TEORIA DELL'INFORMAZIONE E CODICI

Sandro Bellini

Politecnico di Milano

Prefazione

Queste brevi note sono state scritte per gli studenti del corso di Teoria dell'informazione e codici da me tenuto presso la Facoltà di Ingegneria del Politecnico di Milano. Parte del materiale qui presentato si trova su testi classici, ma non mancano modi di vedere più moderni reperibili solo in articoli recenti. Lo scopo principale è di fornire una sintesi, senza che si debba estrarre l'informazione da più fonti e, inevitabilmente, con notazioni diverse.

Il primo capitolo è dedicato alla teoria dell'informazione, creata da Shannon e resa pubblica nel 1948. Sono riassunti i più semplici, ma anche più importanti, risultati sulla codifica di sorgente e di canale. Si può affermare che senza questi riferimenti le applicazioni e gli sviluppi pratici nei decenni seguenti e fino ad oggi sarebbero stati molto più lenti.

Si assume che il lettore abbia già una conoscenza adeguata della teoria delle probabilità, e sia stato almeno parzialmente esposto ai problemi della trasmissione numerica.

Il secondo capitolo presenta una introduzione molto semplice alla codifica a blocco, i cui primi passi sono riferibili agli stessi anni in cui nacque la teoria dell'informazione. Si mostra come molti dei codici di interesse pratico siano sinteticamente descritti da un polinomio generatore, senza tuttavia entrare nel progetto del codice (salvo mostrare, come già Shannon aveva affermato, che se non vi fossero problemi di complessità il caso sarebbe un buon alleato nella costruzione di codici efficienti).

I codici convoluzionali, che hanno dominato la scena per molti anni, sono presentati nel terzo capitolo. Non occorrono particolari strumenti matematici per capire, o addirittura progettare, codici convoluzionali. Il capitolo è quindi abbastanza agevole, come il precedente.

Nel quarto capitolo sono riferiti gli elementi fondamentali della teoria dei campi finiti, cioè della base matematica dei codici a blocco più noti e più frequentemente usati. La stessa matematica è indispensabile per comprendere la decodifica algebrica di tali codici.

Il capitolo successivo dà gli elementi per la costruzione dei codici *BCH* e *Reed-Solomon* e per la loro decodifica. Oggi le occasioni in cui si scelgono tali codici non sono più frequenti come in passato, benché soprattutto i *Reed-Solomon* si difendano ancora bene. La scarsa adattabilità delle tecniche algebriche alla decodifica *soft* fa preferire altri codici quando tale informazione è disponibile all'uscita del canale di trasmissione. Tuttavia la matematica dei campi finiti è talmente interessante e affascinante che vien voglia di augurare ancora lunga vita alle sue applicazioni (fra cui oggi occorre citare alcune tecniche di crittografia).

Il sesto capitolo è dedicato ad approfondimenti sui codici a blocco, e in particolare a due importanti applicazioni della trasformata di *Hadamard*: le relazioni tra pesi di un codice e del suo duale, ed il calcolo delle probabilità a posteriori dei bit d'informazione basato sulle parole del codice duale.

Il capitolo successivo introduce i turbo codici, che hanno rivoluzionato lo stato dell'arte della codifica negli ultimi anni (sempre che siano tollerabili i ritardi nella trasmissione prodotti da blocchi di grande dimensione). Cominciano ad essere disponibili alcuni strumenti

per la previsione delle prestazioni dei turbo codici, che vengono sinteticamente presentati.

Nell'ultimo capitolo si affronta il problema dei codici adatti alle costellazioni multilivello, e si presenta il punto di vista più moderno sull'argomento. Si mostra come i codici binari (e quindi in particolare i turbo codici) possano essere utilizzati anche con tali costellazioni. Tuttavia si accenna anche alla tecnica che ha dominato tali applicazioni per vent'anni, anche se ora sembrerebbe destinata ad un rapido declino perlomeno nelle applicazioni che possono sopportare i considerevoli ritardi dei turbo codici.

Questo testo non è pensato per una rapida consultazione, ma piuttosto per un apprendimento più sistematico dei fondamenti della teoria dell'informazione e dei codici, mettendo in evidenza come i maggiori progressi nelle tecniche di codifica abbiano sempre avuto come guida e ispirazione la teoria dell'informazione.

E' sempre molto difficile resistere alla tentazione di includere qua e là divagazioni e approfondimenti, forse non strettamente indispensabili ma che tuttavia possono incuriosire qualche lettore. In genere si cede all'eleganza formale di un risultato, tendendo a trascurare il fatto che non tutti gli utilizzatori ne trarranno lo stesso piacere. Durante le lezioni si torna rapidamente alla realtà.

Allo stesso modo la tentazione in cui non deve cadere un docente è tramandare tutto quanto ha studiato, indipendentemente dalla reale utilità. Occorre mettere nella giusta prospettiva il vasto materiale disponibile, e in definitiva cancellare ampie porzioni del passato per dare spazio al nuovo.

E' bene avvertire il lettore che per semplificare la notazione sono frequentemente sottintesi gli estremi di integrali e somme, qualora siano infiniti oppure chiaramente deducibili dal contesto.

Ringrazio l'ing. Marco Ferrari, con cui collaboro da tempo nelle attività istituzionali di ricerca e didattica, in particolare (ma non solo) per le figure del capitolo sui turbo codici e per i commenti su tutto il testo.

Le imprecisioni e gli errori sono inevitabili, e sono responsabilità solo dell'autore. Spero che non siano troppo numerosi e che risulti facile intuire cosa avrei voluto, e dovuto, scrivere.

Come tutti i miei lavori dedico anche questo a Ilia, mia moglie.

Sandro Bellini

Indice

Capitolo 1

Teoria dell'informazione

1.1 Introduzione alla teoria dell'informazione

Le principali domande a cui dà risposta la teoria dell'informazione sono le seguenti:

- data una sorgente che emette messaggi da inviare ad un destinatario, quale sia il modo più economico per rappresentare, con qualità prefissata, l'informazione da trasmettere o da memorizzare su un qualche supporto fisico
- dato un canale non ideale, la cui uscita sia una replica distorta (rumorosa) dell'ingresso, come si possa trasmettere o memorizzare l'informazione in modo affidabile

Per entrambi i tipi di questioni si hanno almeno due aspetti da considerare: uno relativo alle tecniche utilizzabili in pratica per codificare una sorgente e per trasmettere l'informazione; l'altro, più teorico, su quali siano le migliori prestazioni ottenibili con vincoli prefissati sulla complessità dell'elaborazione: il caso più semplice, e più spesso considerato, è addirittura *senza vincoli di complessità*.

La teoria dell'informazione si concentra sui limiti teorici alle prestazioni della codifica di sorgente e della trasmissione in presenza di disturbi. Tuttavia le indicazioni che la teoria fornisce non sono solo uno stimolo a ricercare metodi pratici che si avvicinino alle migliori prestazioni possibili, ma spesso danno anche utili suggerimenti per orientarsi nella ricerca.

I problemi più semplici affrontati dalla teoria dell'informazione relativamente alla codifica di sorgente riguardano la codifica *senza perdita di informazione*, ovvero invertibile senza alcuna degradazione. Quando ad esempio si comprime un *file* di calcolatore si pretende di riottenerlo senza alcun bit sbagliato. Quando invece si comprime un'immagine o un segnale vocale solitamente si tollera una qualche degradazione, purché questa sia nota *a priori*. Evidentemente la disponibilità a tollerare imprecisioni nella ricostruzione è anche fortemente condizionata dall'entità del risparmio così ottenibile nella codifica di sorgente.

Analogamente quando si trasmette informazione, soprattutto se è stata effettuata una codifica di sorgente e quindi ogni simbolo ha ormai ben poca ridondanza, si pretende una

probabilità d'errore molto piccola: se si potesse, la si vorrebbe addirittura nulla. Tuttavia la teoria si occupa anche di situazioni meno comuni ma più complesse, in cui ci si accontenti ad esempio di una probabilità d'errore prefissata (magari non piccola).

1.2 Codifica di sorgente

Il primo problema della teoria dell'informazione è come si possa definire, e quindi misurare, la *quantità di informazione* emessa da una sorgente, detta anche *entropia* della sorgente. Si desidera che questa grandezza sia strettamente legata al *costo* minimo per la rappresentazione della sequenza di messaggi emessi dalla sorgente.

Occorre subito dire che ogni aspetto *semantico* viene totalmente ignorato dalla teoria dell'informazione formulata da *Shannon* verso la fine degli anni '40. Si immagini di dover trasmettere la sequenza di teste e croci ottenute nel lancio di una moneta non truccata, mediante una successione binaria di zeri e uni. L'informazione emessa da questa sorgente è raramente di un qualche valore per un possibile destinatario. Tuttavia si vedrà presto che fra le sorgenti binarie questa ha il *massimo* di informazione: *se* davvero occorre far conoscere il risultato di un lancio è inevitabile trasmettere un bit, essendo ogni lancio indipendente da tutti gli altri e non avendo la sorgente alcuna preferenza per le teste o per le croci, ovvero per zeri o uni. L'informazione, corrispondente ad un bit per lancio, non misura dunque l'utilità del messaggio ma il costo della rappresentazione con cifre binarie[1].

1.2.1 Sorgenti continue e discrete

Nel seguito si considerano solo sorgenti discrete, che emettono successioni di cifre, binarie o non binarie. Sorgenti continue *nel tempo* possono essere rese discrete nel tempo mediante il campionamento, con degradazione trascurabile se la frequenza di campionamento è scelta opportunamente. Sorgenti continue *nelle ampiezze* ottenute campionando sorgenti analogiche vengono solitamente discretizzate mediante la quantizzazione. Si tratta di una operazione che introduce un errore, di entità facilmente prevedibile[2]. Si assumerà quindi che la sorgente sia già discreta nel tempo e nelle ampiezze.

1.2.2 Sorgenti senza memoria e con memoria

Le proprietà statistiche della sorgente sono evidentemente importanti per la possibilità di codifica economica: si vedrà che occorre sfruttare sia la eventuale *non equiprobabilità*

[1] nel mondo si scambiano tanti messaggi apparentemente inutili, ma non è certo compito di chi progetta e gestisce i sistemi di trasmissione sindacare su cosa viene richiesto di trasmettere

[2] quella parte di teoria dell'informazione, qui non considerata, che si occupa della codifica di sorgente con perdita di informazione dà indicazioni sul modo migliore per discretizzare le ampiezze (che *non* è la usuale quantizzazione, ma una qualche forma di quantizzazione vettoriale) e sul numero minimo di bit per campione necessario per ottenere la precisione desiderata

dei messaggi sia la eventuale *non indipendenza* dei simboli successivamente emessi dalla sorgente (memoria della sorgente), assegnando stringhe codificate più lunghe (cioè più costose) ai messaggi o sequenze di messaggi meno frequenti.

Le sorgenti senza memoria sono più semplici da trattare teoricamente, e più facili da codificare: è un vero peccato che siano così rare in pratica! Per una descrizione completa di una sorgente senza memoria, che si suppone stazionaria, basta dare le probabilità dei possibili messaggi x_i, tratti da un insieme X, detto alfabeto.

Le sorgenti con memoria sono caratterizzate anche dalle probabilità congiunte, dei vari ordini, di messaggi successivi. Spesso questa conoscenza non è disponibile, o comunque non si hanno stime sufficientemente affidabili di tali probabilità. Ad esempio è evidente che volendo analizzare in un testo scritto le frequenze dei caratteri, delle coppie di caratteri, delle terne, ... il compito diventa via via più difficile: occorrono segmenti sempre più lunghi di testo, al punto da cominciare a dubitare che la sorgente possa essere ritenuta davvero stazionaria.

Spesso non sono note a priori neppure le probabilità dei singoli messaggi. Tuttavia anche in questo caso esistono tecniche di codifica di sorgente che danno prestazioni vicine ai limiti teorici, pur sconosciuti, *qualunque essi siano.*

1.2.3 Entropia di sorgenti senza memoria

Imponendo due semplici condizioni, intuitivamente desiderabili, ovvero che l'informazione portata da un messaggio sia tanto maggiore quanto meno il messaggio è atteso (cioè probabile) e che l'informazione di una coppia di messaggi indipendenti sia la somma delle rispettive quantità di informazione, *Shannon* riconobbe che si deve definire l'informazione di uno *specifico messaggio* x_i avente probabilità $P(x_i)$ come la quantità, non negativa,

$$I(x_i) = \log \frac{1}{P(x_i)} \tag{1.1}$$

Volendo definire l'informazione, o entropia, *della sorgente* si considera il valor medio, su tutto l'alfabeto, di $I(x_i)$:

$$H(X) = E\left[I(x_i)\right] = \sum_i P(x_i) \log \frac{1}{P(x_i)} \tag{1.2}$$

Si ha evidentemente $H(X) \geq 0$. La base dei logaritmi è arbitraria, ma solo due scelte sono di uso comune: i logaritmi naturali, nobili e privilegiati dal punto di vista matematico, oppure i logaritmi in base 2. Nel secondo caso il risultato del lancio di una moneta onesta, rappresentato con zero oppure uno, dà $I(0) = I(1) = 1$ e quindi $H(X) = 1$: chiaramente un *bit* d'informazione per ciascun lancio (messaggio). Se invece si usassero i logaritmi naturali si avrebbe $H(X) = 0.693$ *nat* per messaggio. Nel seguito i logaritmi saranno in base 2, a meno che sia diversamente specificato.

Con quattro messaggi x_i equiprobabili si ottiene $H(X) = 2$ bit per messaggio. Il risultato è confortante se si pensa che quattro messaggi equiprobabili possono essere ottenuti mediante *coppie* di bit indipendenti ed equiprobabili.

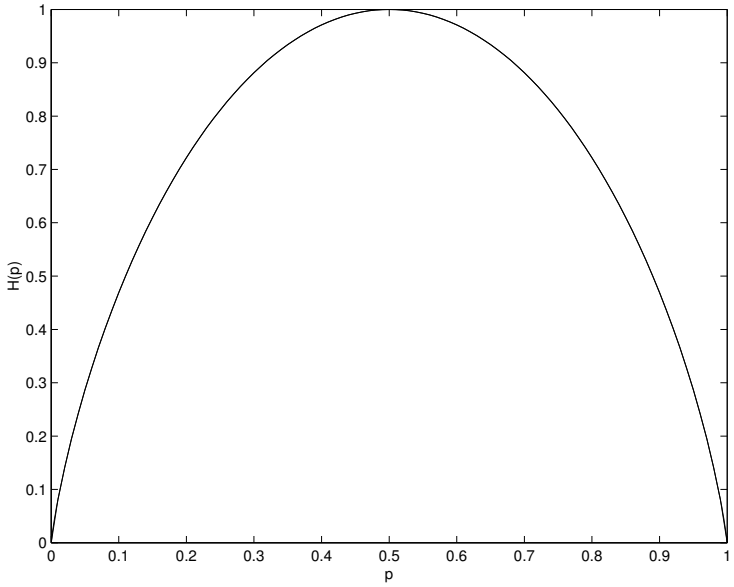

Figura 1.1: Entropia di una sorgente binaria, con probabilità p e $1-p$

Una moneta truccata (ma senza memoria) che dia teste e croci non equiprobabili, con probabilità p e $1-p$, ha entropia (mostrata in Fig. 1.1)

$$H(X) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p} \tag{1.3}$$

funzione che viene indicata anche con $H_2(p)$. Può essere utile osservare che

$$\frac{dH_2(p)}{dp} = \log \frac{1-p}{p} \tag{1.4}$$

da cui si riconosce che agli estremi $p = 0$ e $p = 1$ la derivata è infinita.

Il massimo dell'entropia di sorgente si ha quando gli M simboli dell'alfabeto X sono equiprobabili. Per dimostrarlo basta utilizzare la semplice diseguaglianza $\log_e x \leq x - 1$, valida per ogni $x > 0$, e quindi $\log x \leq \log e \, (x-1)$. Si ha uguaglianza solo per $x = 1$. Si ha

$$\begin{aligned} H(X) &= \sum_i P(x_i) \log \frac{1}{P(x_i)} = \log M + \sum_i P(x_i) \log \frac{1}{MP(x_i)} \\ &\leq \log M + \log e \sum_i P(x_i) \left(\frac{1}{MP(x_i)} - 1 \right) = \log M \end{aligned} \tag{1.5}$$

con uguaglianza solo se $P(x_i) = 1/M$ per ogni i.

Una dimostrazione ancora più semplice, se si conosce la tecnica dei moltiplicatori di Lagrange, è la seguente: i valori $P(x_i)$ sono soggetti al vincolo $\sum P(x_i) = 1$. Il massimo dell'entropia si ottiene imponendo che siano nulle le derivate di

$$\sum_i P(x_i) \log \frac{1}{P(x_i)} + \lambda \sum_i P(x_i)$$

ovvero

$$\log \frac{1}{P(x_i)} - \log e + \lambda = 0$$

da cui si ottiene che i valori $P(x_i)$ sono tutti uguali, e quindi pari all'inverso del numero di possibili messaggi.

1.2.4 Codifica di sorgenti senza memoria

Definita l'entropia di una sorgente senza memoria, si può dimostrare che il numero medio di bit strettamente necessari per codificare la sorgente coincide con l'entropia: non esistono codici con lunghezza media inferiore. Esistono tuttavia codici con lunghezza media vicina quanto si vuole all'entropia, purché non si ponga alcun vincolo sulla complessità del codice.

La lunghezza media di un codice è definita come il valor medio del numero di bit n_i spesi per codificare il generico messaggio x_i:

$$\bar{n} = \sum_i P(x_i) n_i \tag{1.6}$$

Si dimostra abbastanza facilmente che esiste almeno un codice *decodificabile* se vale la *diseguaglianza di Kraft*[3]

$$\sum_i 2^{-n_i} \leq 1 \tag{1.7}$$

La stessa condizione è necessaria per qualsiasi codice decodificabile, anche non istantaneamente[4]. Infatti per un alfabeto X di M simboli e per N intero qualsiasi si ha[5]

$$\left(\sum_{i=1}^{M} 2^{-n_i} \right)^N = \sum_{i_1} \dots \sum_{i_N} 2^{-(n_{i_1} + \dots + n_{i_N})} = \sum_{n=Nn_{\min}}^{Nn_{\max}} A_n 2^{-n} \leq N n_{\max} \tag{1.8}$$

[3]basta ordinare gli n_i per lunghezze crescenti, scegliere un blocco arbitrario di n_1 bit come primo codice, depennare tutte le parole di codice che inizino con queste cifre binarie (se ne scarta la frazione 2^{-n_1}), e procedere in modo analogo per le successive parole; se la diseguaglianza è verificata il procedimento può essere condotto fino in fondo; il codice ha la gradevole proprietà che nessuna parola è uguale all'inizio di un'altra, e quindi è decodificabile *istantaneamente* ovvero appena la generica parola è terminata

[4]ciò toglie ogni interesse ai codici non istantaneamente decodificabili, poiché non offrono alcun vantaggio

[5]la codifica di sorgente potrebbe non essere effettuata *simbolo per simbolo*; prima di codificare si potrebbero raggruppare in un unico simbolo più simboli emessi dalla sorgente; in tal caso l'alfabeto X è quello effettivo a cui viene applicata la codifica di sorgente

dove $n = n_{i_1} + \ldots + n_{i_N}$ è la lunghezza complessiva delle parole con cui sono codificati N messaggi consecutivi x_{i_1}, …, x_{i_N} emessi della sorgente, A_n è il numero di messaggi distinti codificati con n bit, e $n_{\max} = \max(n_1, \ldots, n_N)$. Si è anche utilizzato il fatto che perché il codice possa essere decodificabile deve essere $A_n \leq 2^n$: esistono infatti solo 2^n sequenze distinte di n bit. Poiché N è arbitrario, lasciando tendere N all'infinito si ottiene

$$\sum_{i=1}^{M} 2^{-n_i} \leq (N n_{\max})^{1/N} \to 1 \tag{1.9}$$

Dalla necessità della diseguaglianza di *Kraft* si ricava facilmente che la lunghezza media del codice non può essere minore dell'entropia della sorgente. Infatti

$$\begin{aligned} H(X) - \bar{n} &= \sum_i P(x_i) \left(\log \frac{1}{P(x_i)} - n_i \right) = \sum_i P(x_i) \log \frac{2^{-n_i}}{P(x_i)} \\ &\leq \log e \sum_i P(x_i) \left(\frac{2^{-n_i}}{P(x_i)} - 1 \right) \leq 0 \end{aligned} \tag{1.10}$$

Si noti che il risultato vale anche se si codificano *blocchi* di messaggi elementari: il numero medio di bit per codificare la N-pla non può essere minore dell'entropia della N-pla, data da $NH(X)$: si spendono quindi almeno $H(X)$ bit per ciascun messaggio.

Si noti che se per caso le probabilità $P(x_i)$ sono tutte potenze (negative) di 2 si può avere senza fatica $\bar{n} = H(X)$: basta scegliere $n_i = \log \frac{1}{P(x_i)}$.

In generale si può scegliere per ciascun n_i l'intero immediatamente superiore $\lceil \log \frac{1}{P(x_i)} \rceil$. Si dimostra facilmente che in questo modo si ottiene

$$H(X) \leq \bar{n} < H(X) + 1 \tag{1.11}$$

cioè che si *spreca* al più un bit. Se si codificano N-ple di messaggi elementari si spreca, nel caso peggiore, ancora non più di un bit e quindi in media non più di $1/N$ bit per messaggio. Scegliendo N sufficientemente grande il gioco è fatto.

L'assegnazione delle lunghezze delle parole $n_i = \lceil \log \frac{1}{P(x_i)} \rceil$ non è ottima: il *codice di Huffman* indica la procedura per rendere minimo $\bar{n}$, per un insieme prefissato di messaggi (elementari o non) di cui siano note le probabilità. Non si approfondisce l'argomento sia perché le sorgenti senza memoria sono rare in pratica sia perché esistono tecniche di codifica *universale* che non richiedono di conoscere le probabilità $P(x_i)$ dei messaggi.

Alcune conseguenze non trascurabili di una efficace codifica di sorgente sono le seguenti:

- dalla sequenza binaria codificata, e inviata al sistema di trasmissione, è stata rimossa quasi ogni ridondanza e non è possibile comprimere ulteriormente i messaggi; quindi la sequenza codificata contiene zeri e uni pressoché equiprobabili
- errori nella trasmissione dei bit codificati sono più gravi di errori commessi trasmettendo una sorgente fortemente ridondante (come gli errori di stampa in un testo

scritto, fastidiosi ma spesso innocui); se si è ricercata la massima compressione della sorgente il sistema di trasmissione deve essere molto affidabile

Codifica *universale* per sorgenti senza memoria

Se quattro messaggi hanno probabilità $p_1 = 1/2$, $p_2 = 1/4$ e $p_3 = p_4 = 1/8$ un codice adatto, e istantaneamente decodificabile[6], è 0, 10, 110, 111. Questo però sarebbe un pessimo codice se le probabilità fossero molte diverse da quelle attese. E se addirittura le probabilità dei quattro messaggi non fossero note *a priori* come si potrebbe codificare la sorgente? Sorprendentemente esistono tecniche di codifica *universale* che *non* richiedono la conoscenza delle probabilità e che danno prestazioni prossime all'entropia della sorgente (sconosciuta!) purché si codifichino N-ple sufficientemente lunghe.

