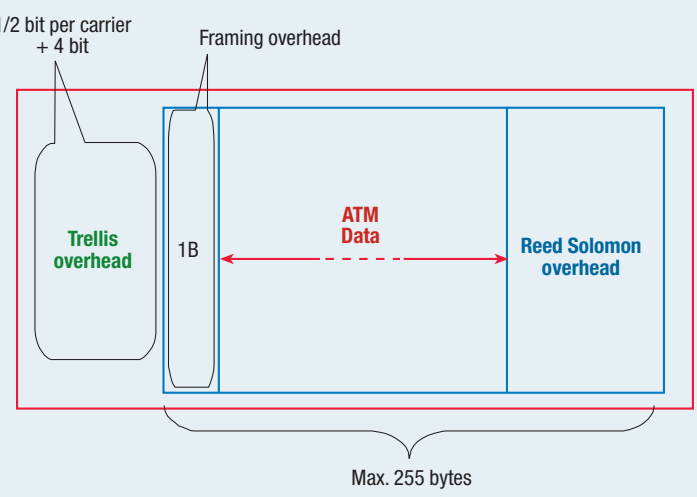


Figura B *Struttura di trama complessiva di un modem ADSL*

Dopo l'introduzione dei codici correttori e del "*time interleaving*", si passa al modulatore DMT (*Discrete Multi Tone modulation*), particolarmente indicato quando si hanno disturbi variabili sulla banda spettrale occupata per il sistema trasmissivo.

La DMT è un perfezionamento della modulazione OFDM (*Orthogonal Frequency Division Multiplex*) che è fondamentalmente una tec-

nica di "spread spectrum": in essa, il flusso digitale seriale (che porta il segnale dati) è ripartito in flussi paralleli (con velocità assai minore) che vanno a modulare in QAM (*Quadrature Amplitude Modulation*), con un opportuno numero di livelli, un elevato numero di portanti nello spettro a disposizione. Questa tecnica prende il nome di "*frequency interleaving*" e corrisponde, nel dominio delle frequenze, all'operazione effettuata sui bit nel dominio dei tempi attraverso il "*time interleaving*". In un sistema ADSL il numero di portanti (denominati in questo caso "toni") è normalmente di 255. Lo schema complessivo del sistema di modulazione OFDM è rappresentato nella figura **C**.