Si consideri come esempio una sequenza di lanci di moneta truccata (testa e croce non equiprobabili). Indichiamo con 1 e 0 i possibili messaggi, e con p e $1-p$ le relative probabilità.

Il singolo messaggio è casuale, cioè impredicibile. In un blocco di N messaggi, con N grande, la *sequenza* è ugualmente imprevedibile, ma il numero complessivo di uni e zeri è quasi prevedibile, per la legge dei grandi numeri: circa Np uni e $N(1-p)$ zeri. Per N grande lo scarto quadratico medio $\sqrt{Np(1-p)}$, cioè l'incertezza sul numero di uni, è trascurabile rispetto al valore medio.

Un semplice metodo di codifica universale per sorgenti binarie senza memoria è il seguente. Presa una N-pla si contano gli uni e se ne codifica il numero con $\lceil \log(N+1) \rceil$ bit. Il fatto notevole è che il costo per messaggio $\lceil \log(N+1) \rceil / N$ è trascurabile se N è grande e tende a zero per N tendente all'infinito.

Noto il numero k di uni, questi possono presentarsi in $\binom{N}{k}$ modi che richiedono $\lceil \log \binom{N}{k} \rceil$ bit per la codifica.

Il costo addizionale per messaggio è prossimo a $H(X)$, come si può mostrare con qualche calcolo. L'ingrediente principale è la semplice approssimazione di *Stirling*

$$n! \approx \sqrt{2\pi n}\,(n/e)^n \tag{1.12}$$

da cui si ottiene

$$\begin{aligned} \binom{N}{k} = \frac{N!}{k!(N-k)!} &\approx \frac{\sqrt{2\pi N}(N/e)^N}{\sqrt{2\pi k}(k/e)^k \sqrt{2\pi(N-k)}((N-k)/e)^{N-k}} \approx \\ \approx \frac{N^N}{k^k (N-k)^{N-k}} &\approx \frac{N^N}{(Np)^{Np}(N(1-p))^{N(1-p)}} = p^{-Np}(1-p)^{-N(1-p)} \end{aligned} \tag{1.13}$$

Negli ultimi passaggi si sono trascurati i termini non esponenziali in N e si è utilizzato il

[6] si provi a scrivere una qualunque sequenza di zeri e uni e ad interpretarla come sequenza di messaggi

fatto che $k \approx Np$. Il costo per ciascun messaggio è quindi

$$\frac{1}{N} \log \binom{N}{k} \approx -p \log p - (1-p) \log(1-p) = H(X) \tag{1.14}$$

Si deve osservare che tale costo varia automaticamente con p! Se $p = 1/2$, $\binom{N}{k}$ è il massimo possibile. Per sorgenti fortemente sbilanciate $\binom{N}{k}$ è invece molto più piccolo.

Si consideri il seguente esempio numerico:

$$p = 0.1 \quad N = 100 \quad Np = 10 \quad \sqrt{Np(1-p)} = 3$$

Poiché k può avere 101 valori diversi, per codificare il valore di k bastano 7 bit. Poi si ha

$$\binom{N}{k} \approx \binom{N}{Np} = \binom{100}{10} = 1.73 \cdot 10^{13} \Rightarrow 44 \text{ bit}$$

Si osservi che potrebbe capitare, ad esempio, $k = 13$; in tal caso

$$\binom{N}{k} = \binom{100}{13} = 7.11 \cdot 10^{15} \Rightarrow 53 \text{ bit}$$

ma con circa la stessa probabilità si ha $k = 7$, e bastano 34 bit; il numero *medio* di bit spesi resta quasi invariato.

Il codice per il caso $k = 10$ contiene più di 10^{13} parole. Inoltre occorre prevedere codici diversi per ciascun valore di k. La complessità è quindi molto elevata.

In totale si spendono in media circa 51 bit per 100 messaggi e quindi 0.51 bit/messaggio[7]. Il lettore può facilmente verificare che con blocchi di soli 15 bit si sarebbe ottenuto un costo medio di 0.60 bit/messaggio, mentre con blocchi di 1000 bit basterebbero 0.475 bit/messaggio, valore molto vicino al limite teorico dato dall'entropia della sorgente

$$H_2(p) = 0.469 \text{ bit/messaggio} \tag{1.15}$$

La non equiprobabilità riduce quindi il costo della codifica. Se $p \approx 1/2$ è invece inutile tentare di comprimere la sorgente: ogni messaggio costa un bit.

1.2.5 Entropia condizionata

Se si considerano le probabilità $P(x_i/y_j)$ dei messaggi x_i emessi dalla sorgente, condizionate ad uno *specifico* evento y_j si potrebbe definire l'entropia condizionata all'evento y_j

$$H(X/y_j) = \sum_i P(x_i/y_j) \log \frac{1}{P(x_i/y_j)} \tag{1.16}$$

[7] il calcolo esatto dà 0.507 bit/messaggio

In genere y_j è a sua volta tratto da un alfabeto Y, con probabilità $P(y_j)$. Risulta quindi più utile definire l'*entropia condizionata* come valor medio rispetto agli eventi y_j, dato da

$$H(X/Y) = \sum_j P(y_j) \sum_i P(x_i/y_j) \log \frac{1}{P(x_i/y_j)} = \sum_i \sum_j P(x_i, y_j) \log \frac{1}{P(x_i/y_j)} \quad (1.17)$$

Si verifica facilmente che il condizionamento non può aumentare l'entropia. Infatti si ha

$$\begin{aligned} H(X/Y) - H(X) &= \sum_i \sum_j P(x_i, y_j) \log \frac{P(x_i)}{P(x_i/y_j)} \\ &\leq \log e \sum_i \sum_j P(x_i, y_j) \left(\frac{P(x_i)}{P(x_i/y_j)} - 1 \right) = 0 \end{aligned} \quad (1.18)$$

con uguaglianza se e solo se $P(x_i/y_j) = P(x_i)$ per ogni i e j, cioè se i messaggi x_i sono statisticamente indipendenti dagli eventi y_j.

In modo del tutto analogo, se si introduce un ulteriore condizionamento si ottiene

$$H(X/Y, Z) \leq H(X/Y) \quad (1.19)$$

Entropia congiunta

Se si considerano *congiuntamente* coppie di messaggi x_i e y_j, aventi probabilità $P(x_i, y_j)$ si può evidentemente definire l'entropia congiunta

$$H(X, Y) = \sum_i \sum_j P(x_i, y_j) \log \frac{1}{P(x_i, y_j)} \quad (1.20)$$

Dalle proprietà delle probabilità congiunte e condizionate risulta evidente che

$$H(X, Y) = H(X) + H(Y/X) = H(Y) + H(X/Y) \quad (1.21)$$

Più in generale valgono relazioni come

$$H(X, Y, Z, \ldots) = H(X) + H(Y/X) + H(Z/X, Y) + \ldots \quad (1.22)$$

Entropia di una funzione di variabile casuale

Sia $H(X)$ l'entropia dell'alfabeto $x_i \in X$ e sia $y = g(x)$. Si ha $H(Y) \leq H(X)$. Infatti se è noto x è noto anche y e quindi

$$H(X, Y) = H(X) + H(Y/X) = H(X) \quad (1.23)$$

mentre noto y può non essere noto x (se la funzione non è univocamente invertibile) e quindi

$$H(X, Y) = H(Y) + H(X/Y) \geq H(Y) \quad (1.24)$$

Si osservi che se la funzione $y = g(x)$ è invertibile $H(Y) = H(X)$. I valori x_i e y_i non contano nulla. L'entropia è determinata solo dall'insieme delle probabilità dei messaggi.

1.2.6 Entropia di sorgenti con memoria

Sia $\{x_k\}$ la successione di messaggi emessi da una sorgente stazionaria con memoria. Se la sorgente non avesse memoria l'entropia $H(X_k)$ verrebbe calcolata dalle probabilità marginali dei messaggi. Tuttavia a causa della memoria il messaggio x_k all'istante k è meglio prevedibile di quanto indicato dalle sole probabilità marginali $P(x_k)$. Infatti è già noto il messaggio precedente x_{k-1}. L'entropia condizionata al messaggio precedente è infatti $H(X_k/X_{k-1}) \leq H(X_k)$. Se la memoria della sorgente si estende ad altri messaggi precedenti si considerano $H(X_k/X_{k-1}, X_{k-2}) \leq H(X_k/X_{k-1}), \ldots$

L'entropia della sorgente $H(X)$ è definita come limite di tale successione non crescente di entropie condizionate, per osservazione tendente all'infinito. Il calcolo dell'entropia risulta semplice nel caso di sorgenti di *Markov* perché non occorre procedere oltre la lunghezza finita della memoria. In generale il calcolo non è semplice, e spesso non è praticamente possibile *misurare* tutte le probabilità condizionate che sono richieste.

Una definizione alternativa di entropia, equivalente alla precedente, è ottenuta considerando blocchi di L messaggi e facendo tendere L a infinito:

$$H_L(X) = \frac{1}{L} H(X_k, X_{k-1}, \ldots, X_{k-L+1}) \tag{1.25}$$

$$H(X) = \lim_{L \to \infty} H_L(X) \tag{1.26}$$

Poiché il condizionamento non aumenta l'entropia e la sorgente è stazionaria si ha

$$\begin{aligned} H_L(X) = \frac{1}{L}\big(&H(X_{k-L+1}) + H(X_{k-L+2}/X_{k-L+1}) + \ldots \\ &+ H(X_k/X_{k-1}, \ldots, X_{k-L+1})\big) \geq H(X_k/X_{k-1}, \ldots, X_{k-L+1}) \end{aligned} \tag{1.27}$$

ma è anche evidente che le due successioni tendono allo stesso limite.

Con questa seconda definizione di entropia è immediato dimostrare che anche nel caso di sorgenti con memoria la lunghezza media di qualsiasi codice non può essere minore dell'entropia, ma può avvicinarvisi quanto si vuole. La dimostrazione è infatti del tutto analoga a quella del caso senza memoria; basta considerare sequenze di messaggi di lunghezza L tendente a infinito.

Codifica di sorgenti con memoria

Se si considerano sorgenti con memoria, in generale una coppia di messaggi consecutivi (x_1, x_2) ha probabilità $P(x_1, x_2) \neq P(x_1)P(x_2)$.

Ad esempio sia $P(x_2 = x_1) = 0.8$ e $P(x_2 \neq x_1) = 0.2$, cioè la sorgente ripeta il messaggio precedente con probabilità 0.8. Si supponga anche per semplicità che la sorgente sia di Markov con memoria uno. E' facile verificare che $P(x_1 = 0) = P(x_1 = 1) = 0.5$ e

$P(x_2 = 0) = P(x_2 = 1) = 0.5$. Le probabilità delle coppie (x_1, x_2) sono date da

$$P(0,0) = 0.4$$
$$P(0,1) = 0.1$$
$$P(1,0) = 0.1$$
$$P(1,1) = 0.4$$

La sorgente emette zeri e uni equiprobabili, ma non indipendenti. Il costo minimo per la codifica delle coppie è

$$\frac{-0.4\log 0.4 - 0.1\log 0.1 - 0.1\log 0.1 - 0.4\log 0.4}{2} = 0.86 \text{ bit/messaggio} \tag{1.28}$$

Considerando terne, quaterne, ecc. di messaggi il risultato può solo migliorare. Il limite per lunghezza L del messaggio elementare tendente a infinito è il costo ineliminabile per la trasmissione. Si ponga attenzione a non si confondere L con N. Il suggerimento della teoria è che il codice debba trattare lunghi blocchi (lunghezza N) di messaggi, costituiti a loro volta da blocchi sufficientemente lunghi (lunghezza L) di messaggi elementari emessi dalla sorgente.

Se nell'esempio precedente si codificassero le terne ($L = 3$) ci si potrebbe avvicinare, con N sufficientemente elevato, a 0.815 bit/messaggio. Aumentando L si tenderebbe all'entropia, il cui valore è $H = 0.722$ bit/messaggio.

Anche per sorgenti con memoria esistono tecniche di codifica *universale*, che non richiedono la conoscenza a priori della statistica della sorgente. Attualmente le tecniche più diffuse sono, con qualche variante, quelle di *Ziv-Lempel*. Il principio è di formare via via, mentre si codifica la sorgente, un dizionario di dimensione crescente di blocchi di messaggi già incontrati e di relativi blocchi codificati. Procedendo nella codifica, alle parole d'informazione già incluse nel dizionario vengono appesi uno zero oppure un uno, secondo la sequenza emessa dalla sorgente. Il dizionario tende ad arricchirsi solo di sequenze tipiche (le più frequenti). Le sequenze mai incontrate, perché impossibili o molto rare, non hanno un codice.

La codifica si adatta automaticamente alle probabilità congiunte della sorgente: si può dimostrare che per sequenze di lunghezza N tendente all'infinito il costo tende all'entropia della sorgente. La dimostrazione è tuttavia molto lunga e complessa.

Per concludere, è evidente che una buona compressione di sorgente elimina quasi totalmente la dipendenza statistica tra messaggi successivi: la sequenza codificata appare, al trasmettitore, pressoché senza memoria.

1.3 Modelli del canale di trasmissione

Ogni canale di trasmissione continuo nel tempo può essere descritto da un equivalente canale discreto nel tempo. Ad esempio nella trasmissione numerica in banda base la cascata

di generazione della forma d'onda, invio sul mezzo trasmissivo, aggiunta dell'inevitabile rumore (termico, elettronico, ecc.), filtraggio in ricezione e campionamento in un opportuno istante di lettura produce in uscita la somma dell'ampiezza trasmessa e di un campione di rumore. Se i simboli trasmessi successivamente non interferiscono tra loro la trasmissione di una successione di simboli equivale a riusare altrettante volte questo semplice canale discreto.

Nella trasmissione numerica multilivello in banda passante risulta conveniente vedere le due componenti trasmesse e le due ricevute come coppie di numeri reali, oppure in modo equivalente come numeri complessi[8].

Per quanto riguarda le ampiezze, nella trasmissione numerica la successione dei simboli in ingresso è già discreta (nel caso più semplice a soli due valori). Ingresso e uscita sono discreti nel tempo, ma solo l'ingresso è discreto nelle ampiezze. Infatti a causa del rumore l'uscita assume valori continui nelle ampiezze. In alcuni casi l'uscita continua è inviata ad un decisore e quindi è disponibile solo un simbolo deciso, tratto da un alfabeto finito.

Limitandosi al caso di canali *senza memoria*, la descrizione statistica del canale di trasmissione richiede la conoscenza di

- alfabeto X in ingresso (ad esempio binario: $x_i = 0, 1$)
- alfabeto Y in uscita (ad esempio un numero reale y, o una coppia di numeri reali, oppure un bit deciso $y_j = 0, 1$)
- probabilità di transizione tra ingresso e uscita: densità di probabilità $p(y/x_i)$ nel caso continuo, o probabilità $P(y_j/x_i)$ nel caso discreto

Per valutare il comportamento del canale è anche utile conoscere

- probabilità $P(x_i)$ di ciascun ingresso (e se vi fosse memoria anche le probabilità congiunte di ingressi successivi)
- analoghe probabilità $P(y_j)$, o densità di probabilità $p(y)$, all'uscita; queste sono calcolabili dai dati precedenti

Alcuni semplici esempi di canali sono:

- canale gaussiano (canale AWGN: *Additive White Gaussian Noise*): l'uscita y è la somma dell'ingresso x e di una variabile casuale n gaussiana a valor medio nullo e con varianza σ_n^2; in pratica si ottiene un canale di questo tipo se si trasmette la forma d'onda $x\,s(t)$ dove $s(t)$ ha energia E_s, il mezzo trasmissivo aggiunge un

[8] con costellazioni QPSK e QAM, purché non vi sia interferenza tra i simboli, la trasmissione della coppia è del tutto equivalente a due successive trasmissioni in banda base, e quindi si ragiona come se si fosse in banda base; ciò non è più vero se le ampiezze trasmesse con le due componenti non sono indipendenti, come ad esempio con costellazioni 8PSK

rumore gaussiano bianco con densità spettrale bilatera $N_0/2$ e in ricezione si valuta $y = \int r(t)s(t)\,dt/E_s$; si può infatti verificare che $y = x + n$ dove $\sigma_n^2 = N_0/2E_s$; l'ingresso x è discreto o continuo, l'uscita y è continua

- canale binario simmetrico (BSC: *Binary Symmetric Channel*): alfabeto binario {0,1} (oppure ±1) sia in ingresso sia in uscita; probabilità di errore $P(y = 1/x = 0) = P(y = 0/x = 1) = \varepsilon$ indipendente dall'ingresso; è ad esempio un canale AWGN con ingresso binario, seguito da un decisore *hard* che rende binaria anche l'uscita

- canale binario con cancellazione (BEC: *Binary Erasure Channel*): alfabeto binario {0,1} in ingresso e ternario in uscita {0,1,E} dove E indica incertezza completa; $P(E/x = 0) = P(E/x = 1) = \varepsilon$; $P(y = 0/x = 0) = P(y = 1/x = 1) = 1 - \varepsilon$; si noti che questo semplice modello assume che gli zeri e uni ricevuti siano sempre corretti

1.4 Informazione mutua

Si considereranno dapprima i canali con ingresso e uscita discreta. Si potranno definire l'entropia sia dell'ingresso sia dell'uscita

$$H(X) = \sum_i P(x_i) \log \frac{1}{P(x_i)} \tag{1.29}$$

$$H(Y) = \sum_j P(y_j) \log \frac{1}{P(y_j)} \tag{1.30}$$

Se si considera ad esempio un canale binario simmetrico con ingressi equiprobabili, è immediato verificare che sono equiprobabili anche le uscite. Si ha quindi $H(X) = H(Y) = 1$. Questo risultato vale sia nel caso di canale ideale, che non commette errori, sia in caso di errori e persino per il peggior canale possibile con probabilità d'errore $\varepsilon = 1/2$. Nel primo caso $H(Y)$ è informazione effettivamente trasmessa, nell'ultimo $H(Y)$ è informazione del tutto inutile. Per distinguere tra questi casi occorre considerare anche l'entropia congiunta, ovvero l'entropia della coppia (x_i, y_j)

$$H(X,Y) = \sum_i \sum_j P(x_i, y_j) \log \frac{1}{P(x_i, y_j)} \tag{1.31}$$

oppure le entropie condizionate

$$H(Y/X) = \sum_i \sum_j P(x_i, y_j) \log \frac{1}{P(y_j/x_i)} \tag{1.32}$$

$$H(X/Y) = \sum_i \sum_j P(x_i, y_j) \log \frac{1}{P(x_i/y_j)} \tag{1.33}$$

Si osservi che $H(Y/X)$ e $H(X/Y)$ sono rispettivamente le quantità di informazione occorrenti per determinare l'uscita noto l'ingresso[9] e l'ingresso data l'uscita[10].

Per il canale ideale risulta $H(Y/X) = H(X/Y) = 0$ mentre per il canale pessimo, cioè del tutto inutile, si ha $H(Y/X) = H(Y)$ e $H(X/Y) = H(X)$. Si è già visto che in ogni caso

$$H(X,Y) = H(X) + H(Y/X) = H(Y) + H(X/Y) \tag{1.34}$$

Inoltre, poiché il condizionamento non può aumentare l'entropia, si ha

$$H(Y/X) \leq H(Y) \tag{1.35}$$

$$H(X/Y) \leq H(X) \tag{1.36}$$

Poiché $H(X)$ è l'informazione all'ingresso del canale e l'equivocazione $H(X/Y)$ ha il significato di informazione persa nel canale, l'informazione che attraversa il canale è data dalla differenza tra le due entropie

$$I(X,Y) = H(X) - H(X/Y) \geq 0 \tag{1.37}$$

ed è detta *informazione mutua*. Dalle proprietà già viste si ottiene anche

$$I(X,Y) = H(X) + H(Y) - H(X,Y) = H(Y) - H(Y/X) \tag{1.38}$$

Per ricordare facilmente le diverse espressioni equivalenti dell'informazione mutua è anche utile scrivere $I(X,Y)$ in una forma che mostra che si tratta di una misura dell'indipendenza statistica tra x e y:

$$I(X,Y) = E\left[\log \frac{P(x,y)}{P(x)P(y)}\right] = E\left[\log \frac{P(x/y)}{P(x)}\right] = E\left[\log \frac{P(y/x)}{P(y)}\right] \tag{1.39}$$

dove la media è da intendere rispetto alle coppie (x,y). Quindi $I(X,Y) = 0$ se x e y sono indipendenti.

Informazione mutua congiunta e condizionata

L'informazione mutua relativa ad alfabeti congiunti e l'informazione mutua condizionata sono definibili in modo analogo. Ad esempio si ha[11]

$$I(X,YZ) = E\left[\log \frac{P(x,y,z)}{P(x)P(y,z)}\right] = E\left[\log \frac{P(x,y)P(z/xy))}{P(x)P(y)P(z/y)}\right] = I(X,Y) + I(X,Z/Y)$$

[9] non uno *specifico* ingresso x_i, ma di volta in volta l'ingresso effettivo! infatti si calcola la media rispetto ai possibili ingressi x_i

[10] l'informazione ancora necessaria per specificare l'ingresso una volta nota l'uscita ha il significato di informazione persa nel passaggio attraverso il canale, ed è anche detta *equivocazione*

[11] invece di $I(X,Y)$ e $I(X,YZ)$ è molto usata anche la notazione $I(X;Y)$ e $I(X;Y,Z)$

(1.40)

e analogamente si ottiene

$$I(X, YZ) = I(X, Z) + I(X, Y/Z) \tag{1.41}$$

Se z è una funzione deterministica di y si ha $P(z/x, y) = P(z/y)$ e quindi

$$I(X, YZ) = I(X, Y) \tag{1.42}$$

ma anche

$$I(X, YZ) = I(X, Z) + I(X, Y/Z) \geq I(X, Z) \tag{1.43}$$

e quindi infine

$$I(X, Z) \leq I(X, Y) \tag{1.44}$$

Qualunque elaborazione venga fatta a partire da y per ottenere z l'informazione mutua non può aumentare. Ciò è vero, ad esempio, anche quando x è una parola di codice, y il blocco ricevuto (ancora da decodificare) e z la parola decodificata. L'informazione su x contenuta in z non è maggiore di quella contenuta in y, anche se è presentata in una forma molto più comoda per l'utilizzatore finale.

1.5 Capacità di canale

L'informazione mutua dipende non solo del canale ma anche dalla statistica della sorgente. Se infatti si trasmettesse una sorgente con entropia $H(X) = 0$ si vede dalla (1.37) che l'informazione mutua $I(X, Y)$ sarebbe nulla anche se il canale fosse ideale. Si definisce *capacità* del canale la massima informazione mutua possibile, ottenuta con la miglior distribuzione di probabilità dell'ingresso:

$$C = \max_{P(x)} I(X, Y) \tag{1.45}$$

Un esempio molto semplice è la capacità del canale binario simmetrico. Poiché[12]

$$H(Y/X) = H_2(\varepsilon) \tag{1.46}$$

e il massimo di $H(Y)$, ottenuto con ingressi equiprobabili e quindi uscite equiprobabili, è pari a 1 la capacità è data da

$$C = 1 - H_2(\varepsilon) \tag{1.47}$$

[12]evidentemente sia $H(Y/x = 0)$ sia $H(Y/x = 1)$, entropie dell'uscita condizionata agli *specifici* ingressi $x = 0$ e $x = 1$, valgono $H_2(\varepsilon)$; questo è quindi anche il valor medio rispetto all'ingresso x. Si noti che il risultato non dipende dalla distribuzione di x

Ad esempio se $\varepsilon = 0.1$ si ha $C = 0.53$: ogni bit uscente dal canale vale 0.53 bit d'informazione. Non deve sorprendere che quasi metà dell'informazione sia persa con il 10% degli errori; il problema per la trasmissione affidabile dell'informazione è *localizzare* gli errori.

Anche nel caso di canale con cancellazione $H(Y/X)$ è data dalla (1.46). Con semplici calcoli si verifica che il massimo di $H(Y)$ si ottiene con ingressi equiprobabili e infine che $C = 1 - \varepsilon$. Il risultato ha una interpretazione intuitiva: le uscite 0 e 1 sono sicure; l'uscita E, che si verifica con probabilità ε, è totalmente priva di informazione.

L'espressione $I(X,Y) = H(Y) - H(Y/X)$ è spesso la più comoda per il calcolo della capacità. In funzione delle probabilità $P(x_i)$ degli ingressi e delle probabilità $P(y_j/x_i)$ di transizione l'espressione della capacità è

$$C = \max_{P(x)} \sum_i \sum_j P(x_i) P(y_j/x_i) \log \frac{P(y_j/x_i)}{\sum_i P(y_j/x_i) P(x_i)} \tag{1.48}$$

dove per $P(y_j)$ si è usata l'espressione $\sum_i P(y_j/x_i) P(x_i)$.

1.5.1 Entropia, informazione mutua e capacità nel caso continuo

Volendo generalizzare al caso continuo[13] si può pensare di discretizzare le variabili continue in modo via via più fitto, con un procedimento di limite. Tuttavia l'entropia perde ogni significato fisico: le probabilità $P(y_j)$ diventano gli infinitesimi $p(y)\,dy$ e la somma diventerebbe un integrale, ma il termine dy nel logaritmo farebbe divergere il risultato. Dunque l'entropia cresce, e tende all'infinito. Tuttavia nelle espressioni che contengono la *differenza* di due entropie, come quelle dell'informazione mutua e della capacità di canale, solitamente il logaritmo del passo di discretizzazione si cancella. Ignorando dunque il passo di discretizzazione nei logaritmi, basta definire convenzionalmente

$$H(Y) = \int_y p(y) \log \frac{1}{p(y)}\, dy \tag{1.49}$$

e in modo analogo le altre quantità. L'entropia di una variabile continua y viene anche detta *entropia differenziale*, e talvolta indicata con $h(Y)$. In qualche caso limite occorre cautela nell'uso dell'entropia differenziale. Ad esempio in generale, a differenza del caso discreto, se $y = g(x)$ è una funzione invertibile $H(Y) \neq H(X)$. Inoltre nel calcolo dell'informazione mutua $I(X,Y) = H(Y) - H(Y/X)$ il passo di discretizzazione *non* si cancella se $p(y)$ è una *ddp* continua ma $p(y/x)$ è invece discreta[14].

[13] spesso l'uscita è continua ma l'ingresso è discreto

[14] come accadrebbe nel caso, di nessun interesse pratico, di x variabile casuale continua e y funzione deterministica di x; l'informazione mutua sarebbe infinita

Ignorando queste rare eccezioni, se x è una variabile discreta si ha

$$C = \max_{P(x)} \sum_i \int_y P(x_i) p(y/x_i) \log \frac{p(y/x_i)}{\sum_i p(y/x_i) P(x_i)} \, dy \tag{1.50}$$

Se anche x è continuo si ha

$$C = \max_{p(x)} \int_x \int_y p(x) p(y/x) \log \frac{p(y/x)}{\int_x p(y/x) p(x) \, dx} \, dx \, dy \tag{1.51}$$

Capacità nel caso di rumore additivo indipendente

Se il rumore è additivo e indipendente dal segnale l'uscita è $y = x + n$, con x e n variabili casuali indipendenti. Nel caso di ingresso discreto si verifica subito che l'entropia condizionata $H(Y/X)$ è data da

$$\sum_i P(x_i) \int_y p(y/x) \log \frac{1}{p(y/x)} \, dy = \sum_i P(x_i) \int_n p(n) \log \frac{1}{p(n)} \, dn = H(N) \tag{1.52}$$

ed è indipendente da x. Quindi

$$C = \max_{P(x)} H(Y) - H(N) \tag{1.53}$$

Nel caso continuo è facile verificare che il risultato è analogo:

$$C = \max_{p(x)} H(Y) - H(N) \tag{1.54}$$

Capacità del canale gaussiano con ingresso continuo

Per calcolare la capacità con l'espressione appena vista basta dunque cercare il massimo dell'entropia $H(Y)$. Se non si ponesse alcun vincolo sulla varianza di y si potrebbe aumentare $H(Y)$ e quindi la capacità senza limite. In molti casi di interesse pratico è fissata la varianza σ_x^2 e quindi anche $\sigma_y^2 = \sigma_x^2 + \sigma_n^2$.
Se $p(y)$ è la densità di probabilità di una generica variabile casuale y con valor medio m_y e varianza σ_y^2 e se $Z(y)$ è la densità gaussiana di pari valor medio e varianza, si ha

$$\begin{aligned} &\int_y p(y) \log \frac{1}{Z(y)} \, dy = \int_y p(y) \left(\log \sqrt{2\pi\sigma_y^2} + \frac{(y - m_y)^2}{2\sigma_y^2} \log e \right) \, dy = \\ &= \log \sqrt{2\pi\sigma_y^2} + \frac{1}{2} \log e = \frac{1}{2} \log(2\pi e \sigma_y^2) \end{aligned} \tag{1.55}$$

In particolare se si pone $p(y) = Z(y)$ si ottiene immediatamente che l'entropia di una variabile casuale gaussiana con varianza σ_y^2 è

$$H(Y) = \frac{1}{2}\log(2\pi e\sigma_y^2) \tag{1.56}$$

Per una generica variabile casuale y con densità $p(y)$ e varianza σ_y^2 si ha

$$H(Y) - \frac{1}{2}\log(2\pi e\sigma_y^2) = \int_y p(y)\log\frac{Z(y)}{p(y)}\,dy \le \log e\int_y p(y)\left(\frac{Z(y)}{p(y)} - 1\right)dy = 0 \tag{1.57}$$

con uguaglianza se e solo se $p(y) = Z(y)$, cioè nel caso gaussiano. Fissata la varianza σ_y^2, l'entropia $H(Y)$ è quindi massima nel caso di densità di probabilità gaussiana[15]. Poichè $y = x + n$, dove n ha densità gaussiana, anche x è una variabile casuale gaussiana. La densità $p(x)$ dell'ingresso che rende massima l'informazione mutua e permette di ottenere la capacità del canale è dunque gaussiana. Fissata σ_x^2 la capacità è data da

$$C = H(Y) - H(N) = \frac{1}{2}\log\left(2\pi e(\sigma_x^2 + \sigma_n^2)\right) - \frac{1}{2}\log(2\pi e\sigma_n^2) = \frac{1}{2}\log\left(1 + \frac{\sigma_x^2}{\sigma_n^2}\right) \tag{1.58}$$

Si può ottenere un canale discreto nel tempo trasmettendo $2B$ impulsi al secondo, modulati in ampiezza senza interferenza reciproca, in una banda B. In presenza di rumore gaussiano con densità spettrale di potenza bilatera $N_0/2$ si ha $\sigma_n^2 = N_0/2$. Moltiplicando σ_x^2 e σ_n^2 per $2B$ si ottengono rispettivamente la potenza P del segnale e la potenza N_0B del rumore nella banda B. Poiché il canale trasmette $2B$ messaggi al secondo si ha la notissima espressione della capacità, espressa in bit al secondo,

$$C = B\log\left(1 + \frac{P}{N_0B}\right) \tag{1.59}$$

Capacità del canale gaussiano con ingresso discreto

Se l'ingresso è binario antipodale ($x = \pm 1$) l'entropia $H(Y)$ dell'uscita deve essere calcolata numericamente. Si può verificare che il massimo si ha per ingressi equiprobabili. Si ottiene

$$C = \int_n p(n-1)\log\frac{2}{p(n-1) + p(n+1)}\,dn - \frac{1}{2}\log(2\pi e\sigma_n^2) \tag{1.60}$$

Nella trasmissione a più di due livelli il massimo di $H(Y)$ si ottiene con ingressi *non* equiprobabili. Il problema di maggior interesse, anche in questo caso, è la ricerca del massimo di $H(Y)$ e quindi dell'informazione mutua, a parità di *energia media* per simbolo

$$E_s = \sigma_x^2 = \sum x_i^2 P(x_i) \tag{1.61}$$

[15]il risultato è dimostrabile anche cercando il massimo rispetto a $p(y)$ di $-\int p(y)\log p(y)\,dy$ con i vincoli $\int p(y)\,dy = 1$ e $\int (y - m_y)^2 p(y)\,dy = \sigma_y^2$. Si ottiene infatti $-\log p(y) - \log e + \lambda + \mu(y - m_y)^2 = 0$ ovvero $p(y) \equiv \exp(-\mu(y - m_y)^2)$

Conviene utilizzare meno frequentemente i livelli più esterni, più costosi. Tuttavia i codici più comuni e più semplici producono livelli equiprobabili. Inoltre le probabilità $P(x_i)$ ottimizzate dipendono dal rapporto segnale-rumore. Può quindi essere di interesse effettuare il calcolo con livelli equiprobabili, pur sapendo di non raggiungere la capacità.

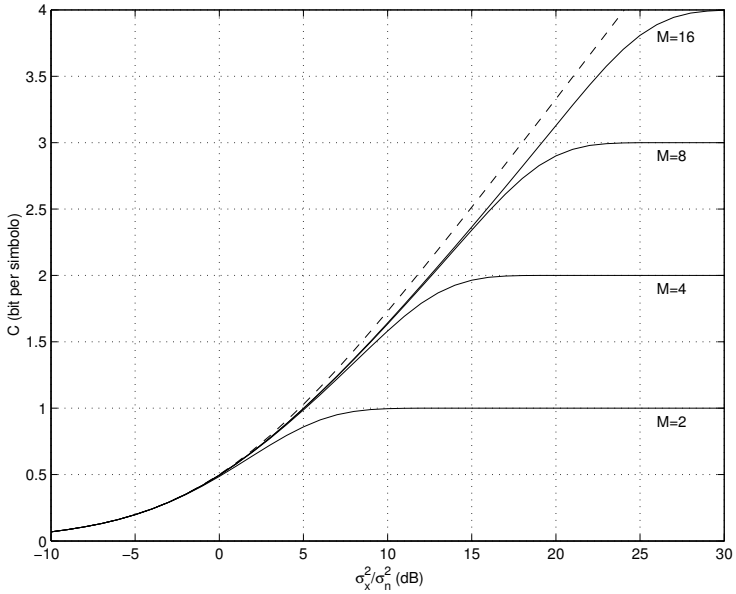

Figura 1.2: Capacità nella trasmissione a $M = 2$, 4, 8 e 16 livelli equiprobabili (curve continue) e con ingresso continuo gaussiano (curva tratteggiata)

In Fig. 1.2 è mostrato il risultato, con $M = 2$, 4, 8 e 16 livelli equiprobabili. Per capacità[16] fino a circa 0.5 bit per simbolo non occorrono più di 2 livelli, mentre ad esempio 4 livelli sono sufficienti fino a circa 1.5 bit per simbolo. Con M livelli la capacità non può superare $\log M$ bit per simbolo, per quanto grande sia il rapporto segnale-rumore.

Per confronto in Fig. 1.2 è mostrata anche la capacità nel caso di ingresso continuo con densità di probabilità gaussiana. Poiché, soprattutto ad alto rapporto segnale-rumore, si possono ottenere interessanti guadagni, nei sistemi di trasmissione più raffinati si usa un numero elevato di livelli e un codice che li utilizza con probabilità non uniformi.

In Fig. 1.3 è mostrata la capacità di alcune costellazioni bidimensionali PSK, QAM e CR[17], con punti equiprobabili. In ascissa σ_x^2 e σ_n^2 sono le varianze di segnale e rumore misurate entrambe in una sola dimensione (oppure entrambe nelle due dimensioni). Si noti che le costellazioni 4PSK, 16QAM e 64QAM hanno capacità doppia delle corrispondenti PAM a 2, 4 e 8 livelli. Sono infatti equivalenti a due usi del canale monodimensionale. Anche in Fig. 1.3 è mostrata la capacità nel caso di ingresso continuo con *ddp* gaussiana.

[16] a rigore è scorretto chiamare capacità i valori di informazione mutua ottenuti senza ottimizzare le probabilità dell'ingresso

[17] le costellazioni CR sono ottenute dalle QAM eliminando alcuni dei punti di maggiore energia: ad esempio la costellazione 32CR è una QAM 6x6 senza i quattro spigoli

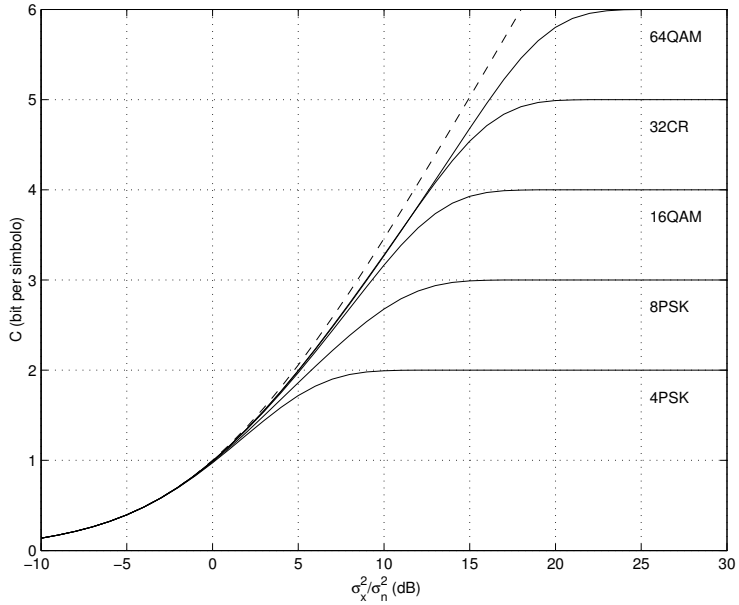

Figura 1.3: Capacità di costellazioni bidimensionali con punti equiprobabili (curve continue) e con ingresso continuo gaussiano (curva tratteggiata)

1.6 Teorema della codifica di canale

Siano $\mathbf{x}$ e $\mathbf{y}$ generiche N-ple in ingresso e in uscita dal canale. Si supporrà che gli alfabeti $\mathbf{x}$ e $\mathbf{y}$ siano discreti. L'estensione al caso continuo richiede solo di sostituire le probabilità con corrispondenti densità di probabilità e le somme con integrali.

Si generino $M = 2^{NR}$ *parole di codice* $\mathbf{x}_m$ formate da N-ple di simboli elementari. Le parole del codice siano scelte indipendentemente con probabilità $P(\mathbf{x})$. Si noti che si potrebbe essere così sfortunati da scegliere due o più volte la *stessa* parola di codice. Tuttavia per N grande ciò è estremamente improbabile.

1.6.1 Maggiorazione della probabilità di errore

Si utilizzi un ricevitore a massima verosimiglianza, che decide a favore di $\mathbf{x}_m$ se $P(\mathbf{y}/\mathbf{x}_m) > P(\mathbf{y}/\mathbf{x}_{m'})$ per ogni $m' \neq m$. In caso di pareggio il ricevitore decide a caso. Fissate le parole di codice, e supponendo di aver trasmesso $\mathbf{x}_m$, la probabilità d'errore è maggiorata da

$$P_{em} \leq \sum_{\mathbf{y}} P(\mathbf{y}/\mathbf{x}_m) \left(\frac{\sum_{m' \neq m} P(\mathbf{y}/\mathbf{x}_{m'})^{1/(1+\rho)}}{P(\mathbf{y}/\mathbf{x}_m)^{1/(1+\rho)}} \right)^{\rho} \tag{1.62}$$

per ogni $\rho \geq 0$. Infatti il termine che moltiplica $P(\mathbf{y}/\mathbf{x}_m)$ non è mai negativo, ed è maggiore o uguale a 1 per ogni $\mathbf{y}$ che produce errore. Si ottiene quindi

$$P_{em} \leq \sum_{\mathbf{y}} P(\mathbf{y}/\mathbf{x}_m)^{1/(1+\rho)} \left(\sum_{m' \neq m} P(\mathbf{y}/\mathbf{x}_{m'})^{1/(1+\rho)} \right)^{\rho} \tag{1.63}$$

Il valor medio di P_{em}, rispetto alla scelta casuale delle parole di codice, si calcola utilizzando l'indipendenza di $\mathbf{x}_m$ da tutti gli $\mathbf{x}_{m'}$:

$$E\left[P_{em}\right] \leq \sum_{\mathbf{y}} E\left[P(\mathbf{y}/\mathbf{x}_m)^{1/(1+\rho)}\right] \; E\left[\left(\sum_{m' \neq m} P(\mathbf{y}/\mathbf{x}_{m'})^{1/(1+\rho)} \right)^{\rho} \right] \tag{1.64}$$

Se si impone l'ulteriore vincolo $\rho \leq 1$ si può si può applicare alla funzione z^{ρ} la diseguaglianza di *Jensen*[18] $E\left[z^{\rho}\right] \leq \left(E\left[z\right]\right)^{\rho}$ per ottenere

$$E\left[P_{em}\right] \leq \sum_{\mathbf{y}} E\left[P(\mathbf{y}/\mathbf{x}_m)^{1/(1+\rho)}\right] \; E\left[\sum_{m' \neq m} P(\mathbf{y}/\mathbf{x}_{m'})^{1/(1+\rho)} \right]^{\rho} \tag{1.65}$$

Poiché risulta

$$E\left[P(\mathbf{y}/\mathbf{x}_m)^{1/(1+\rho)}\right] = E\left[P(\mathbf{y}/\mathbf{x}_{m'})^{1/(1+\rho)}\right] = \sum_{\mathbf{x}} P(\mathbf{x}) P(\mathbf{y}/\mathbf{x})^{1/(1+\rho)} \tag{1.66}$$

la probabilità media d'errore data dalla (1.65) è indipendente da m, e si ottiene

$$E\left[P_e\right] \leq (M-1)^{\rho} \sum_{\mathbf{y}} \left(\sum_{\mathbf{x}} P(\mathbf{x}) P(\mathbf{y}/\mathbf{x})^{1/(1+\rho)} \right)^{1+\rho} \tag{1.67}$$

Allo stesso risultato si sarebbe potuto arrivare anche per altra via: detta $\mathbf{x}_m$ la parola (casuale) di codice trasmessa, il valor medio della probabilità d'errore è dato da

$$E\left[P_e\right] = \sum_{\mathbf{x}_m} \sum_{\mathbf{y}} P(\mathbf{x}_m) P(\mathbf{y}/\mathbf{x}_m) P(\text{errore}/\mathbf{x}_m, \mathbf{y}) \tag{1.68}$$

Sia $A_{m'}$ l'evento $P(\mathbf{y}/\mathbf{x}_{m'}) \geq P(\mathbf{y}/\mathbf{x}_m)$. Si ha, per $0 \leq \rho \leq 1$,

$$P(\text{errore}/\mathbf{x}_m, \mathbf{y}) \leq \left(\sum_{m' \neq m} P(A_{m'}) \right)^{\rho} \tag{1.69}$$

Infatti la probabilità dell'unione di eventi non supera la somma delle singole probabilità; l'elevamento a potenza aumenta il termine tra parentesi, se minore di 1; lo lascia maggiore di 1 se già lo era. Si ha anche

$$P(A_{m'}) \leq \sum_{\mathbf{x}_{m'}: P(\mathbf{y}/\mathbf{x}_{m'}) \geq P(\mathbf{y}/\mathbf{x}_m)} P(\mathbf{x}_{m'}) \leq \sum_{\mathbf{x}_{m'}} P(\mathbf{x}_{m'}) \left(\frac{P(\mathbf{y}/\mathbf{x}_{m'})}{P(\mathbf{y}/\mathbf{x}_m)} \right)^{1/(1+\rho)} \tag{1.70}$$

[18] una semplice dimostrazione della diseguaglianza è la seguente: se $g(z)$ è una funzione concava, tale che $g(z) \leq g(E[z]) + K(z - E[z])$, moltiplicando per $p(z)$ e integrando si ottiene $E[g(z)] \leq g(E[z])$

Basta infatti osservare che tutti i termini da includere nella somma sono considerati con peso maggiore o uguale a 1 (anche dopo l'elevamento a potenza) e che si sono aggiunti altri termini non negativi. Si noti che il risultato non dipende da m', per cui in seguito si può indicare $\mathbf{x}_{m'}$ con un generico $\mathbf{x}$. Combinando le diseguaglianze (1.69) e (1.70) si ottiene

$$P(\text{errore}/\mathbf{x}_m, \mathbf{y}) \leq (M-1)^\rho P(\mathbf{y}/\mathbf{x}_m)^{-\rho/(1+\rho)} \left(\sum_{\mathbf{x}} P(\mathbf{x}) P(\mathbf{y}/\mathbf{x})^{1/(1+\rho)} \right)^\rho \tag{1.71}$$

e quindi, sostituendo nella (1.68),

$$E\left[P_e\right] \leq (M-1)^\rho \sum_{\mathbf{y}} \left(\sum_{\mathbf{x}} P(\mathbf{x}) P(\mathbf{y}/\mathbf{x})^{1/(1+\rho)} \right)^{1+\rho} \tag{1.72}$$

1.6.2 Canali senza memoria

Si supponga ora che il canale sia senza memoria:

$$P(\mathbf{y}/\mathbf{x}) = \prod_{n=1}^{N} P(y_n/x_n) \tag{1.73}$$

Si supponga anche che gli N simboli che costituiscono le parole di codice siano scelti indipendentemente, con probabilità $P(x)$. Si ha

$$\begin{aligned}
E\left[P_e\right] &< M^\rho \sum_{\mathbf{y}} \left[\sum_{x_1} \sum_{x_2} \cdots \sum_{x_N} \prod_{n=1}^{N} P(x_n) P(y_n/x_n)^{1/(1+\rho)} \right]^{1+\rho} \\
&= M^\rho \sum_{\mathbf{y}} \left[\prod_{n=1}^{N} \sum_{x_n} P(x_n) P(y_n/x_n)^{1/(1+\rho)} \right]^{1+\rho} \\
&= M^\rho \sum_{\mathbf{y}} \prod_{n=1}^{N} \left[\sum_{x_n} P(x_n) P(y_n/x_n)^{1/(1+\rho)} \right]^{1+\rho} \\
&= M^\rho \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \prod_{n=1}^{N} \left[\sum_{x_n} P(x_n) P(y_n/x_n)^{1/(1+\rho)} \right]^{1+\rho} \\
&= M^\rho \prod_{n=1}^{N} \sum_{y_n} \left[\sum_{x_n} P(x_n) P(y_n/x_n)^{1/(1+\rho)} \right]^{1+\rho}
\end{aligned} \tag{1.74}$$

Si osservi che tutti i termini del prodotto finale non dipendono da n, e sono quindi uguali. Enumerando con gli indici i e j rispettivamente gli alfabeti in ingresso e in uscita, indicando

con p_i le probabilità in ingresso e con $p_{j/i}$ le probabilità di transizione del canale, si ottiene

$$E\left[P_e\right] < M^{\rho}\left[\sum_j\left[\sum_i p_i p_{j/i}^{1/(1+\rho)}\right]^{1+\rho}\right]^N \tag{1.75}$$

1.6.3 Esponente d'errore

Ricordando che $M = 2^{NR}$ e definendo

$$E_0(\rho, p_i) = -\log\sum_j\left[\sum_i p_i p_{j/i}^{1/(1+\rho)}\right]^{1+\rho} \tag{1.76}$$

e infine scegliendo le p_i in modo da garantire la minor probabilità d'errore, ovvero ponendo

$$E_0(\rho) = \max_{p_i} E_0(\rho, p_i) \tag{1.77}$$

si ottiene

$$E\left[P_e\right] < 2^{-N[-\rho R + E_0(\rho)]} \tag{1.78}$$

Infine si può ottimizzare il valore di ρ, e definire l'*esponente d'errore*

$$E(R) = \max_{0\le\rho\le 1} E_0(\rho) - \rho R \tag{1.79}$$

e quindi

$$E\left[P_e\right] < 2^{-NE(R)} \tag{1.80}$$

da cui si vede che è possibile ridurre la probabilità media d'errore a valori piccoli a piacere, scegliendo N sufficientemente grande, purché sia $E(R) > 0$.