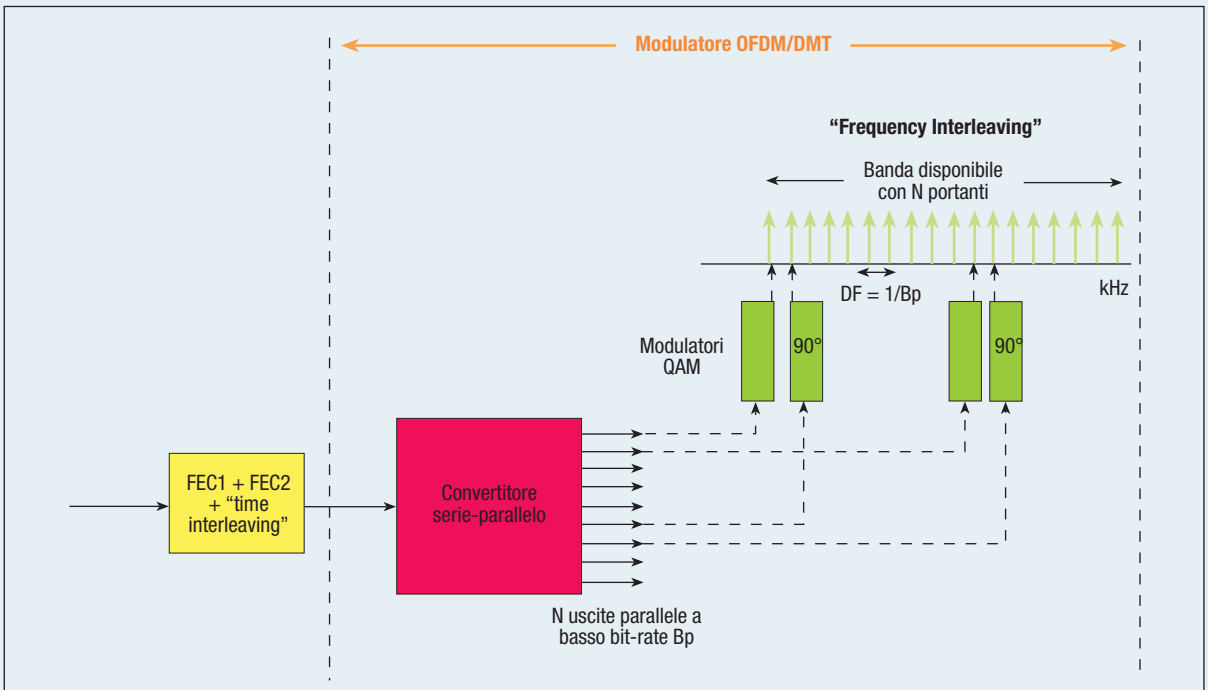


Figura C *Schema di principio della modulazione OFDM/DMT*

Per la metodologia seguita, i bit vicini temporalmente vanno a modulare portanti poste in zone distanti dello spettro di frequenza: poiché i disturbi conseguenti a interferenze, si traducono dal punto di vista spettrale, in un deterioramento considerevole o nella totale distruzione di strette frazioni di spettro – che si muovono in continuazione sulla banda a seconda della fase dei disturbi - l'effetto combinato delle due tecniche di *frequency* e *time interleaving* permette che non siano distrutte lunghe sequenze consecutive di bit per i disturbi di carattere impulsivo sia temporali che spettrali, disturbi che creano distruzioni consecutive di bit difficilmente proteggibili, senza accorgimenti del tipo descritto, con codici correttori convenzionali. La modulazione OFDM studiata teoricamente per la prima volta negli anni 50 nei Laboratori Bell, nonostante i vantaggi molto cospicui da essa presentata, era improponibile nella sua realizzazione pratica perché avrebbe richiesto una batteria elevatissima di modulatori QAM in numero pari, cioè, alle sottobande necessarie per conseguire i vantaggi desiderati. Si può, tuttavia, dimostrare teoricamente che l'intera operazione sopra descritta - a partire dal convertitore serie-parallelo di figura **C** fino al corrispondente circuito parallelo-serie di ricezione - equivale a effettuare una trasformata finita di Fourier sul segnale sequenziale d'entrata. Questo algoritmo, con i progressi della tecnologia, è effettuabile solo da pochi anni con un singolo DSP (*Digital Signal Processor*) integrabile in un unico "chip" accanto a tutte le altre funzioni digitali del trasmettitore e del ricevitore. L'integrazione in un circuito monolitico ha permesso la realizzabilità fisica dello schema e, al contempo, l'abbattimento dei costi per cui la corrispondente tecnologia è stata introdotta nella diffusione televisiva digitale DVB-T (*Digital Video Broadcasting-Terrestrial*) e in quella radiofonica DAB (*Digital Audio Broadcasting*). Nel caso dei sistemi ADSL, la modulazione OFDM è stata ulteriormente migliorata (a spese di un necessario aumento di complessità) e ha preso il nome il nome DMT (*Discrete Multi Tone modulation*). Nella DMT, oltre allo schema dell'OFDM, velocità di cifra e numero di livelli impiegati per modulare i toni in QAM vengono ottimizzati per ottenere uno sfruttamento ottimale dello spettro compatibilmente con le condizioni di disturbo nello spettro occupato. La misura delle condizioni di rumore di ciascuna sottobanda è effettuata attraverso l'invio di segnali pilota sul canale *upstream* di ritorno: le informazioni ricavate dai toni pilota permettono di comandare l'adeguamento del numero di livelli QAM di modulazione da associare a una particolare sottobanda. Il sistema di modulazione DMT provvede, perciò, automaticamente a ridistribuire, in modo ottimale, la capacità di trasmissione del sistema sullo spettro disponibile.
Nello standard ADSL, la modulazione DMT porta ad un'eccezionale resistenza ai disturbi di qualunque tipo, conservando, tuttavia, un'ottima efficienza spettrale.

quali modulazioni codificate multidimensionali adattative e C-OFDM (Coded-OFDM).