Poiché il risultato vale per la media di tutti i codici, esiste almeno un codice con probabilità d'errore non peggiore della media. Anzi con qualche altro calcolo è possibile dimostrare che si potrebbero teoricamente scegliere le parole del codice in modo che per ciascuna risulti

$$P_{em} < 4 \cdot 2^{-NE(R)} \tag{1.81}$$

$E(R)$ è interpretabile come inviluppo delle rette (1.79), mostrate in Fig. 1.4 nel caso di canale AWGN con ingresso binario $x = \pm 1$ e con rapporto segnale-rumore[19] $E_s/N_0 = 1$. E' facile verificare che in $\rho = 0$ la funzione $E_0(\rho)$ si annulla. Inoltre la derivata è pari

[19]il rapporto segnale-rumore sul canale è indicato in vari modi: σ_x^2/σ_n^2 mette in evidenza il modello $y = x + n$; E_s/N_0 sottolinea il canale continuo nel tempo su cui vengono trasmesse le forme d'onda $x\,s(t)$; E_b/N_0 considera l'energia spesa per ciascun *bit d'informazione*, data da E_s/R, anziché l'energia per simbolo; spesso il rapporto segnale-rumore è espresso in unità logaritmiche

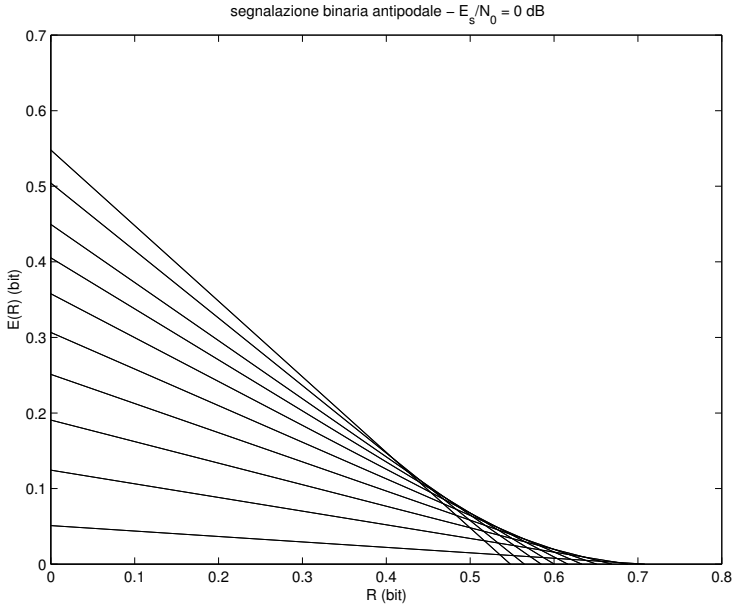

Figura 1.4: Esponente d'errore $E(R)$ ottenuto come inviluppo di rette di pendenza $-\rho$

alla capacità del canale: infatti con semplici calcoli si vede che la derivata di $E_0(\rho, p_i)$ è pari all'informazione mutua tra ingresso e uscita del canale, quando si utilizzi l'alfabeto in ingresso con probabilità p_i. E' poi immediato riconoscere (si veda la Fig. 1.4) che la funzione $E(R)$ si annulla in

$$R = \lim_{\rho \to 0} E_0(\rho)/\rho = \frac{dE_0(0)}{d\rho} = C \tag{1.82}$$

Si ha quindi il fondamentale risultato che è possibile trasmettere con probabilità d'errore piccola a piacere a qualunque ritmo minore della capacità del canale[20].

La Fig. 1.5 mostra $E(R)$, con $E_s/N_0 = 2$ dB, utilizzando la segnalazione binaria $x = \pm 1$. Nel caso della curva continua y è l'uscita *soft* del canale gaussiano; nel caso tratteggiato y è l'uscita *hard*, ottenuta dal segno del campione ricevuto: si ha un canale BSC con probabilità d'errore $\varepsilon = Q(\sqrt{\frac{2E_s}{N_0}}) = 0.0375$. E' evidente la perdita di capacità e prestazioni dovuta alle decisioni *hard*.

La Fig. 1.6 mostra, in funzione del rapporto segnale-rumore σ_x^2/σ_n^2, la probabilità d'errore per codici di lunghezza $N = 511$ con $M = 2^{484}$ parole (cioè $R = 484/511 \approx 0.95$). La curva continua mostra le prestazioni della media di tutti i codici scelti casualmente, valutate calcolando l'esponente d'errore per ogni rapporto segnale-rumore. La probabilità d'errore non nulla anche in assenza di rumore è dovuta alla possibilità di selezionare due o più volte la *stessa* parola di codice: la parola trasmessa ha almeno un concorrente coincidente con

[20] è anche possibile dimostrare che la probabilità d'errore non può essere piccola a piacere se il ritmo di trasmissione supera la capacità

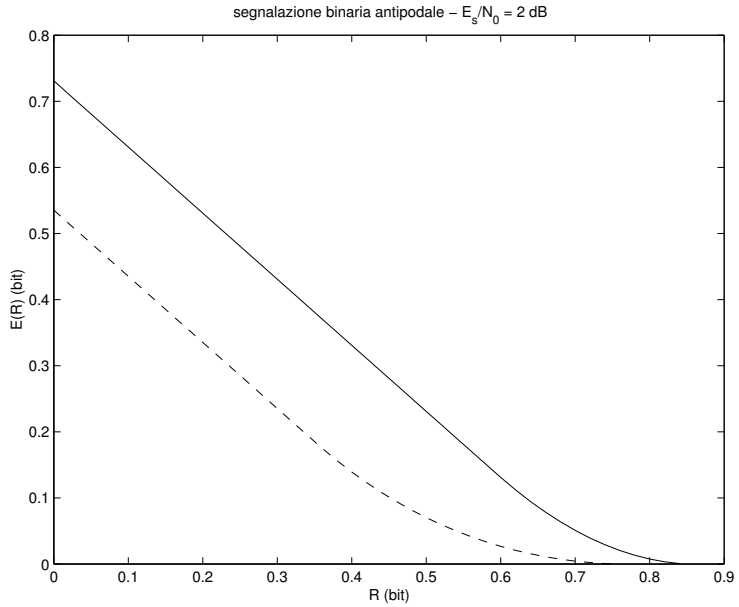

Figura 1.5: Esponente d'errore $E(R)$ con segnalazione binaria antipodale e uscita *soft* (canale gaussiano; curva continua) e *hard* (BSC; curva tratteggiata)

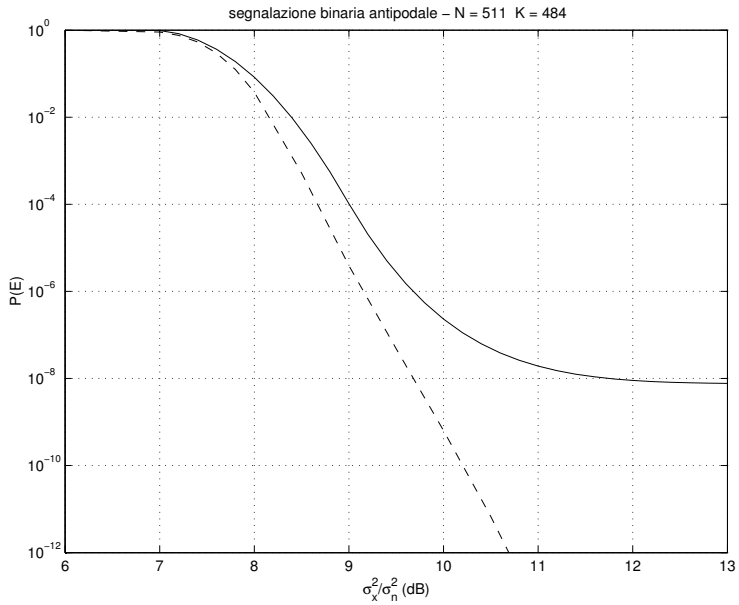

Figura 1.6: Probabilità d'errore, su canale gaussiano, di codici binari ($N = 511$; $M = 2^{484}$; $R = 484/511$): maggiorazione per la media dei codici mediante l'esponente d'errore (curva continua) e maggiorazione per il codice *BCH* (511,484) mediante il miglior *bound* disponibile (curva tratteggiata)

probabilità $1-(1-2^{-N})^{M-1} \approx 2^{N(1-R)} = 2^{-27} = 7.5 \cdot 10^{-9}$. Aumentando N (o anche diminuendo R) la probabilità di questo evento diventa trascurabile.

La curva tratteggiata di Fig. 1.6 mostra le prestazioni di un buon codice, il *BCH* (511,484), valutata con il miglior *bound* disponibile per un codice specifico. Nel calcolo di $P(E)$ si è usata la distribuzione esatta delle distanze tra le parole di codice. Questa è approssimativamente binomiale; tuttavia il codice è costruito in modo tale che le parole concorrenti di quella trasmessa differiscano in almeno 7 posizioni (ce ne sono $1.3 \cdot 10^7$ a distanza 7), con un netto miglioramento ad alto rapporto segnale-rumore. Il comportamento (lievemente) migliore anche a bassi e medi rapporti segnale-rumore è da attribuire alla distribuzione quasi binomiale delle distanze tra le parole di codice[21].

1.6.4 Considerazioni finali sulla codifica di canale

Il teorema della codifica di canale mostra che si può ridurre la probabilità d'errore in diversi modi, più o meno raffinati. Si può

- aumentare il rapporto segnale-rumore (maggior potenza trasmessa o minor rumore); soluzione banale con cui si aumentano la capacità C e l'esponente d'errore $E(R)$
- ridurre il ritmo di trasmissione; altra soluzione banale con cui si aumenta l'esponente d'errore $E(R)$ in modo che sia possibile ottenere buone prestazioni anche con blocchi di dimensione N piccola, cioè con codici semplici
- aumentare N, cioè la complessità del codice (e il ritardo nelle operazioni di codifica e decodifica); occorre però che le maggiori potenzialità del codice siano messe a frutto da ricevitori non lontani dall'ottimo
- evitare, per quanto possibile, di distruggere l'eventuale informazione *soft* contenuta nei simboli ricevuti; decisioni *hard* producono danni rilevanti, come mostra l'esempio di Fig. 1.5

Evidentemente non si possono costruire codici con N grande affidandosi al caso, non perché le prestazioni non sarebbero buone ma perché codifica e decodifica avrebbero complessità del tutto inaccettabile. Occorre una solida struttura matematica del codice che riduca enormemente la complessità delle operazioni richieste, in particolare in ricezione.

[21] codici costruiti casualmente hanno una distribuzione binomiale delle distanze solo *in media*; i codici che deviano troppo dalla media penalizzano le prestazioni dell'insieme; per N sufficientemente elevato il teorema limite centrale assicura che la distribuzione del singolo codice tenda alla media: tutti i codici tendono ad avere le stesse prestazioni

Capitolo 2

Introduzione alla codifica: codici a blocco

2.1 Codifica di canale

Questo capitolo introduttivo presenta, in modo volutamente semplificato, i concetti basilari della codifica di canale che verranno ampliati nel seguito. Il modo più tradizionale di vedere un codice correttore consiste nel considerare un canale binario simmetrico che commetta errori con probabilità maggiore di quella accettabile. Se il canale è stazionario e non ha memoria le posizioni degli errori sono casuali e le prestazioni dei codici sono molto più facili da analizzare. Canali più complessi, con memoria oppure con rapporto segnale-rumore variabile nel tempo, tendono a produrre errori a pacchetti (o *burst*). In tal caso sia la scelta del codice sia la valutazione delle prestazioni risultano più difficili.

Volendo semplificare al massimo, e considerando solo canali con ingresso e uscita discreti, sono possibili le seguenti strategie nell'uso dei codici:

- correzione degli errori, quando il loro numero non superi la *capacità di correzione* del codice; naturalmente si deve accettare il rischio, di cui si calcolerà la probabilità, di non riconoscere errori commessi dal canale oppure di correggere in modo errato, introducendo altri errori oltre a quelli dovuti al canale

- rivelazione degli errori e richiesta di ritrasmissione, purché esista un canale di ritorno e i ritardi così introdotti siano accettabili; anche in questo caso possono sfuggire blocchi errati, usualmente con probabilità molto minore rispetto al caso di correzione (usando gli stessi codici)

- strategie miste di correzione parziale, limitata ad un numero di errori minore di quelli correggibili dal codice, e di rivelazione degli errori nei restanti casi; la probabilità di avere un blocco errato in uscita dal decodificatore è intermedia tra quelle dei casi precedenti

Sempre restando nel caso discreto si potranno avere codici binari e non binari, ovvero con *cifre* d'informazione e di codice binarie o non binarie.

Volendo una classificazione dei tipi di codici più comuni si avranno:

- codici a blocco, nei quali ad esempio K cifre d'informazione sono seguite da $N - K$ cifre di parità, calcolate in modo deterministico dalle K cifre d'informazione
- codici convoluzionali, in cui ad esempio una cifra d'informazione è seguita da una di parità, in modo alternato; ogni cifra di parità dipende da un numero prefissato (solitamente non grande) di cifre d'informazione precedenti
- codici composti: codici concatenati; codici prodotto e turbo-codici (con tecniche di decodifica iterativa)

In questo capitolo introduttivo verranno considerati i codici a blocco: date K cifre d'informazione le regole del codice determinano la N-pla di cifre da inviare sul canale ($N > K$). Il codice è solitamente indicato come codice a blocco (N, K). Il rapporto K/N è detto *rate* del codice; nel caso binario coincide con il ritmo R di trasmissione dell'informazione, espresso in bit per simbolo.

2.1.1 Codici lineari

Si è osservato nel capitolo sulla teoria dell'informazione che per ridurre la complessità della codifica, e soprattutto della decodifica, occorre introdurre una efficace struttura nella definizione delle regole del codice. Per questo motivo in pratica si utilizzano quasi esclusivamente codici lineari: le cifre trasmesse sono combinazioni lineari delle cifre d'informazione (con algebra opportuna, ad esempio binaria).

Solitamente le prime K cifre del blocco codificato coincidono con quelle d'informazione. In tal caso il codice è detto *sistematico*: le K cifre d'informazione sono seguite da $N - K$ cifre di parità[1].

Si definisce distanza di *Hamming* tra due N-ple il numero di posizioni in cui esse differiscono.

Per un codice lineare è evidente che la somma di due parole di codice è parola di codice. In particolare la parola di tutti zeri è sempre parola di codice.

La distanza di *Hamming* tra due parole è pari al numero di elementi non nulli della parola di codice che ne è somma. Escludendo, perché di nessun interesse, il confronto di una parola con sé stessa l'insieme delle possibili distanze tra parole di codice diverse coincide con l'insieme del numero di elementi non nulli delle parole di codice non nulle.

[1] non è fondamentale, ovviamente, l'ordine con cui le cifre vengono trasmesse; solitamente le cifre di parià seguono quelle d'informazione

Su un canale di trasmissione binario, la capacità di correzione e di rivelazione degli errori dipendono dalla distanza minima di *Hamming* tra le parole di codice. Se ad esempio la distanza *minima* di *Hamming* tra parole di codice è $d = 3$, il codice può essere usato come *correttore* di un errore (in posizione qualsiasi nella N-pla): un solo errore lascia la N-pla ricevuta a distanza minore dalla parola trasmessa che da tutte le altre.

Lo stesso codice può essere usato come *rivelatore* di due errori: questi non producono comunque una parola di codice[2].

Analogamente se $d = 4$ il codice può essere usato come correttore di un errore (con due errori si può finire a mezza strada tra due parole di codice concorrenti), oppure come rivelatore di tre; se $d = 5$ il codice può correggere due errori oppure rivelarne quattro. La regola generale discende in modo evidente da questi esempi.

Si può rinunciare ad una parte della capacità di correzione per ridurre la probabilità di correzione errata. Ad esempio se $d = 5$ ci si può limitare a correggere un solo errore, lasciando la capacità di rivelarne altri due.

L'esempio più semplice di codice binario ha $N = K+1$: l'ultimo bit è la somma (nell'algebra binaria, cioè modulo 2) di *tutti* i K bit d'informazione[3]. Ad esempio se $K = 7$ i bit d'informazione 1 0 1 0 0 0 1 danno la parola di codice 1 0 1 0 0 0 1 1.

La parola di codice ha sempre un numero pari di uni (da cui deriva il nome dato al bit *di parità*). Qualunque errore singolo, cioè in una sola posizione tra le otto, viene rivelato. Vengono rivelate tutte le configurazioni con un numero dispari di errori, ma non quelle con numero pari. Evidentemente il codice non può correggere alcun errore, poiché $d = 2$.

2.1.2 Codici di *Hamming*

Un altro codice molto semplice è il *codice di Hamming* (7,4) in cui le tre cifre di parità sono combinazioni *diverse* delle quattro cifre d'informazione[4]:

$$p_1 = i_2 + i_3 + i_4$$

$$p_2 = i_1 + i_3 + i_4$$

$$p_3 = i_1 + i_2 + i_4$$

Si ottengono quindi le sedici parole di codice seguenti:

0 0 0 0 → 0 0 0 0 0 0 0

0 0 0 1 → 0 0 0 1 1 1 1

0 0 1 0 → 0 0 1 0 1 1 0

[2]evidentemente vengono rivelate anche molte configurazioni di tre o più errori, ma *non tutte*

[3]sarebbe ben strano tralasciare nell'*unico* bit di parità qualcuno dei bit d'informazione

[4]fin dai primi tentativi di costruzione di codici risultò evidente che trasmettere più volte una stessa combinazione di cifre d'informazione non è una soluzione brillante

$0\ 0\ 1\ 1 \rightarrow 0\ 0\ 1\ 1\ 0\ 0\ 1$

$0\ 1\ 0\ 0 \rightarrow 0\ 1\ 0\ 0\ 1\ 0\ 1$

$0\ 1\ 0\ 1 \rightarrow 0\ 1\ 0\ 1\ 0\ 1\ 0$

$0\ 1\ 1\ 0 \rightarrow 0\ 1\ 1\ 0\ 0\ 1\ 1$

$0\ 1\ 1\ 1 \rightarrow 0\ 1\ 1\ 1\ 1\ 0\ 0$

$1\ 0\ 0\ 0 \rightarrow 1\ 0\ 0\ 0\ 0\ 1\ 1$

$1\ 0\ 0\ 1 \rightarrow 1\ 0\ 0\ 1\ 1\ 0\ 0$

$1\ 0\ 1\ 0 \rightarrow 1\ 0\ 1\ 0\ 1\ 0\ 1$

$1\ 0\ 1\ 1 \rightarrow 1\ 0\ 1\ 1\ 0\ 1\ 0$

$1\ 1\ 0\ 0 \rightarrow 1\ 1\ 0\ 0\ 1\ 1\ 0$

$1\ 1\ 0\ 1 \rightarrow 1\ 1\ 0\ 1\ 0\ 0\ 1$

$1\ 1\ 1\ 0 \rightarrow 1\ 1\ 1\ 0\ 0\ 0\ 0$

$1\ 1\ 1\ 1 \rightarrow 1\ 1\ 1\ 1\ 1\ 1\ 1$

In totale si utilizzano $2^K = 16$ delle $2^N = 128$ configurazioni di 7 bit. Si può verificare che le 15 parole non nulle contengono almeno tre uni, e quindi che le parole di codice differiscono in almeno tre posizioni. Quindi $d = 3$ e il codice può rivelare tutti gli errori semplici e doppi (ed altri di peso maggiore) ed è in grado di correggere tutti gli errori semplici: per la correzione basta cercare la parola che differisce in un solo bit da quella ricevuta.

2.1.3 Matrice generatrice e di parità

Si può rappresentare la codifica nella forma

$$c = iG \tag{2.1}$$

dove c è il vettore di 7 elementi trasmesso (parola di codice), i è il vettore di 4 elementi delle cifre d'informazione, G è la *matrice generatrice* di dimensione $4 \cdot 7$:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Le parole di codice sono tutte le combinazioni lineari delle righe della matrice generatrice. Le tre equazioni di parità

$$c_5 = c_2 + c_3 + c_4$$

$$c_6 = c_1 + c_3 + c_4$$

$$c_7 = c_1 + c_2 + c_4$$

possono essere scritte come

$$cH = 0 \tag{2.2}$$

dove H è una matrice $7 \cdot 3$ detta *matrice di parità*[5]:

$$H = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.1.4 Sindrome

Ricevuta la N-pla $y = c + e$ dove e è il vettore errore, si calcola la *sindrome*

$$s = yH = cH + eH = eH \tag{2.3}$$

La sindrome è un vettore di $N - K = 3$ componenti, che dice se le tre regole di parità sono o meno soddisfatte.

La sindrome può presentarsi in $2^3 = 8$ modi, mentre le possibili configurazioni d'errore (inclusa la mancanza di errori) sono $2^7 = 128$. Si verifica facilmente che la stessa sindrome corrisponde a 128/8=16 diverse configurazioni d'errore. Fra queste in ricezione si sceglie la più probabile, cioè quella con il minor numero di errori.

Per i codici semplici (ma spesso troppo semplici, e quindi di nessun interesse) la ricerca può essere fatta a priori e memorizzata. Nel caso in esame si ha:

0 0 0 → 0 0 0 0 0 0 0

0 0 1 → 0 0 0 0 0 0 1

0 1 0 → 0 0 0 0 0 1 0

0 1 1 → 1 0 0 0 0 0 0

1 0 0 → 0 0 0 0 1 0 0

1 0 1 → 0 1 0 0 0 0 0

1 1 0 → 0 0 1 0 0 0 0

1 1 1 → 0 0 0 1 0 0 0

[5] in altri testi le matrici generatrice o di parità possono essere definite come trasposte di quelle qui utilizzate

Risultano correggibili tutti e soli gli errori singoli. Infatti con un errore in una sola posizione la sindrome coincide con la corrispondente riga della matrice di parità. Poiché le righe di H sono tutte diverse, e in numero pari ai possibili errori singoli (cioè N) tutti gli errori singoli sono correggibili. In generale si hanno infiniti codici di *Hamming* con parametri N e K legati dalla relazione

$$N = 2^{N-K} - 1 \tag{2.4}$$

Alcuni esempi sono:

$N = 3$, $K = 1$ (codice banale: 000,111)

$N = 7$, $K = 4$

$N = 15$, $K = 11$

$N = 31$, $K = 26$

...

$N = 1023$, $K = 1013$

...

Si noti che il *rate* K/N tende a 1 per $N \to \infty$, ma si corregge comunque un solo errore. Quindi N grande va bene solo per canali con probabilità d'errore già molto piccola, che viene ulteriormente ridotta dal codice. In tutti gli altri casi occorrono codici correttori di più di un errore.

A parità di ridondanza, codici con blocco lungo e correttori di molti errori sono migliori di codici con blocco corto e correttori di pochi errori, o di un solo errore. Però la complessità del decodificatore aumenta fortemente all'aumentare del numero di errori correggibili.

In ricezione, in linea di principio basta calcolare la sindrome $s = yH$ e se il codice è usato solo come rivelatore basta verificare se $s = 0$. Se invece il codice è usato come correttore dalla sindrome si ricava la N-pla d'errore più probabile, e si provvede a correggere la parola ricevuta.

Se $N - K$ è elevato, la correzione diventa molto pesante. E diventa addirittura troppo pesante il calcolo di yH. Le matrici generatrice e di parità sono concettualmente semplici, ma raramente utili in pratica e solo per codici di piccola dimensione. Occorre una descrizione del codice più sintetica, con una struttura algebrica più potente.

2.1.5 Rappresentazione polinomiale

Alla parola $c_{N-1}c_{N-2}\dots c_0$ di lunghezza N si può associare il polinomio $c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + \ldots + c_0$. La variabile x non ha alcun significato particolare: individua solo la posizione della cifra.

Analogamente a K cifre d'informazione $i_{K-1}i_{K-2}\dots i_0$ è associato il polinomio $i_{K-1}x^{K-1} + i_{K-2}x^{K-2} + \dots + i_0$.

Anche la matrice generatrice è sostituita da un *polinomio generatore* del codice $g(x)$, di grado $N-K$. Infine si stabilisce che $c_{N-1}c_{N-2}\dots c_0$ sia una parola di codice se e solo se il corrispondente polinomio $c(x)$ è divisibile per $g(x)$.

Si noti che mentre la matrice generatrice di un codice (N, K) contiene $K \cdot N$ elementi binari, il polinomio generatore ha solo $N-K+1$ coefficienti binari. Si vedrà in seguito che anche la matrice di parità può avere un equivalente *polinomio di parità*, di grado K.

2.1.6 Decodifica *hard* e *soft*

La decodifica algebrica (*hard*) dei codici *BCH* è totalmente incomprensibile senza buone nozioni di algebra dei campi finiti (campi di *Galois*). Tuttavia non risulta eccessivamente complessa, per codici correttori di, ad esempio, $5 \div 20$ errori.

Se il canale di trasmissione ha uscita continua, si prendono *decisioni indipendenti* bit per bit. Poi si effettua la ricerca di quell'unica parola di codice che non dista dalla ricevuta più del potere correttore t. Se sono avvenuti più di t errori la decodifica può fallire in più modi:

- non si trova una parola che disti meno di t; si rinuncia alla correzione (ma l'errore viene rivelato)
- si trova una parola, ma non è quella trasmessa (si accetta una parola con errori, non rivelati)

Per ridurre la probabilità di errori non rivelati si può limitare la correzione a meno di t errori.

Se all'uscita del canale è disponibile l'informazione *soft* y_n sul livello ricevuto (e non solo il segno) si distrugge informazione prendendo decisioni *hard* bit per bit. La capacità del canale e l'esponente d'errore ne risultano degradati. Si dovrebbe preferire la decodifica *soft*, ad esempio a massima verosimiglianza (ML). In presenza di rumore additivo gaussiano si dovrebbe calcolare *per ciascuna parola del codice* la correlazione

$$\sum_{n=1}^{N} y_n c_n \tag{2.5}$$

e scegliere il massimo. L'esperienza mostra che la decodifica *soft* guadagna circa 2 dB rispetto alla *hard*. Il problema è che le parole di codice sono 2^K, e se $K = 100$ si hanno $2^{100} \approx 10^{30}$ parole! Se non si trova una scorciatoia per non calcolare tutte le correlazioni non c'è speranza. Esistono metodi esatti di complessità proporzionale a $N \cdot 2^{N-K}$, già troppo elevata per gran parte dei codici.

Sono stati proposti metodi approssimati (*Chase*) basati sulla ricerca di un piccolo numero di parole di codice candidate, individuate decodificando in modo algebrico la parola ricevuta ed altre con alcuni bit (i meno affidabili) complementati. Entro questo insieme si determina la parola con correlazione migliore.

Tra le tecniche di decodifica *soft* vi è anche quella a massima probabilità a posteriori bit per bit: si calcola, per ciascun bit d'informazione, la probabilità che sia pari a 0, e si decide per lo zero se la probabilità supera 0.5. Esistono due metodi esatti (*Hartmann-Rudolph*; *Bahl et al.*) di complessità proporzionale a $N \cdot 2^{N-K}$, proibitiva per molti codici.

2.1.7 Decodifica a massima verosimiglianza e bit per bit

Per chiarire la differenza tra i due criteri basta il semplicissimo esempio di un codice a singola parità di dimensioni (3,2). Le quattro parole di codice sono 000, 011, 101 e 110. Si supponga che i tre campioni ricevuti y_n siano tali che $P(0/y_n)$ $(n = 1 \dots 3)$ valga rispettivamente 0.4, 0.7 e 0.7. Se si prendessero decisioni indipendenti sui tre bit si otterrebbe 100, che non è parola di codice, né si saprebbe poi individuare la parola più probabile.

Le probabilità delle quattro parole sono proporzionali rispettivamente a $0.4 \cdot 0.7 \cdot 0.7 = 0.196$, $0.4 \cdot 0.3 \cdot 0.3 = 0.036$, $0.6 \cdot 0.7 \cdot 0.3 = 0.126$ e $0.6 \cdot 0.3 \cdot 0.7 = 0.126$ (la costante di proporzionalità può essere ignorata). La decodifica a massima verosimiglianza sceglie tra le quattro la parola più probabile, ovvero 000. I due bit d'informazione decodificati sono quindi 00 (il terzo bit è di parità).

Se si vuol calcolare la probabilità a posteriori che il *primo* bit sia 0 o 1, tenendo conto delle regole del codice, occorre sommare le probabilità delle due parole 000 e 011, e confrontare con la somma delle probabilità degli altri due casi (di nuovo si può ignorare una costante di proporzionalità comune a tutte le parole). Il risultato è 0.232 contro 0.252, e quindi la decisione è a favore di 1. Analogamente si decodifica il secondo bit, ottenendo 0. E' curioso osservare che se si decodificasse allo stesso modo anche il terzo bit si otterrebbe 0: 100 non è una parola di codice, eppure questi sono i valori più probabili per i tre bit.

La decodifica a massima verosimiglianza, scegliendo la parola più probabile, rende minima la probabilità che il *blocco* di bit decisi sia sbagliato[6]. La decodifica bit per bit invece rende minima la probabilità che i *bit* decodificati siano errati[7]. Le differenze nelle prestazioni sono tuttavia spesso di ben poco rilievo. Ad esempio la Fig. 2.1 mostra la probabilità d'errore nella decodifica del codice di *Hamming* (31,26), in funzione del rapporto segnale-rumore E_b/N_0.

Sono mostrate anche le prestazioni, molto peggiori, del decodificatore *hard* che prima decide

[6]una buona strategia se un blocco contenente errori deve essere eliminato, come avviene nella trasmissione a pacchetti

[7]strategia ottima se si devono accettare tutti i bit, giusti o sbagliati che siano, e si vuole minimizzare il numero medio di errori

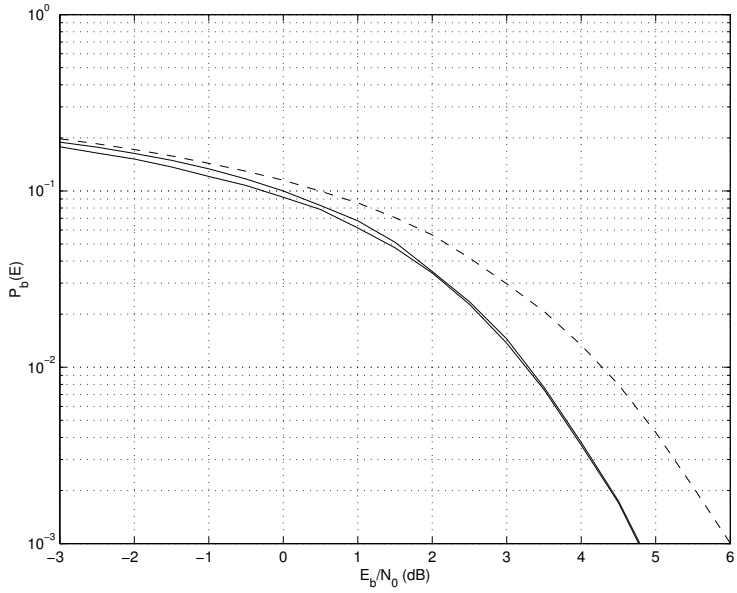

Figura 2.1: Probabilità che i bit decodificati siano errati con decodifica a massima verosimiglianza e bit per bit (curve continue) e con decodifica *hard* (curva tratteggiata) per il codice di *Hamming* (31,26)

bit per bit ignorando il codice e poi individua tra le parole di codice quella alla minima distanza di Hamming.

La Fig. 2.2 mostra analoghi risultati per un codice di maggiore complessità, e quindi migliori prestazioni: il codice convoluzionale a 256 stati con *rate* $R = 1/2$ che sarà descritto in un successivo capitolo.

La decodifica bit per bit è generalmente più complessa di quella a massima verosimiglianza. Se talvolta viene preferita non è per le prestazioni, ma perché fornisce non solo i bit decisi ma anche la probabilità che tali bit siano corretti. Questa informazione è preziosa negli schemi di codifica concatenata, in cui un decodificatore utilizza l'informazione *soft* prodotta da un altro decodificatore.

2.1.8 Codici generati in modo casuale

Si è già osservato che in linea di principio per costruire buoni codici con dimensione del blocco N grande basterebbe affidare al caso la generazione delle parole del codice. Può essere interessante vedere quali risultati si otterrebbero, e confrontare codici costruiti casualmente con i migliori codici conosciuti. Ciò risulta possibile solo per valori di N piuttosto piccoli.

Se davvero si generassero casualmente tutte le parole del codice si otterrebbe un codice non lineare. La complessità di codificatore e decodificatore ne sarebbe enormemente aumentata, senza vantaggio. Si potrebbe infatti dimostrare che codici a blocco *lineari* generati

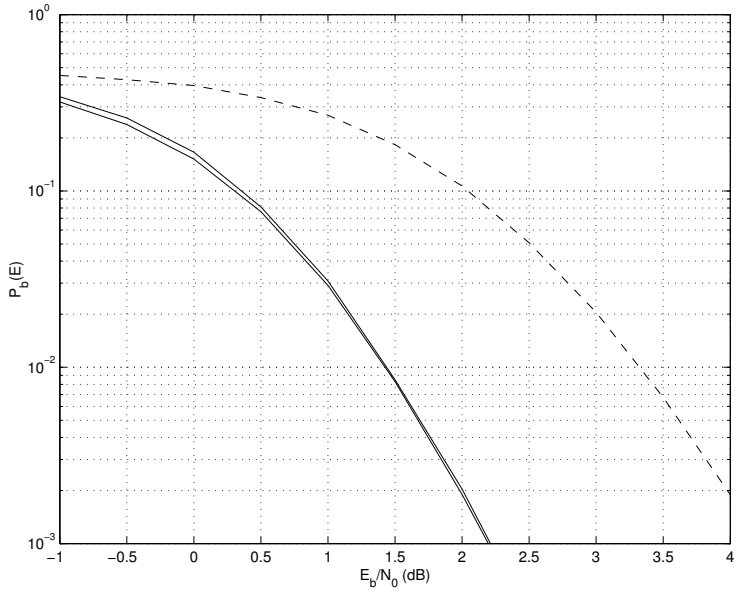

Figura 2.2: Probabilità che i bit decodificati siano errati con decodifica a massima verosimiglianza e bit per bit (curve continue) e con decodifica *hard* (curva tratteggiata) per il codice convoluzionale a 256 stati con *rate* $R = 1/2$

casualmente ottengono le stesse prestazioni dei non lineari. In pratica basta generare, scegliendole casualmente, K parole di codice da porre nelle righe della matrice generatrice. Tutte le parole di codice sono ottenute dalle 2^K combinazioni lineari di questa base.

La Fig. 2.3 mostra un primo esempio, con $N = 20$ e $K = 4$. La costruzione casuale è stata ripetuta 100 volte, e in figura è mostrata la probabilità d'errore (approssimata per eccesso mediante lo *union bound*) in funzione del rapporto segnale-rumore E_b/N_0 per ciascuno dei codici. La Fig. 2.4 mostra analoghi risultati per blocco di lunghezza $N = 100$, a parità di ridondanza ($K/N = 0.2$).

Le Fig. 2.5 e 2.6 mostrano i risultati con $K/N = 1/2$, per $N = 24$ e $N = 48$. Sono mostrate per confronto le prestazioni dei migliori codici di uguali dimensioni. Si può osservare che

- la dispersione delle prestazioni si riduce all'aumentare di N; le differenze tra le scelte più o meno fortunate diventano trascurabili, a probabilità d'errore di interesse pratico
- le prestazioni migliorano all'aumentare di N; la probabilità d'errore decresce circa esponenzialmente con N e il valore di E_b/N_0 necessario per una prefissata probabilità d'errore si riduce
- a basse probabilità d'errore le prestazioni sono determinate soprattutto dalla distanza minima del codice; la suddivisione in classi con diverse distanze minime è piuttosto evidente ad esempio nelle Fig. 2.5 e 2.6, in cui si vede anche che i codici prodotti dalla teoria hanno distanza minima migliore di quelli casuali. In passato si è data grande

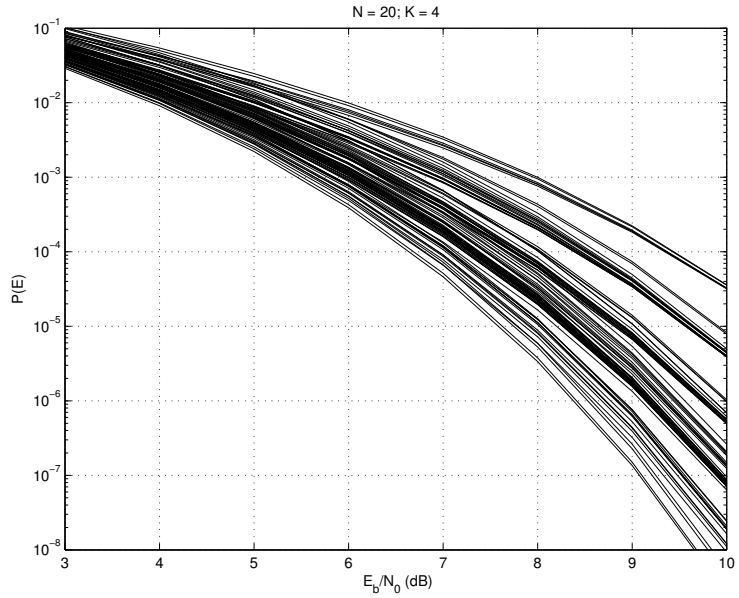

Figura 2.3: Probabilità d'errore (*union bound*) di 100 codici lineari generati casualmente ($N = 20$; $K = 4$)

importanza, forse esagerata, alla distanza minima dei codici; tuttavia resta vero che se interessano probabilità d'errore molto basse la distanza minima è il parametro dominante.

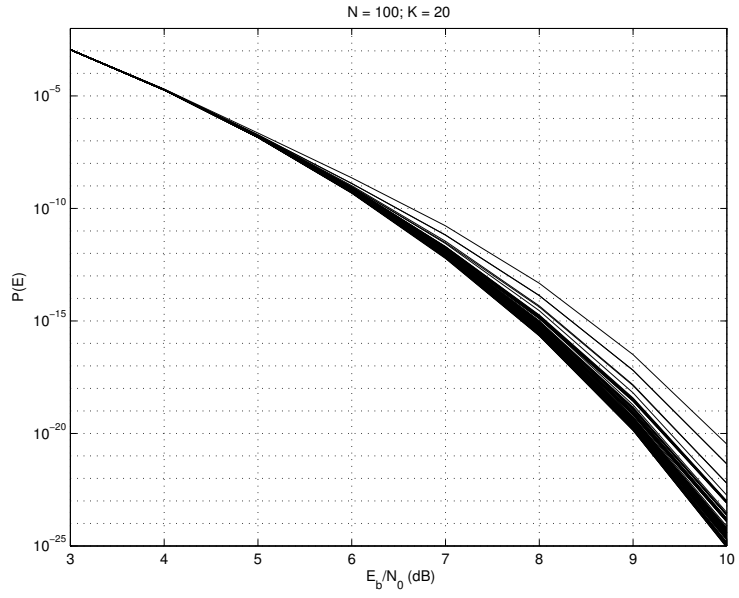

Figura 2.4: Probabilità d'errore (*union bound*) di 100 codici lineari generati casualmente ($N = 100$; $K = 20$)

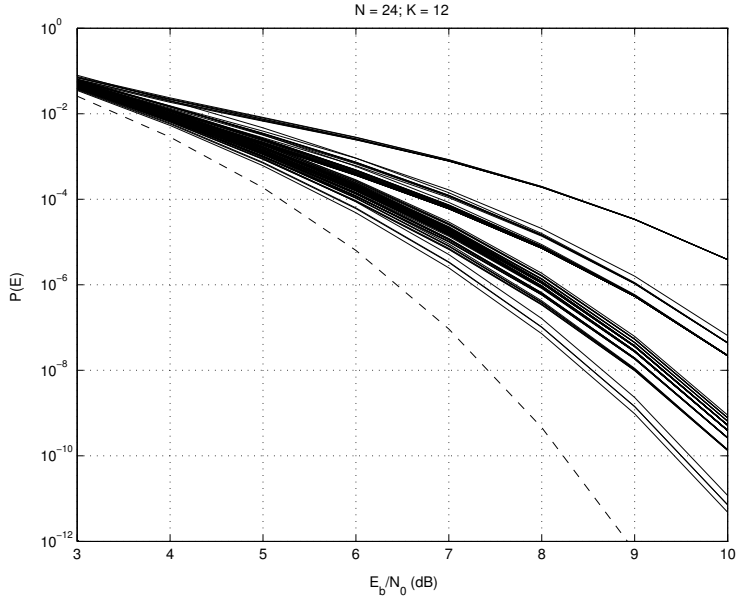

Figura 2.5: Probabilità d'errore (*union bound*) di 100 codici lineari generati casualmente ($N = 24$; $K = 12$) e del miglior codice delle stesse dimensioni (curva tratteggiata)

- è sorprendente che già per valori di N di poche decine il caso dia risultati così buoni, non solo in media ma anche nei casi peggiori; tuttavia una buona teoria è preferibile al caso

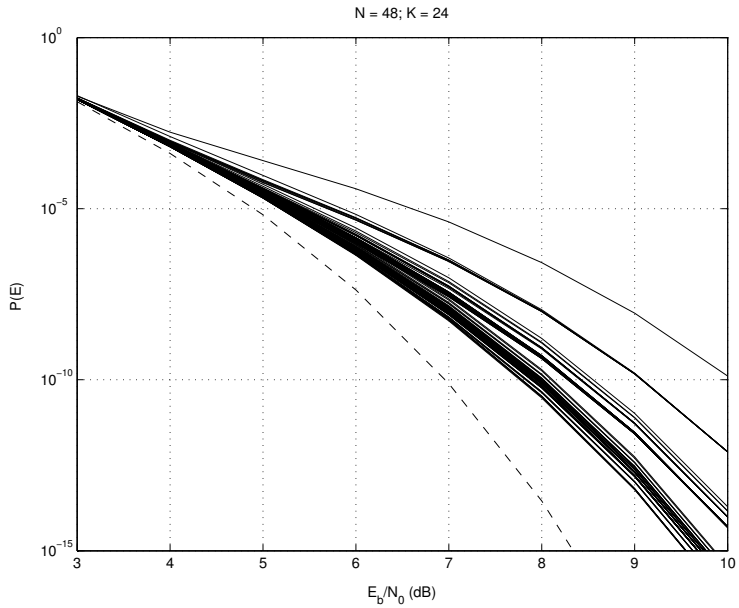

Figura 2.6: Probabilità d'errore (*union bound*) di 100 codici lineari generati casualmente ($N = 48$; $K = 24$) e del miglior codice delle stesse dimensioni (curva tratteggiata)

Se si aggiunge che la complessità di decodifica di un codice casuale potrebbe essere maggiore di quella di un codice con una solida struttura matematica si comprende perché alla teoria dei codici sia stata dedicata per molti decenni una così grande attenzione. Tuttavia senza lo stimolo delle straordinarie promesse della teoria dell'informazione la teoria dei codici non avrebbe appassionato tanti ricercatori e non avrebbe raggiunto i livelli attuali.

2.2 Codici ciclici

Sono codici polinomiali in cui $g(x)$ è un divisore di x^N+1. Ad esempio il codice di *Hamming* (15,11), in versione ciclica, ha

$$N = 15$$
$$K = 11$$
$$N - K = 4$$
$$g(x) = x^4 + x + 1$$
$$g(x)(x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1) = x^{15} + 1$$

I codici ciclici hanno la proprietà, da cui deriva il nome, che se $c_{N-1}c_{N-2}\dots c_0$ è parola di codice anche $c_{N-2}c_{N-3}\dots c_0c_{N-1}$ lo è. Codificatori e decodificatori dei codici ciclici sono un po' più semplici di quelli per codici non ciclici.

Esempio di parola del codice di *Hamming* (15,11):

$$i = 11000000001$$
$$i(x) = x^{10} + x^9 + 1$$
$$c(x) = g(x)i(x) = (x^4 + x + 1)(x^{10} + x^9 + 1) =$$
$$= x^{14} + x^{13} + x^4 + x^{11} + x^{10} + x + x^{10} + x^9 + 1 =$$
$$= x^{14} + x^{13} + x^{11} + x^9 + x^4 + x + 1$$
$$c = 110101000010011$$

In questa forma il codice non è sistematico. Si ottiene la versione sistematica trasmettendo dapprima le cifre d'informazione $i_{K-1}, \dots, i_0$, cioè il polinomio $i(x)x^{N-K}$ che ha $N-K$ zeri nelle ultime posizioni, dove verranno poste le cifre di parità. Queste sono scelte in modo che il polinomio trasmesso sia divisibile per $g(x)$. Basta sottrarre da $i(x)x^{N-K}$ il resto della divisione per $g(x)$ (se una divisione dà resto basta sottrarlo dal dividendo, e non vi sarà più resto!):

$$c(x) = i(x)x^{N-K} - R_{g(x)}\left[i(x)x^{N-K}\right] \tag{2.6}$$

dove $R_{g(x)}[\cdot]$ è il resto della divisione per $g(x)$, di grado massimo $N-K-1$. In algebra binaria la differenza è uguale alla somma. Il quoziente della divisione non interessa. Il codificatore ha una struttura molto semplice, che sarà mostrata in un successivo capitolo. Anche il circuito per il calcolo della sindrome in ricezione è molto semplice.