5. CONCLUSIONI

Il concetto introdotto da Shannon di limite di velocità massima per un canale di trasmissione - che successivamente ha assunto la denominazione di "limite di Shannon" - e la relativa formulazione matematica non sono più da considerarsi una pura speculazione teorica (come per molti anni si è ritenuto) ma hanno rappresentato un faro di riferimento nello sviluppo ottimale dei sistemi di trasmissione. In particolare, dopo aver posto le basi del problema e accennato allo sviluppo teorico dei codici a controllo di errore, in questo articolo ci si è posti l'obiettivo di illustrare, esaminando una serie di sistemi di comunicazione reali, come gli apparati industriali si siano progressivamente avvicinati al limite teorico, ottenendo oggi efficienze prossime al valore predetto da Shannon fin dal 1948.
Oltre che una riprova dell'importanza della teoria nello sviluppo dei sistemi, è giusto evidenziare il ruolo fondamentale giocato dalla componentistica microelettronica nel realizzare economicamente codici di alta complessità e quello che, sempre di più, sta giocando il software nella implementazione di vari codici. La metodologia software ha permesso - senza effettuare modifiche hardware e superando i problemi di compatibilità tra ricevitore e trasmettitore - di migliorare le prestazioni di molti impianti durante la loro vita commerciale o di quei sistemi che, per loro natura, non avrebbero potuto consentire in alcun modo modifiche hardware (per esempio, sonde spaziali).

Bibliografia

[1] Biglieri E.: Digital transmission in the 21st century: Conflating modulation and coding. *IEEE Communications Magazine*, 50th-Anniversary Issue, May 2002, p. 128--137.

[2] Verdú S., McLaughlin S.W.: Information Theory: 50 Years of Discovery. *IEEE Press*, 2000.

Ezio Biglieri si è laureato in Ingegneria Elettronica al Politecnico di Torino. Dal 1975 è Professore Ordinario, attualmente presso il Politecnico di Torino, in precedenza all'Università di Napoli e all'Università della California di Los Angeles (UCLA). Ha tenuto corsi in numerose Università estere, tra cui l'Università di Princeton (USA), l'Università di Sydney (Australia), il Politecnico di Poznàn (Polonia), la Pontificia Universidade do Rio (Brasile), l'Ecole Nationale Supérieure des Télécommunications (Parigi) e l'Università Nazionale di Yokohama (Giappone). Ha pubblicato circa 300 lavori scientifici e 5 libri dedicati alla teoria dei sistemi di telecomunicazione. Nel 1999, è stato presidente della "IEEE Information Theory Society". È "Fellow" dell'Institute of Electrical and Electronics Engineers (IEEE). Di recente, ha ricevuto il "IEEE Donald G. Fink Prize Paper Award" (2000), la medaglia IEEE del Terzo Millennio per contributi alla Teoria dell'Informazione (2000), e il premio alla carriera "IEEE Edwin Howard Armstrong Achievement Award" (2001).
Biglieri@polito.it

Guido Vannucchi, laureato in Ingegneria Industriale all'Università di Bologna nel 1958, "Master Science" in "Electrical Engineering" alla Stanford University nel 1963, Libera Docenza in Comunicazioni Elettriche nel 1971.
Dal 1960 in Telettra SpA (oggi Alcatel Italia), come Direttore Generale dal 1983 al 1990.
"Senior Consultant" prima di Italtel e poi di Olivetti Telemedia nonché coordinatore del progetto MxM ("Milano per la Multimedialità"). Vice Direttore Generale della RAI dal 1996 al 1998.
Docente al Politecnico di Milano di "Sistemi e Tecnologie della Comunicazione".
Laurea "ad honorem" in Ingegneria delle Telecomunicazioni, conferita dall'Università di Padova nel 1998 per i contributi scientifici e manageriali apportati al campo della trasmissione dei segnali e per gli studi e le realizzazioni pionieristiche nel campo della televisione digitale.
redazione@mondodigitale.net