2.2.1 Codici *BCH*

Sono codici binari correttori di più errori, che includono come caso particolare i codici di *Hamming*. Non si conosce una spiegazione elementare delle proprietà di distanza dei codici *BCH*: è necessaria l'algebra dei campi finiti, per cui si rimanda al seguito.

Fra le possibili modificazioni dei codici, e quindi anche dei *BCH*, le più semplici e comuni sono:

- *estensione*: aggiunta di cifre di parità; l'esempio più comune è l'aggiunta di un bit di parità complessiva
- *accorciamento*: si possono modificare i valori di N e K, quando non siano convenienti, ponendo a zero e ovviamente *non trasmettendo* le prime b cifre d'informazione. In questo modo si ottiene un codice $(N-b, K-b)$. La distanza non diminuisce (potrebbe aumentare, ma di solito ciò non avviene). Nel caso dei codici ciclici basta saltare i primi b passi della codifica. Il codice ottenuto per accorciamento non è ciclico.

2.2.2 Codici *Reed-Solomon*

Fra i codici non binari di gran lunga più importante è la classe dei *Reed-Solomon*. L'alfabeto delle cifre d'informazione ha q elementi, dove q è un numero primo oppure una potenza di un numero primo. Il caso più comune è $q = 2^m$. In tal caso una cifra q-aria è rappresentabile anche con un byte di m bit.

I codici *Reed-Solomon* hanno parametri

$N = q - 1$

$K < N$ qualsiasi

$d = N - K + 1$

Un semplice esempio è:

$m = 8$ (byte di 8 bit)

$N = 2^8 - 1 = 255$

$K = 239$

$d = 17$

Il codice corregge 8 errori, cioè fino a 8 byte errati (non importa quanti bit errati contiene un byte errato).

Se lo si vuol considerare come un codice binario si hanno in totale $239 \cdot 8$=1912 bit d'informazione e $255 \cdot 8$=2040 bit totali.

Il codice può ovviamente essere accorciato (esempio: $N = 204$; $K = 188$).

2.2.3 Codici per errori concentrati a pacchetti

Esistono molte classi di codici per errori concentrati; solitamente la decodifica è agevole.

Il principale problema è avere un buon modello del canale, ovvero conoscere bene quanto possono concentrarsi gli errori. Un codice non adatto può addirittura fare peggio della trasmissione non codificata.

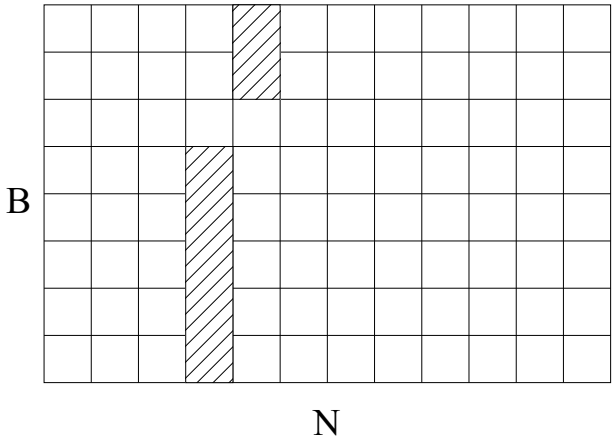

Figura 2.7: *Interleaving* a blocchi

Una semplice tecnica di codifica per canali con errori a pacchetti (*burst*), non raccomandata dalla teoria ma usata molto spesso in pratica, è l'*interleaving*, ad esempio a blocchi: in trasmissione si scrive in memoria per righe in una matrice $B \cdot N$ e si rilegge per colonne. In ricezione si effettua l'operazione opposta. Eventuali errori consecutivi commessi dal canale vengono quindi separati di almeno N simboli, purché la lunghezza del *burst* non superi B. Usando un codice a blocco di lunghezza N gli errori cadono in parole diverse e il codice può essere progettato per errori casuali.

Esistono anche *interleaver* convoluzionali. A parità di prestazioni richiedono (circa) metà memoria, ed introducono (circa) metà ritardo.

2.2.4 Codici concatenati

La figura 2.8 mostra un esempio semplice di concatenazione dei codici (concatenazione tradizionale, oggi detta *concatenazione serie*): codice interno *Hamming* (7,4) correttore di un errore; codice esterno *Reed-Solomon* (15,11) correttore di due errori, che opera su byte di 4 bit. Un blocco ha 44 bit d'informazione, suddivisi in 11 byte di 4 bit. Il codificatore esterno aggiunge 4 byte di parità (16 bit). Ad ogni byte di 4 bit il codificatore interno aggiunge 3 bit di parità. Il tutto equivale ad un codice $(15 \cdot 7{,}11 \cdot 4)$=(105,44).

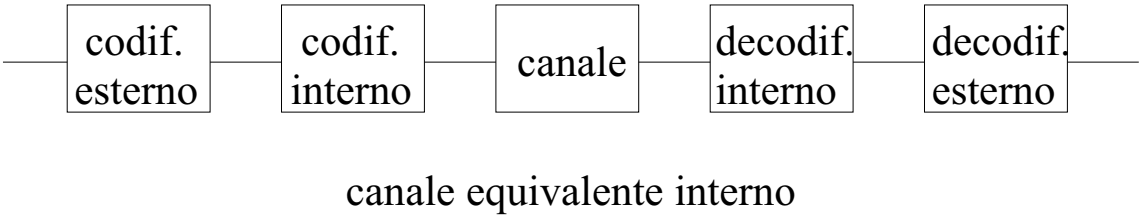

Figura 2.8: Codifica concatenata

Le prestazioni sono inferiori a quelle di un *BCH* accorciato (106,43), correttore di 10 errori, ma la decodifica è molto più semplice.

Nei codici concatenati tradizionali il codice esterno è sempre un *Reed-Solomon*. Il codice interno è un semplice codice a blocco (con decodifica *hard* o *soft*), oppure più spesso un convoluzionale con decodifica di Viterbi (*soft*). In questo caso occorre introdurre un *deinterleaver* per sparpagliare i blocchi di errori prodotti dal decodificatore interno, ed un corrispondente *interleaver* tra i due codificatori. Altrimenti verrebbe facilmente superato il potere correttore del codice esterno.

2.2.5 Codici prodotto

Un tipo particolare di codice concatenato è il prodotto di codici. Si supponga di disporre in una matrice con K_1 colonne e K_2 righe le cifre di informazione e di completare ciascuna riga con $N_1 - K_1$ cifre di parità calcolate sulla base di un codice *orizzontale* a blocco. Poi si completino le N_1 colonne, che contengono sia cifre d'informazione sia di parità, con $N_2 - K_2$ cifre di parità di un codice *verticale*. Si sono ottenute in questo modo le parità orizzontali e verticali delle K_1K_2 cifre d'informazione, organizzate rispettivamente per righe e per colonne, nonché le parità verticali delle parità orizzontali. Queste ultime sono dette più semplicemente *parità delle parità* poiché è facile verificare che si otterrebbe lo stesso risultato codificando prima per colonne e poi per righe. In totale si hanno N_1N_2 bit di codice, e il *rate* complessivo del codice è il prodotto dei *rate* dei codici componenti. In ogni riga e ogni colonna vi è una parola di codice (orizzontale e verticale).

Non è difficile mostrare che le parole del codice prodotto di peso minimo si ottengono disponendo una stessa parola orizzontale di peso minimo nelle righe corrispondenti ad una parola verticale di peso minimo, e che quindi la distanza minima del codice è il prodotto delle distanze minime dei codici componenti.

La costruzione può essere estesa a tre o più dimensioni, disponendo matrici bidimensionali su più strati. Si possono quindi raggiungere distanze minime elevate con codici componenti semplici.

I codici prodotto sono noti da lungo tempo, ma non avevano trovato applicazione pratica per la difficoltà di decodificarli in modo efficiente. L'idea più semplice è la decodifica *hard* riga per riga con correzione degli errori così individuati, poi per colonne con ulteriore

correzione, poi per strati (nel caso di tre dimensioni), ecc. ma non dà risultati soddisfacenti. Le prestazioni migliorano molto se la decodifica è *soft* e *iterativa* con una tecnica simile a quella usata per i turbo codici.

2.3 Prestazioni dei codici a blocco

- La probabilità di decisione corretta, per un codice utilizzato per correggere fino ad C errori[8] è

$$P_{cd} = \sum_{i=0}^{C} \binom{N}{i} p^i (1-p)^{N-i} \tag{2.7}$$

  dove p è la probabilità che il singolo simbolo sia ricevuto errato. Ad esempio con segnalazione binaria antipodale con energia $E_s = E_b R = E_b K/N$ e rumore gaussiano

$$p = Q\left(\sqrt{\frac{2E_b}{N_0}\frac{K}{N}}\right) \tag{2.8}$$

  e quindi p aumenta diminuendo il *rate* K/N.

- Per un codice binario, la probabilità che i bit a valle del decodificatore siano errati è maggiorata da

$$P_b \leq \frac{1}{N} \sum_{i=d-C}^{N} (i+C) \binom{N}{i} p^i (1-p)^{N-i} \tag{2.9}$$

  Infatti la decodifica può essere errata solo per parole che contengano già almeno $d-C$ errori; il decodificatore ne può aggiungere al massimo C.

- Per codici utilizzati solo come rivelatori di errori, se si conosce il numero A_i di parole di codice di peso i per ogni i, è facile calcolare la probabilità P_{icd} di errori non rivelati:

$$P_{icd} = \sum_{i \neq 0} A_i p^i (1-p)^{N-i} \tag{2.10}$$

- Anche per codici utilizzati come correttori di errori è possibile (ma non facilissimo) calcolare la probabilità P_{icd} di decisione errata non rivelata.

[8] C è un intero che non ha nulla a che fare con la capacità; ovviamente $C \leq t$

Capitolo 3

Introduzione alla codifica: codici convoluzionali

3.1 Codici convoluzionali

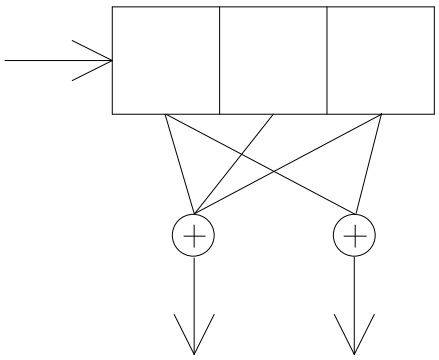

Figura 3.1: Semplice codificatore convoluzionale

La struttura di un semplice codificatore convoluzionale è mostrata in Fig. 3.1. La lunghezza K del registro a scorrimento ($K = 3$ nell'esempio) è detta *constraint length*. All'inizio i registri sono azzerati. Entra un bit d'informazione per volta e ne escono due, combinazioni lineari del bit attuale e di alcuni precedenti (due, in figura). I bit codificati sono convoluzioni (in algebra binaria) della sequenza in ingresso con due diverse risposte impulsive: indicando con $i(D)$ la trasformata zeta della sequenza dei bit d'informazione[1]

$$i(D) = i_0 + i_1 D + i_2 D^2 + \dots \tag{3.1}$$

[1] D sta per *delay*; ogni altra variabile (x, z, Z, ...) sarebbe equivalente, ma D è il simbolo più spesso usato in questo contesto

e con $c_1(D)$ e $c_2(D)$ le due sequenze in uscita si ha

$$\begin{aligned} c_1(D) &= i(D)(1 + D + D^2) \\ c_2(D) &= i(D)(1 + D^2) \end{aligned} \tag{3.2}$$

Per quanto lunga sia la sequenza in ingresso, un generico bit d'informazione ha effetto sui bit codificati solo in un intervallo molto limitato, contrariamente ai codici a blocco.

Conviene distinguere, nel contenuto del registro a scorrimento di lunghezza $K = 3$, una parte riservata al bit attuale ed una contenente i $K-1 = 2$ bit del passato, che costituiscono lo *stato* del codificatore. Sono possibili 2^{K-1} stati. Le uscite attuali dipendono sia dallo stato sia dal dato attuale. Lo stato al passo successivo, corrispondente all'ingresso di un nuovo bit d'informazione, è anch'esso funzione dello stato e del dato attuale.

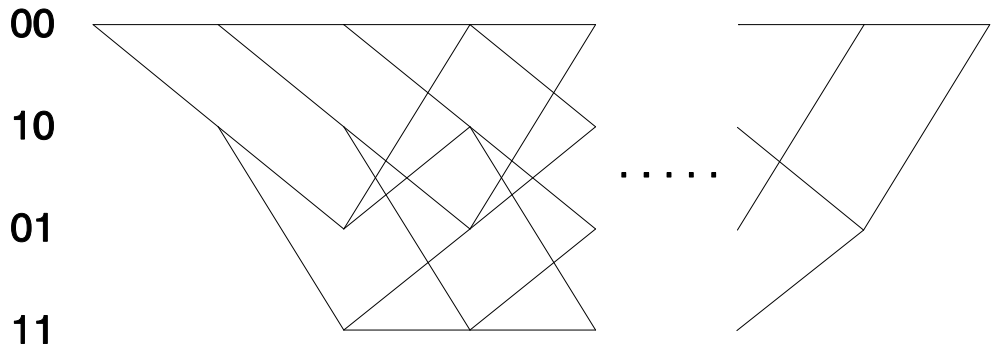

Figura 3.2: Traliccio delle transizioni di stato del codice convoluzionale di Fig. 3.1

Le *transizioni di stato* possibili per i primi passi sono indicate in Fig. 3.2, dove gli stati sono enumerati secondo il contenuto della coppia di registri di memoria, nell'ordine in cui appaiono in Fig. 3.1. Si osservi che da ogni stato se ne possono raggiungere solo due, corrispondenti ai due valori del bit d'informazione che si sposta dalla prima alla seconda cella, così come ogni stato è raggiungibile solo da due, poiché viene scartato un solo bit dal registro a scorrimento. In due passi ($K - 1$, in generale) è possibile raggiungere qualsiasi stato, dopo di che il diagramma di Fig. 3.2, detto *traliccio* (*trellis*), si ripete invariato. Quando eventualmente la sequenza di bit d'informazione termina, risulta conveniente per motivi che si vedranno in seguito forzare lo stato finale ad uno noto, ad esempio quello nullo, terminando la sequenza con $K - 1$ zeri.

Si noti che le transizioni di stato, mostrate in Fig. 3.2, dipendono solo dalla particolare struttura a registro a scorrimento, e non dalle connessioni tra celle del registro e sommatori modulo 2. Inoltre ciascuna transizione di stato individua non solo il contenuto attuale delle celle di memoria (stato iniziale), ma anche il bit d'informazione attuale (dato dal primo bit dello stato finale). Ogni sequenza di bit d'informazione è in corrispondenza biunivoca con una successione di transizioni di stato, cioè con un percorso nel traliccio. Ad ogni biforcazione il percorso superiore corrisponde all'ingresso di uno zero e quello inferiore ad

un uno. Se si numerano progressivamente gli stati a partire da 0 a $2^{K-1}-1$ (o da 1 a 2^{K-1}) tutti gli stati pari (o dispari) corrispondono al bit d'informazione zero.

I bit codificati, e di conseguenza la forma d'onda trasmessa, dipendono sia dal contenuto dei registri sia dalle connessioni con i sommatori modulo 2. Per tutti i valori di K di interesse pratico la ricerca di buoni codici è stata effettuata in modo esaustivo provando tutte le possibili connessioni, con qualche regola per scartare al più presto le cattive soluzioni. I risultati sono tabulati sui libri di codici.

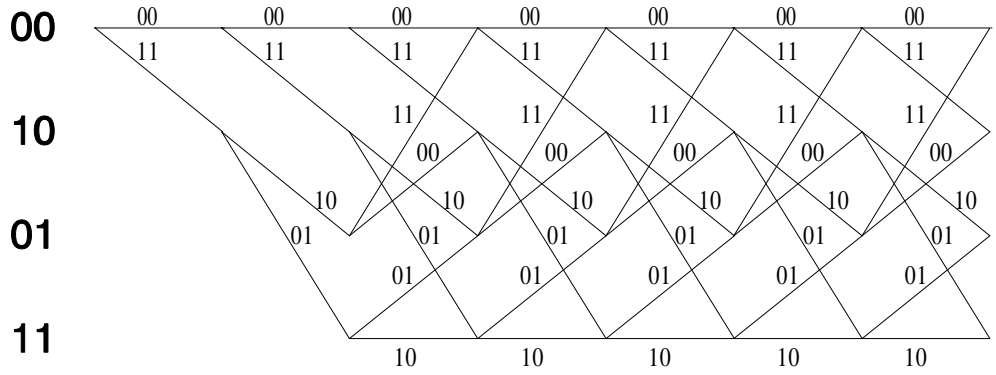

Figura 3.3: Traliccio del codificatore di Fig. 3.1; sono mostrati anche i bit codificati

Il codice di Fig. 3.1 è, ad esempio, il migliore fra quelli a quattro stati[2]. In Fig. 3.3 è riprodotto il traliccio, con l'aggiunta delle coppie di bit codificati corrispondenti a ciascuna transizione di stato (già per otto stati la figura risulterebbe quasi illeggibile, e converrebbe sostituirla con una tabella). Il traliccio mostra tutti i possibili segnali, che sono 2^L se L è il numero dei passi, e quindi le coordinate del vettore trasmesso: ad esempio con segnalazione binaria antipodale 1 corrisponde a $\sqrt{E_s}$ e 0 a $-\sqrt{E_s}$ (o viceversa), dove E_s è l'energia del simbolo inviato sul canale. Si noti che il codice non è sistematico. Contrariamente ai codici a blocco, i codici convoluzionali non sistematici sono generalmente migliori dei sistematici.

3.2 Decodifica a massima verosimiglianza

Il ricevitore a massima verosimiglianza richiede il calcolo delle correlazioni del vettore ricevuto con tutti i possibili segnali. Una generica correlazione può essere calcolata sommando via via le coppie di termini corrispondenti a ciascuna transizione di stato. Nel primo intervallo occorre calcolare le due correlazioni $-y_1-y_2$ e y_1+y_2; nel secondo occorre aggiungere a queste, rispettivamente, $-y_3-y_4$, y_3+y_4, y_3-y_4 e $-y_3+y_4$; nel terzo intervallo le quattro ipotesi finora considerate si biforcano ancora, ed alle correlazioni già calcolate si deve sommare una delle quattro combinazioni $\pm y_5 \pm y_6$, secondo le indicazioni del traliccio. La novità è che le otto strade convergono a due a due in uno stesso stato, ed è possibi-

[2]una variante banale è scambiare i due bit codificati

le scartare definitivamente la peggiore delle due mantenendo solo quella *sopravvissuta* al confronto (una per ciascuno stato; si memorizza il percorso ed il valore della correlazione).

Al passo successivo le quattro strade sopravvissute si biforcano nuovamente, ma di nuovo si incontrano a due a due, per cui si potrà scartare allo stesso modo un'altra metà delle ipotesi. In definitiva occorrerà effettuare ad ogni passo quattro aggiornamenti e confronti, e conservare le informazioni relative ai quattro percorsi sopravvissuti.

E' questo il famosissimo *algoritmo di Viterbi*, proposto verso la fine degli anni '60 per i codici convoluzionali ma che risulta utile anche in altri contesti, come si vedrà in seguito. La complessità è proporzionale al numero 2^{K-1} degli stati, e cresce solo linearmente con la lunghezza L della sequenza. In altri termini nel funzionamento in tempo reale basta che nel tempo di un bit d'informazione vengano effettuate tutte le operazioni di somma, confronto e memorizzazione per tutti gli stati[3].

Il codice a quattro stati risulta fin troppo semplice in pratica; molto usati sono quelli a 64 e 256 stati. Naturalmente i livelli ricevuti y_k sono rappresentati con precisione finita, cioè sono quantizzati. Si scopre che un bit per il segno e due o tre per la parte frazionaria sono largamente sufficienti.

Unico inconveniente, se L non è piccolo, è la dimensione della memoria per i cammini sopravvissuti. Peraltro risulta difficile credere che si debba attendere la fine della trasmissione per decidere sui primi bit d'informazione. Ed infatti avviene il seguente fenomeno: ad un generico istante, con elevata probabilità i 2^{K-1} cammini sopravvissuti coincidono fino a $4 \div 5K$ passi precedenti. Il decodificatore può quindi già annunciare le relative decisioni, dopo di che è inutile mantenere in memoria la parte già decisa. Ad ogni passo sono memorizzati 2^{K-1} cammini sopravvissuti per una lunghezza pari ad esempio a $5K$ transizioni di stato, e viene emessa la decisione relativa al bit che sta uscendo dalla memoria disponibile[4].

Quando la trasmissione ha termine si è costretti a prendere una decisione anche sugli ultimi bit d'informazione, che risulterebbero quindi meno affidabili. E' questo il motivo per cui si preferisce avere uno stato finale certo, ottenuto forzando a zero il contenuto finale dei registri, e quindi un solo cammino sopravvissuto da considerare. Se si aggiungono all'ingresso del decodificatore $K-1$ zeri è facile verificare che negli $L+K-1$ passi nel traliccio si hanno 2^L percorsi distinti, ovvero 2^L parole di codice.

Una volta capito l'algoritmo di Viterbi risulta facile vedere le modifiche richieste con altri valori dei parametri del codice. Una modifica banale è variare il numero di bit di codice per bit d'informazione, portandolo ad esempio a tre: il codificatore ha tre sommatori, anziché due; ogni ramo nel traliccio corrisponde ad una terna di bit, e le correlazioni vengono aggiornate sommando tre contributi. Il rapporto tra bit d'informazione e bit codificati (pari a 1/3, nel caso in esame) è il *rate* del codice.

[3] la struttura a farfalle del traliccio consente una forte parallelizzazione, con unità elementari che trattano coppie di stati

[4] occasionalmente i 2^{K-1} percorsi sopravvissuti non coincidono, ma si è forzati comunque a prendere una decisione; ad esempio si potrà scegliere il bit d'informazione corrispondente alla sequenza sopravvissuta con la miglior correlazione al momento attuale

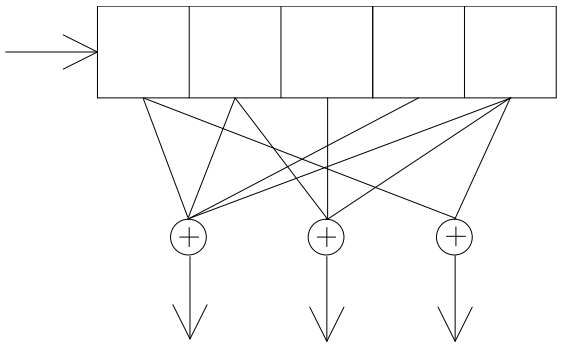

Figura 3.4: Codificatore convoluzionale con *rate* $R = 2/3$ a otto stati; i bit entrano a coppie

Più interessante è il caso di *rate* con numeratore $b \neq 1$, ad esempio $R = 2/3$. La soluzione più intuitiva è far entrare i bit d'informazione nel registro a scorrimento a due per volta, ed avere tre sommatori, come in Fig. 3.4 che rappresenta un codificatore a $2^{K-b} = 8$ stati[5]. La novità è che ad ogni transizione di stato si hanno due bit freschi, e quindi da ogni stato se ne possono raggiungere quattro ed in ogni stato se ne ricongiungono quattro. L'algoritmo di Viterbi esegue dunque confronti tra quattro ipotesi, e mantiene solo la migliore. Tanto meglio, verrebbe da dire, si sfoltisce maggiormente; però i confronti tra quattro concorrenti sono un po' più fastidiosi di quelli tra due. Ed infatti è stata anche proposta una diversa soluzione, basata sulla *perforazione* di codici con *rate* 1/2 (codici *punctured*): una volta si prelevano entrambi i bit codificati, ed una volta uno solo, periodicamente; in totale si sono trasmessi tre bit ogni due d'informazione. Per ottenere $R = 3/4$ basta considerare prima due bit, poi uno, poi ancora uno. In ricezione basta porre $y_k = 0$ in corrispondenza dei bit mancanti, esattamente a metà strada tra i livelli nominali, ed utilizzare il normale algoritmo di Viterbi. Particolarmente apprezzati sono i codici *punctured* universali, cioè che offrono buone prestazioni per molti valori di R ad esempio da 1/2 a 7/8, modificando solo le regole di perforazione[6].

E quali bit è meglio eliminare? L'analisi esaustiva di tutte le possibili perforazioni conduce alle tabelle dei migliori tra tali codici. Le prestazioni dei codici convoluzionali *punctured* sono quasi equivalenti a quelle dei migliori codici, a parità di numero di stati.

[5]osservando che i bit d'informazione *pari* e *dispari* vanno in celle distinte del registro a scorrimento, il codificatore può essere rappresentato in modo equivalente con due registri di tre e due celle, alimentati in parallelo

[6]però gli eventi errore dei codici *punctured* sono decisamente più lunghi, perché una distanza consistente viene accumulata solo su un gran numero di simboli; nel decodificatore occorre una memoria per i percorsi sopravvissuti pari ad alcune volte quella richiesta dal codice con *rate* 1/2

3.3 Codici convoluzionali recursivi sistematici

I migliori codici convoluzionali sono molto spesso non sistematici. Ciò non comporta alcuna difficoltà nella decodifica. Infatti l'associazione tra percorsi nel traliccio e bit d'informazione è molto semplice. Tuttavia ogni codice convoluzionale può essere realizzato anche nella forma sistematica se si accetta un codificatore con struttura retroazionata. Si consideri ad esempio il seguente codice a otto stati:

$$\begin{aligned} c_1(D) &= i(D)(1 + D^2 + D^3) \\ c_2(D) &= i(D)(1 + D + D^3) \end{aligned} \tag{3.3}$$

Il codificatore può essere realizzato come cascata di due blocchi: il primo calcola $i'(D) = i(D)(1 + D^2 + D^3)$, e può essere considerato un codice con *rate* $R = 1$; il secondo produce

$$\begin{aligned} c_1(D) &= i'(D) \\ c_2(D) &= i'(D)\frac{1 + D + D^3}{1 + D^2 + D^3} \end{aligned} \tag{3.4}$$

è un codificatore sistematico retroazionato. Codici con questa struttura retroazionata sono anche detti *recursivi sistematici*. Si osservi che la relazione tra $i'(D)$ e $i(D)$ è invertibile:

$$i(D) = i'(D)\frac{1}{1 + D^2 + D^3} \tag{3.5}$$

Ad ogni possibile sequenza $i(D)$ di bit d'informazione corrisponde una *distinta* sequenza $i'(D) = i(D)(1 + D^2 + D^3)$, e viceversa. Si tratta quindi di sequenze d'informazione equivalenti. L'insieme delle sequenze codificate prodotte dal codificatore recursivo sistematico (3.4), alimentato da $i'(D)$ o da $i(D)$, è lo stesso. In pratica si alimenta il codificatore (3.4) direttamente con la sequenza d'informazione $i(D)$.

Sono possibili diverse implementazioni *canoniche* del circuito che produce $c_2(D)$. Due di queste sono mostrate nelle Fig. 3.5 e 3.6. Alle diverse realizzazioni possono corrispondere modi diversi di enumerare gli stati del traliccio; le diverse realizzazioni sono comunque equivalenti. Cambia la corrispondenza tra percorsi nel traliccio e bit d'informazione, ma di ciò si tiene conto facilmente in fase di decodifica.

Si può mostrare che anche nel caso generale di *rate* $R = b/n$ ogni codice convoluzionale ha un equivalente recursivo sistematico.

In passato non c'era un vero motivo per preferire i codici sistematici a quelli non sistematici. Con l'introduzione dei *turbo codici*, codici complessi ottenuti componendo codici *sistematici* relativamente semplici, i codici convoluzionali recursivi sistematici hanno assunto grande importanza.

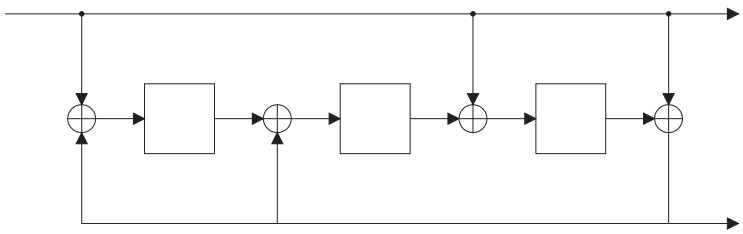

Figura 3.5: Possibile realizzazione del codificatore convoluzionale recursivo sistematico $(1, \frac{1+D+D^3}{1+D^2+D^3})$

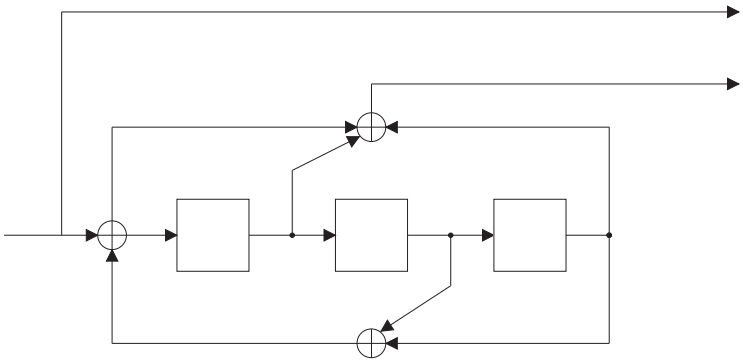

Figura 3.6: Altra possibile realizzazione del codificatore convoluzionale recursivo sistematico $(1, \frac{1+D+D^3}{1+D^2+D^3})$

3.4 Decodifica bit per bit

La decodifica bit per bit, pur proposta decine di anni fa, non ha avuto interesse pratico fino a quando non sono stati introdotti codici concatenati che richiedono all'uscita dei decodificatori dei codici componenti le probabilità di zeri e uni decodificati. Il metodo di decodifica con uscita *soft* utilizzato per i codici convoluzionali è quello di *Bahl*[7].

In ogni sezione del traliccio vengono valutate le seguenti quantità:

- probabilità a posteriori $\alpha_t(S_t)$ di ciascuno stato S_t al tempo t, condizionata ai campioni ricevuti dall'istante iniziale fino al tempo t
- probabilità a posteriori $\beta_t(S_t)$ di ciascuno stato S_t al tempo t, condizionata ai campioni ricevuti dall'istante $t+1$ alla fine
- probabilità a posteriori $\gamma_t(S'_{t-1}, S_t)$ di ciascuna transizione di stato $S'_{t-1} \rightarrow S_t$, condizionata ai soli campioni ricevuti al tempo t

Nel calcolo di tutte queste probabilità si possono ignorare costanti moltiplicative comuni, recuperabili con semplici normalizzazioni. E' facile verificare che le variabili $\alpha_t(S_t)$ e $\beta_t(S_t)$

[7]spesso indicato con BCJR per onorare i quattro autori

possono essere calcolate con due recursioni, rispettivamente in avanti e all'indietro:

$$\alpha_t(S_t) = \sum_{S'_{t-1}} \alpha_{t-1}(S'_{t-1})\gamma_t(S'_{t-1}, S_t) \tag{3.6}$$

$$\beta_t(S_t) = \sum_{S'_{t+1}} \beta_{t+1}(S'_{t+1})\gamma_{t+1}(S_t, S'_{t+1}) \tag{3.7}$$

dove le somme sono estese a tutte le transizioni di stato possibili nel traliccio. Ad esempio nel calcolo di $\alpha_t(S_t)$ si sommano le probabilità di tutti gli stati S'_{t-1} che sono connessi nel traliccio allo stato S_t, ciascuna moltiplicata per la probabilità della transizione $S'_{t-1} \to S_t$.

Una volta calcolati i valori di $\alpha_t(S_t)$ e $\beta_t(S_t)$ ad ogni istante t e per tutti gli stati S_t, le probabilità a posteriori dello stato S_t condizionate a tutti i campioni ricevuti sono date da $\alpha_t(S_t)\beta_t(S_t)$. Analogamente le probabilità delle transizioni di stato $S'_{t-1} \to S_t$ condizionate a tutti i campioni ricevuti sono date da $\alpha_{t-1}(S'_{t-1})\gamma_t(S'_{t-1}, S_t)\beta_t(S_t)$.

Da queste grandezze è facile calcolare, sommando sugli opportuni sottoinsiemi di stati o di transizioni di stato e normalizzando a uno, le probabilità a posteriori sia dei bit d'informazione sia dei bit di codice (in qualche caso occorrono anche queste).

Supponendo di partire dallo stato di tutti zeri, e di tornarvi alla fine della trasmissione, le variabili $\alpha_0(S_0)$ e $\beta_L(S_L)$ agli estremi del traliccio sono inizializzate a 1 per lo stato di tutti zeri e a 0 per tutti gli altri stati. Se si valuta la probabilità d'errore nelle diverse sezioni del traliccio si trova che i bit iniziali e finali sono favoriti rispetto ai centrali, grazie alla conoscenza dei valori iniziali delle variabili α e β. L'effetto delle inizializzazioni diventa trascurabile dopo un numero di passi pari ad alcune volte la *constraint length* del codice.

3.5 Codici convoluzionali *tail biting*

Si può evitare la terminazione dei codici convoluzionali, e il relativo costo in termini di bit nulli aggiunti in coda, con la tecnica *tail biting*: si parte da uno stato generico, anziché quello di tutti zeri, e si impone di terminare nello *stesso* stato. In tal modo si ottengono b^L percorsi distinti nel traliccio, ovvero b^L parole di codice, in L passi anziché $L + K - b$. Non vi sono quindi i $K - b$ zeri di terminazione, e si risparmia il tempo e l'energia per la trasmissione dei corrispondenti bit di codice. Ovviamente i bit d'informazione iniziali e finali perdono la loro migliore protezione nei riguardi del rumore, mancando l'informazione a priori sullo stato agli estremi del traliccio.

Nel caso dei codici non recursivi imporre che stato iniziale e finale coincidano è molto semplice: poichè lo stato finale, dopo L passi, dipende solo dagli ultimi $K - b$ bit d'informazione basta calcolare tale stato e far partire la codifica da questo. Nel caso dei codici recursivi il calcolo dello stato iniziale e finale è un po' più complesso (e vi sono valori di L che non ammettono soluzione).

L'algoritmo BCJR applicato ai codici *tail biting* non consente di imporre che stato iniziale e finale coincidano[8]. Si aggira il problema replicando un numero sufficiente degli ultimi campioni ricevuti in testa al blocco e dei primi in coda (e inizializzando α e β in modo arbitrario, ad esempio con tutti i valori uguali). A valle della decodifica si prende solo la sezione centrale (L passi) del risultato.

3.6 Prestazioni dei codici convoluzionali

Una valutazione approssimata delle prestazioni dei codici convoluzionali può essere ottenuta mediante lo *union bound*. Se non si limita a priori la durata della trasmissione è evidente che prima o poi qualche errore verrà commesso, e quindi la probabilità d'errore è pari a uno (risultato assolutamente inespressivo sulla qualità del collegamento). Occorre definire in modo conveniente *eventi errore* di cui sia facile valutare la probabilità. In particolare si trova conveniente calcolare la probabilità che ad un generico istante il percorso scelto dal ricevitore *inizi a divergere* da quello corretto, cioè la frequenza con cui il ricevitore devia dalla retta via. Un evento errore è concluso, per definizione, quando si ricongiunge *per la prima volta* al percorso corretto. Si noti che se si volesse invece valutare la probabilità che ad un generico istante la transizione di stato decisa sia errata si dovrebbe tener conto anche degli eventi errore già in corso.

Occorre enumerare i possibili eventi errore, e la distanza geometrica (proporzionale a quella di Hamming) tra il segnale corretto e ciascun concorrente. La linearità del codice consente di assumere che sia stata trasmessa una particolare sequenza, ad esempio quella di tutti zeri. Un evento errore deve terminare nello stato $0 \ldots 0$ e quindi non può avere lunghezza minore di K, mentre non c'è limite alla lunghezza massima. Per codici semplici, come il quattro stati di cui si è visto il traliccio, l'ispezione si può fare manualmente. Ad esempio si individuano in Fig. 3.3 un percorso errato di lunghezza 3 con distanza di Hamming pari a 5, due con distanza pari a 6 e lunghezza rispettivamente 4 e 5, quattro con distanza 7 e lunghezze comprese tra 5 e 7, e così via. Ricordando che $E_s = E_b R$, si ha che la probabilità dell'inizio di un evento errore è maggiorata dallo *union bound*

$$\begin{aligned} P(E) &\leq Q\left(\sqrt{\frac{2E_b}{N_0}5R}\right) + 2Q\left(\sqrt{\frac{2E_b}{N_0}6R}\right) + 4Q\left(\sqrt{\frac{2E_b}{N_0}7R}\right) + \ldots = \\ &= \sum_d a(d) Q\left(\sqrt{\frac{2E_b}{N_0}dR}\right) \end{aligned} \tag{3.8}$$

dove $a(d)$ è il numero di eventi errore a distanza d. Ad alto rapporto segnale-rumore è particolarmente importante la distanza minima del codice. Questa, in generale, non corrisponde all'evento errore di lunghezza minore e va quindi ricercata senza porre limiti a

[8] non basta porre i valori iniziali degli α e dei β uguali per tutti gli stati (stati iniziali e finali equiprobabili); ciò non equivale a garantire l'*uguaglianza* di stato iniziale e finale

tale lunghezza. Tale distanza è detta *distanza libera* (*free distance*) ed indicata con d_f. Il guadagno asintotico del codice rispetto alla trasmissione binaria antipodale non codificata, ignorando i coefficienti moltiplicativi all'esterno della funzione $Q(\cdot)$, è dato da

$$G = d_f R \tag{3.9}$$

ed è pari a 5/2 (4 dB) nel caso in esame.

Se si vuol valutare la probabilità che i bit d'informazione siano errati occorre pesare la probabilità di ogni evento errore con il numero di bit d'informazione errati. Ad esempio si vede che con l'evento errore a distanza minima si sbaglia un solo bit d'informazione (i bit d'informazione decodificati sono 100 anziché 000); con ciascuno dei due a distanza 6 si hanno due bit errati, e così via. La probabilità d'errore sui bit è maggiorata da

$$\begin{aligned} P_b(E) &\le Q\left(\sqrt{\frac{2E_b}{N_0}5R}\right) + 2Q\left(\sqrt{\frac{2E_b}{N_0}6R}\right) + 2Q\left(\sqrt{\frac{2E_b}{N_0}6R}\right) + \ldots = \\ &= \sum_d \sum_i i\, n(d,i) Q\left(\sqrt{\frac{2E_b}{N_0}dR}\right) = \sum_d w(d) Q\left(\sqrt{\frac{2E_b}{N_0}dR}\right) \end{aligned} \tag{3.10}$$

dove $n(d,i)$ è il numero di eventi errore a distanza d e con i bit d'informazione errati, e $w(d) = \sum i\, n(d,i)$ è il numero complessivo di bit d'informazione errati negli eventi errori aventi distanza d. La formula tiene implicitamente conto non solo degli eventi errore che iniziano ad un generico istante, ma anche di quelli già in corso. Infatti la frequenza con cui *inizia* un evento errore viene moltiplicata per il numero *complessivo* di bit d'informazione errati che esso produce, e non solo per quello prodotto nella prima transizione di stato. Se il numeratore b del *rate* del codice è diverso da uno occorre anche tener conto del numero di bit trasmessi per ciascuna transizione di stato e si ha

$$P_b(E) \le \frac{1}{b} \sum_d w(d) Q\left(\sqrt{\frac{2E_b}{N_0}dR}\right) \tag{3.11}$$

Il lettore non del tutto convinto da tali formule (che effettivamente la prima volta danno da pensare) immagini un evento errore con probabilità P, che quindi abbia inizio mediamente ogni $1/P$ passi nel traliccio, e che provochi w bit d'informazione errati. In L passi, con L molto grande, si trasmettono Lb bit d'informazione e se ne sbagliano $w\,LP$, in media. Il contributo dell'evento errore a $P_b(E)$ è quindi wP/b. La maggiorazione (3.11) non è altro che la somma estesa a tutti i possibili eventi errore.

Una formula analoga vale per i codici *punctured*, con l'avvertenza di mediare la probabilità degli eventi errore su più transizioni di stato.

Le Tab. 3.1 e 3.2 danno un'idea dei valori di distanza d_f ottenibili con codici convoluzionali con *rate* $R = 1/2$ e $1/3$. Sono indicati anche il numero complessivo $w(d_f)$ di bit d'informazione errati negli eventi errore a distanza minima, per valutare almeno il termine

K	numero di stati	d_f	$w(d_f)$	generatori
3	4	5	1	7,5
4	8	6	2	17,15
5	16	7	4	35,23
6	32	8	2	75,53
7	64	10	36	171,133
8	128	10	2	371,247
9	256	12	33	753,561

Tabella 3.1: Codici convoluzionali con *rate* $R = 1/2$

K	numero di stati	d_f	$w(d_f)$	generatori
3	4	8	3	7,7,5
4	8	10	6	17,15,13
5	16	12	12	37,33,25
6	32	13	1	75,53,47
7	64	15	7	171,165,133
8	128	16	1	367,331,225

Tabella 3.2: Codici convoluzionali con *rate* $R = 1/3$

dominante della 3.11, e la configurazione dei sommatori, in ottale. Si noti che il codice con $R = 1/2$ e 64 stati, utilizzato molto spesso, ha un guadagno asintotico di ben 7 dB.

Tra i codici *punctured*, limitandosi a quelli derivati dall'ottimo codice con *rate* $R = 1/2$ a 64 stati, si trovano ad esempio quelli con *rate* $R = 2/3$, $3/4$ e $7/8$ che hanno $d_f = 6$, 5 e 3, rispettivamente. Anche l'ultimo dà un guadagno asintotico degno di nota (4.2 dB).

L'enumerazione degli eventi errore, facile per l'esempio a quattro stati, diventa faticosa nei casi veramente interessanti e deve essere meccanizzata. Ad esempio esplorando in modo esaustivo il traliccio del codice con $R = 1/2$ e 64 stati[9] si trovano 11 eventi errore a distanza 10 con lunghezza compresa tra 7 e 16, che contribuiscono un totale di 36 bit d'informazione errati, 38 a distanza 12 (lunghezza tra 9 e 24; 211 bit errati), 193 a distanza 14 (lunghezza tra 10 e 28; 1404 bit errati) e così via.

Le Fig. 3.7 e 3.8 presentano le prestazioni di codici convoluzionali con *rate* $R = 1/2$ e $2/3$, valutate mediante lo *union bound*. Per basso rapporto segnale-rumore lo *union bound* è molto largo, e quindi inutilizzabile, e la serie può addirittura divergere.

[9]gli eventi errore sono infiniti; ovviamente si scartano quelli che già ad una certa profondità nel traliccio superano la massima distanza che interessa

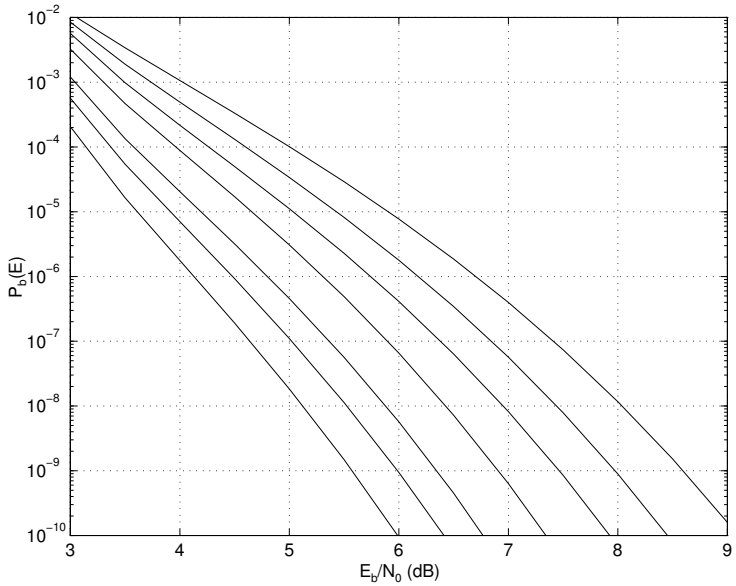

Figura 3.7: Maggiorazione della probabilità d'errore dei codici convoluzionali con *rate* 1/2 da 4 a 256 stati (da destra a sinistra)

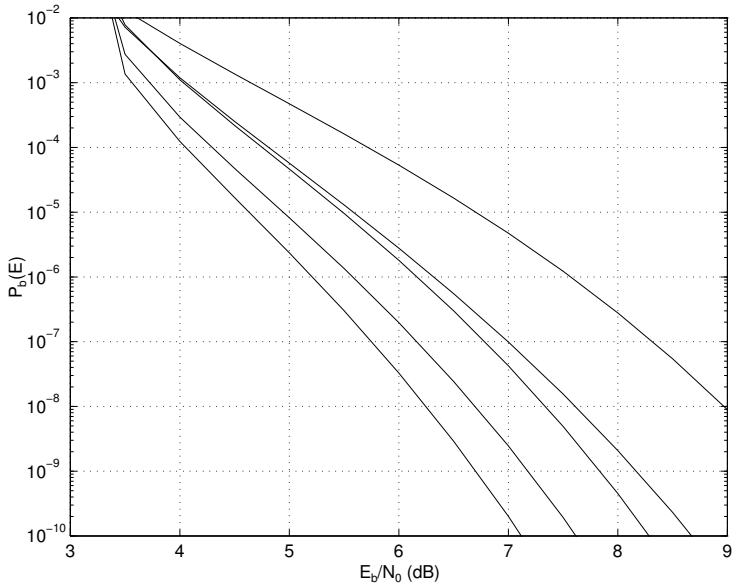

Figura 3.8: Maggiorazione della probabilità d'errore dei codici convoluzionali con *rate* 2/3 da 4 a 64 stati (da destra a sinistra)

Capitolo 4

Algebra dei campi finiti

4.1 Campi finiti (o di *Galois*)

Con la notazione GF(q) si indica un campo finito, detto anche campo di *Galois* (*Galois field*), con q elementi. Sui q elementi del campo sono definite due operazioni (*addizione* e *moltiplicazione*) che producono comunque elementi del campo (proprietà di chiusura). Si hanno inoltre le seguenti altre proprietà:

- esistono due elementi (indicati rispettivamente con 0 e 1) tali che, per ogni elemento a del campo,

$$a + 0 = a \tag{4.1}$$

$$a \cdot 1 = a \tag{4.2}$$

- per ogni a esiste l'opposto (-a), e per ogni $a \neq 0$ esiste l'inverso (a^{-1})

$$a + (-a) = 0 \tag{4.3}$$

$$a \cdot a^{-1} = 1 \tag{4.4}$$

  In altri termini, si possono definire anche la *sottrazione*

$$a - b = a + (-b) \tag{4.5}$$

  e la *divisione*

$$a/b = a \cdot b^{-1} \tag{4.6}$$

- valgono inoltre le usuali proprietà commutativa e associativa

$$a + b = b + a \tag{4.7}$$

$$a \cdot b = b \cdot a \tag{4.8}$$

$$a + (b + c) = (a + b) + c \tag{4.9}$$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \tag{4.10}$$

- vale infine l'usuale proprietà distributiva

$$(a + b) \cdot c = a \cdot c + b \cdot c \tag{4.11}$$

In sintesi *valgono tutte le normali regole dell'algebra, ma il numero di elementi su cui si opera è finito.* Naturalmente gli elementi del campo e le operazioni di *addizione* e *moltiplicazione* non sono quelli usuali: basta pensare a $1 + 1 + 1 + 1 + \cdots$, che non può produrre numeri sempre nuovi, essendo finiti gli elementi del campo.

Tutte le proprietà dell'algebra usuale restano valide, e si possono applicare senza remore (naturalmente eseguendo le operazioni di somma e prodotto secondo le regole del campo). In particolare, per fare solo qualche esempio:

- se $a \cdot b = 0$ almeno uno dei due fattori è nullo
- $a \cdot b = a \cdot c$ (con $a \neq 0$) implica $b = c$; infatti si ottiene $a \cdot (b - c) = 0$ e quindi $b = c$; si può quindi *semplificare* cancellando il fattore $a \neq 0$
- l'equazione $a \cdot x = b$ si risolve moltiplicando per l'inverso di a: $x = a^{-1} \cdot b = b \cdot a^{-1}$
- sistemi di equazioni lineari si possono risolvere con tutte le tecniche usuali (anche con la regola di *Cramer*; è banale definire il determinante di una matrice)
- anche un'equazione di secondo grado si risolve con le tecniche (e formule) usuali, ma alla condizione (spesso non verificata) che sia possibile completare il quadrato!

Nota: nel seguito l'operatore di moltiplicazione ($\cdot$) verrà sottinteso quando possibile, come nell'algebra usuale.

4.1.1 Campi finiti con un numero primo di elementi

Esistono campi finiti? e per quali valori di q?

$q = 2$

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

$q = 3$

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

$q = 5$

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\cdot$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Addizione e *moltiplicazione* differiscono dalle usuali per il fatto che il risultato è ridotto modulo q, cioè è il *resto* della divisione per q.

Si dimostra che operando in questo modo si ottiene un campo se q è un numero primo[1].

4.1.2 Campi finiti con un numero di elementi non primo

I campi con un numero di elementi q primo sono poco comuni nelle applicazioni ai codici correttori. Se q non è primo, le operazioni modulo q non definiscono un campo. Si consideri ad esempio $q = 4$, e si provi ad usare l'algebra modulo q:

$$2 \cdot 2 = 0 \mod 4 \tag{4.12}$$

e moltiplicando per 2^{-1}

$$(2 \cdot 2) \cdot 2^{-1} = 0 \cdot 2^{-1} = 0 \tag{4.13}$$

ma, per la proprietà associativa, si deve ottenere lo stesso risultato da

$$2 \cdot (2 \cdot 2^{-1}) = 2 \cdot 1 = 2 \neq 0 \tag{4.14}$$

Il problema deriva dal fatto che nell'algebra modulo 4 non esiste 2^{-1}, inverso di 2! Inoltre $a \cdot b = 0$ non implica $a = 0$ oppure $b = 0$, come invece deve essere in un campo.

Tuttavia GF(4) esiste, con *altre regole* per l'*addizione* e la *moltiplicazione*:

[1] l'unica proprietà che richiede una verifica non banale è l'esistenza e unicità dell'inverso di $a \neq 0$. Siano b_i gli elementi non nulli ($b_i = 1, 2, \ldots, q-1$) e sia $b_i a = b_j a \mod q$, e quindi $(b_i - b_j)a$ un multiplo di q. Essendo $a < q$, $|b_i - b_j| < q$ e q primo deve essere $b_i = b_j$: dunque l'inverso di a, se esiste, è unico. Avendo mostrato che i $q-1$ prodotti $b_i a$ sono distinti, uno di questi deve dare 1 e il b_i corrispondente è a^{-1}

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Esaminando le prime due righe e colonne delle tabelle si può notare che GF(2) è un sottoinsieme di GF(4). Si dice anche che GF(4) è un'estensione di GF(2). Si noti che invece GF(2) non è un sottoinsieme di GF(3) o di GF(5).

In generale si può dimostrare che GF(q) esiste se q è un numero primo o è potenza di un numero primo. Non esistono altri campi finiti (ad esempio non esiste un campo con 6 elementi).

Per un campo finito generico si usa la notazione GF(q), dove q è il numero di elementi. Il campo con q^m elementi ha sempre GF(q) come sottoinsieme, ed è quindi estensione di GF(q). Si usa quindi anche la notazione GF(q^m), soprattutto se si vuol sottolineare che si considera il campo come estensione di GF(q).

Ad esempio il campo con 64 elementi viene indicato genericamente come GF(64), ma anche come GF(8^2), GF(4^3) o GF(2^6) se lo si considera estensione rispettivamente di GF(8), GF(4) o GF(2). In ogni caso q è primo o potenza di un numero primo. In enunciati in cui si voglia mettere in evidenza che q è primo si usano le notazioni GF(p) e GF(p^m).

4.1.3 Rappresentazione degli elementi dei campi finiti

Rappresentazione conveniente per gli elementi di GF(q^m), estensione di GF(q): m-ple di elementi di GF(q); ovvero polinomi di grado $m-1$ con coefficienti in GF(q):

$$a_{m-1}a_{m-2}\cdots a_1\ a_0 \tag{4.15}$$

ovvero

$$a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2}\cdots + a_1\alpha + a_0 \tag{4.16}$$

La variabile α ha il ruolo di individuare la posizione nella m-pla. Ad esempio è indifferente scrivere $\alpha^3+\alpha+1$ oppure $1+\alpha+\alpha^3$. Tuttavia si vedrà presto che le potenze di α hanno anche il significato di elementi del campo GF(q^m): il generico elemento del campo è quindi espresso come combinazione lineare, con coefficienti in GF(q), della *base* $1, \alpha, \ldots, \alpha^{m-2}, \alpha^{m-1}$.

In luogo di α è molto usata una variabile generica, ad esempio x. Tuttavia ciò può dar luogo a qualche perplessità quando, come nel seguito, si scrivono espressioni come $(x-\alpha)$.

Regola per l'*addizione*: somma termine a termine con le regole di GF(q) (ad esempio modulo q, se q è primo). Ad esempio in GF(2^m) si ha

$$(\alpha^2+\alpha+1)+(\alpha^2+1)=\alpha \tag{4.17}$$

Si noti che quindi in GF(2^m) $a + a = 0$, per ogni a: addizione e sottrazione coincidono[2].

La regola per la *moltiplicazione* è più complessa. Anzitutto si individua un polinomio $p(x)$ di grado m irriducibile, cioè non fattorizzabile come prodotto di polinomi con coefficienti in GF(q), detto *polinomio generatore del campo*[3]. Si esegue poi il prodotto dei polinomi modulo $p(\alpha)$, cioè dopo la moltiplicazione si calcola il resto della divisione[4] per $p(\alpha)$.

Esempio: $p(x) = x^3+x+1$ ha coefficienti in GF(2) ed è irriducibile. Infatti non è divisibile[5] né per x né per $x + 1$.

$$\begin{aligned}&(\alpha^2 + \alpha + 1) \cdot (\alpha^2 + 1) = (\alpha^4 + \alpha^3 + \alpha^2 + \alpha^2 + \alpha + 1) \mod p(\alpha) = \\ &= (\alpha^4 + \alpha^3 + \alpha + 1) \mod p(\alpha) = \alpha^2 + \alpha\end{aligned} \tag{4.18}$$

Nota: il resto modulo $p(\alpha)$ in GF(q^m) è l'equivalente del resto modulo p in GF(p). I polinomi irriducibili (non fattorizzabili) hanno proprietà analoghe ai numeri primi.

Un caso particolare di estensione di GF(q) ricorda cose familiari. Se $x^2 + 1$ è irriducibile in GF(q), gli elementi di GF(q^2) sono coppie (a, b) di elementi di GF(q), ovvero polinomi $a + b\alpha$.

Le regole per le operazioni sono:

$$(a + b\alpha) \pm (c + d\alpha) = (a \pm c) + (b \pm d)\alpha \tag{4.19}$$

$$(a+b\alpha)(c+d\alpha) = ac+(ad+bc)\alpha+bd\alpha^2 \mod \alpha^2+1 = (ac-bd)+(ad+bc)\alpha \tag{4.20}$$

e da queste, con qualche calcolo, si ottiene anche

$$(a + b\alpha)^{-1} = \frac{a}{a^2 + b^2} - \frac{b}{a^2 + b^2}\alpha \tag{4.21}$$

Si tratta delle stesse regole dei numeri complessi! Effettivamente $a + b\alpha$ si comporta come $a + jb$ perché $\alpha^2 + 1 = 0$ equivale a $j^2 = -1$.

Il campo dei numeri complessi è un'estensione del campo dei numeri reali[6] perchè $x^2+1 = 0$ non ha radici reali (è irriducibile nel campo dei numeri reali).

---

[2]una conseguenza inattesa è che in GF(2^m) non è possibile risolvere nel modo tradizionale la generica equazione di secondo grado $ax^2 + bx + c = 0$. Infatti la formula risolvente avrebbe a denominatore $2a$, cioè $a + a$, ed è facile verificare che si annulla anche il numeratore. Si vedrà nel seguito che se $b \neq 0$ non è possibile trovare un elemento d tale che $ax^2 + bx + d$ sia un quadrato perfetto (proprietà cruciale per la soluzione tradizionale dell'equazione di secondo grado!)

[3]per uniformità si dovrebbe indicare il polinomio generatore del campo con $p(\alpha)$, ma è molto più comune usare la variabile x

[4]volendo dimostrare anche per i polinomi l'esistenza e unicità dell'inverso basta ripetere quanto già visto, sostituendo ai *valori* di a, b_i e b_j il *grado* del corrispondente polinomio

[5]è inutile provare a dividere anche per $x^2 + 1$ e $x^2 + x + 1$: eventualmente si otterrebbe un polinomio di primo grado, già escluso; per analogia, si ricordi che volendo verificare se un numero q è primo basta provare a dividere per gli interi $\leq \sqrt{q}$

[6]in questo caso si tratta di campi con un numero *infinito* di elementi

Nota: si osservi che x^2+1 non è *mai* irriducibile in GF(2^m), che è il caso più comune nelle applicazioni. Infatti $(x+1)^2 = x^2+x+x+1 = x^2+1$. In GF(3) x^2+1 è invece irriducibile, mentre in GF(5) si ha $x^2+1 = (x+2)(x+3)$.

In *hardware* l'addizione di due elementi di GF(2^m) consiste nella semplice somma bit a bit (EXOR), e può essere eseguita serialmente o in parallelo. La moltiplicazione di un elemento del campo, ad esempio $a_2\alpha^2 + a_1\alpha + a_0$ in GF(2^3), per una costante prefissata, ad esempio α^2, può essere ottenuta come

$$\begin{aligned}&(a_2\alpha^2 + a_1\alpha + a_0)\alpha^2 = a_2\alpha^4 + a_1\alpha^3 + a_0\alpha^2 \mod \alpha^3 + \alpha + 1 = \\ &= a_2(\alpha^2 + \alpha) + a_1(\alpha + 1) + a_0\alpha^2 = (a_2 + a_0)\alpha^2 + (a_2 + a_1)\alpha + a_1\end{aligned} \tag{4.22}$$

e richiede un collegamento diretto e due sommatori EXOR.

Esistono moltissime strutture di moltiplicatori di due elementi del campo. Per campi di piccola dimensione si può utilizzare una ROM che contenga tutti i casi (occorrono $2m$ bit di indirizzo). Può anche risultare conveniente dare la mappa di verità da realizzare ad un circuito ottimizzatore di reti combinatorie.

4.1.4 Elementi primitivi e rappresentazione esponenziale

Se si considerano, ad esempio in GF(5), le potenze dell'elemento $\alpha = 2$ si ottiene $2^2 = 4$, $2^3 = 3$, $2^4 = 1$ e le successive potenze si ripetono. Si sono ottenuti tutti gli elementi non nulli del campo. L'elemento $\alpha = 2$ è detto *primitivo*. Analogamente si verifica che 3 è un elemento primitivo, mentre 4 non lo è. Infatti già la seconda potenza dà 1.

In GF(8) se si considerano le potenze dell'elemento α si ottengono[7]

$$\begin{aligned}&\alpha^2 \\ &\alpha^3 = \alpha + 1 \\ &\alpha^4 = \alpha^2 + \alpha \\ &\alpha^5 = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1 \\ &\alpha^6 = \alpha^3 + \alpha^2 + \alpha = \alpha + 1 + \alpha^2 + \alpha = \alpha^2 + 1 \\ &\alpha^7 = \alpha^3 + \alpha = 1 \\ &\alpha^8 = \alpha\end{aligned}$$

e così via. Tutti gli elementi non nulli del campo sono rappresentabili come potenze di α, e quindi α è primitivo.

In GF(8) si verifica facilmente che tutti gli elementi del campo (eccetto 0 e 1) sono primitivi. Questo non sempre avviene, come visto sopra.

[7] si faccia attenzione a non confondere i due α di GF(5) e GF(8), che non hanno nulla in comune: α è un simbolo generico per indicare l'elemento primitivo scelto come riferimento

Tuttavia si dimostra che esiste sempre *almeno un elemento primitivo* del campo (non necessariamente α). Se α è primitivo il polinomio $p(x)$ che genera il campo è detto primitivo. Esiste sempre almeno un polinomio primitivo, e non si ha alcun vantaggio ad usare polinomi non primitivi. In GF(q^m) con $q \neq 2$ conviene prendere un polinomio primitivo con coefficiente di grado massimo pari a 1.

Scegliendo $p(x)$ primitivo, tutti gli elementi non nulli del campo possono essere rappresentati non solo con m-ple di elementi di GF(q), ma anche come potenze dell'elemento primitivo α del campo. Talvolta si pone (ma è convenzionale!) $0 = \alpha^{-\infty}$.

Possibili rappresentazioni di GF(8) sono quindi:

000	0	$(\alpha^{-\infty})$
001	1	α^0
010	α	α^1
100	α^2	α^2
011	$\alpha + 1$	α^3
110	$\alpha^2 + \alpha$	α^4
111	$\alpha^2 + \alpha + 1$	α^5
101	$\alpha^2 + 1$	α^6

La prima rappresentazione (equivalente alla seconda[8]) è comoda per le addizioni, l'ultima per le moltiplicazioni: si sommano gli esponenti e si riducono modulo 7, poiché $\alpha^7 = 1$. Ad esempio $\alpha^4\alpha^6 = \alpha^{10} = \alpha^3$.

4.1.5 Calcolo di espressioni algebriche

Le tabelle di conversione tra le due rappresentazioni di GF(8) possono essere usate per calcolare espressioni algebriche come

$$(\alpha^2 + \alpha^5)\alpha + (1 + \alpha^3)\alpha = \alpha^3\alpha + \alpha\alpha = \alpha^4 + \alpha^2 = \alpha = 010$$

dove per calcolare $\alpha^2 + \alpha^5$ si è usata la rappresentazione $\alpha^2 = 100$, $\alpha^5 = 111$; quindi $\alpha^2 + \alpha^5 = 011 = \alpha^3$; e così via, con varie conversioni.

Nota: potremmo chiamare le conversioni *logaritmo* e *esponenziale* (o *antilogaritmo*). Le due tabelle, realizzate come memorie, hanno m bit di indirizzo.

Volendo usare solo la rappresentazione esponenziale può essere conveniente definire il *logaritmo di Zech* $z(n)$:

$$1 + \alpha^n = \alpha^{z(n)} \tag{4.23}$$

Infatti volendo calcolare $\alpha^k + \alpha^j$, con $k \leq j$, si ha

$$\alpha^k + \alpha^j = \alpha^k(1 + \alpha^{j-k}) = \alpha^{k+z(j-k)} \tag{4.24}$$

[8] attenzione all'ordine dei bit, che è convenzionale! qui i coefficienti binari sono dati per potenze *decrescenti* di α

dove infine l'esponente $k + z(j - k)$ verrà ridotto modulo 7. In GF(8) si ottiene facilmente che a n =-$\infty, 0, 1, 2, 3, 4, 5, 6$ corrispondono[9] $z(n)$ = 0,-$\infty, 3, 6, 1, 5, 4, 2$.

Ad esempio per ottenere $\alpha^3 + \alpha^5 = \alpha^2$ (considerando solo gli esponenti) si ha

$$(3, 5) \rightarrow 3 + z(2) = 3 + 6 = 9 = 2$$

Il *logaritmo di Zech* può essere comodo in *software*, cioè in un programma di calcolo, anche se oggi le enormi quantità di memoria disponibili rendono possibile *precalcolare e memorizzare* una tabella con *tutte* le possibili somme, oppure usare la notazione binaria e precalcolare tutti i possibili prodotti. In *hardware* spesso è conveniente la rappresentazione binaria. Infatti operazioni che sembrano semplici, come la lettura da una tabella, possono avere costi non modesti.

4.1.6 Polinomi primitivi generatori del campo

Esempi di polinomi primitivi, per generare GF(2^m):

m	$p(x)$
2	$x^2 + x + 1$
3	$x^3 + x + 1$
4	$x^4 + x + 1$
5	$x^5 + x^2 + 1$
6	$x^6 + x + 1$
7	$x^7 + x^3 + 1$
8	$x^8 + x^4 + x^3 + x^2 + 1$
9	$x^9 + x^4 + 1$
10	$x^{10} + x^3 + 1$
11	$x^{11} + x^2 + 1$
12	$x^{12} + x^6 + x^4 + x + 1$
...	...

Per $m = 3$ anche $x^3 + x^2 + 1$ è primitivo, e per m grande esistono numerosi polinomi primitivi; in genere si preferisce usare polinomi con il minor numero possibile di coefficienti non nulli (tre, quando possibile)[10].

Volendo ad esempio generare gli elementi non nulli di GF(16)=GF(2^4) come potenze di α, si moltiplica successivamente per α e si riduce modulo $p(\alpha) = \alpha^4 + \alpha + 1$, ottenendo:

[9] -∞ sarà rappresentato con l'intero negativo più comodo, o con un intero positivo maggiore di 6

[10] in GF(2) polinomi con un numero pari di coefficienti uguali ad 1 sono sempre riducibili e quindi non primitivi; infatti (dal teorema del resto della divisione) se $p(1) = 0$ si può dividere $p(x)$ per $(x + 1)$. Si è invece già visto che in GF(3) $x^2 + 1$ è irriducibile

α	0010
α^2	0100
α^3	1000
$\alpha^4 = \alpha + 1$	0011
$\alpha^5 = \alpha^2 + \alpha$	0110
$\alpha^6 = \alpha^3 + \alpha^2$	1100
$\alpha^7 = \alpha^3 + \alpha + 1$	1011
$\alpha^8 = \alpha^2 + 1$	0101
$\alpha^9 = \alpha^3 + \alpha$	1010
$\alpha^{10} = \alpha^2 + \alpha + 1$	0111
$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$	1110
$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$	1111
$\alpha^{13} = \alpha^3 + \alpha^2 + 1$	1101
$\alpha^{14} = \alpha^3 + 1$	1001
$\alpha^{15} = 1$	0001
...	

Si osservi che calcolare le potenze di α riducendo il risultato modulo $p(\alpha)$ equivale ad imporre $\alpha^4 + \alpha + 1 = 0$. Quindi α è radice del polinomio $p(x)$ generatore del campo. Nel seguito si vedrà come determinare quali sono le altre radici di $p(x)$.

Nota: α non è l'unico elemento primitivo di GF(2^4); ad esempio calcolando 15 potenze consecutive di α^2 ($\alpha^2, \alpha^4, \alpha^6, \ldots$) oppure di α^7 ($\alpha^7, \alpha^{14}, \alpha^{21} = \alpha^6, \alpha^{28} = \alpha^{13}, \ldots$) si ritrovano tutti i 15 elementi non nulli, e dunque anche α^2 e α^7 sono primitivi.

Si osservi invece che α^3 non è primitivo: infatti $(\alpha^3)^5 = \alpha^{15} = 1$ e quindi le potenze di α^3 si ripetono dopo la quinta. Si dice che α^3 ha *ordine* 5. Analogamente non sono primitivi, ed hanno ordine 5, $\alpha^6, \alpha^9, \alpha^{12}$; α^5 e α^{10} non sono primitivi ed hanno ordine 3.

Con semplici ragionamenti si deduce che l'ordine di α^n è $(2^m - 1)/\text{MCD}(2^m - 1, n)$. Quindi α^n è primitivo se e solo se n e $2^m - 1$ sono numeri relativamente primi. Se $2^m - 1$ è un numero primo, come accade ad esempio per GF(8) e GF(32), tutti gli elementi (esclusi 0 e 1) sono primitivi.

Si osservi che GF(2) è un sottoinsieme di GF(16), che contiene solo 0 e 1. GF(16) è stato infatti ottenuto come estensione di GF(2): gli elementi di GF(16) sono polinomi di grado 3 con coefficienti binari (ovvero quaterne di elementi binari); il polinomio generatore del campo è $p(x) = x^4 + x + 1$.

Si può notare che anche GF(4) è un sottoinsieme di GF(16): i suoi elementi sono 0, 1, $\beta = \alpha^5$ e $\beta^2 = \alpha^{10}$. Basta infatti verificare che le operazioni di somma e prodotto tra questi elementi restituiscono uno dei quattro elementi. Quindi GF(16) può anche essere considerato estensione di GF(4): gli elementi di GF(16) sono rappresentabili come polinomi di primo grado con coefficienti 0, 1, β o β^2, ovvero con coppie di elementi di GF(4). Con un po' di pazienza si può *verificare* che il polinomio generatore del campo è $p(x) = x^2 + x + \beta$. Si vedrà più avanti come si possa *determinare* questo polinomio $p(x)$.

Esistenza di almeno un elemento primitivo

In GF(q) è evidente che l'ordine n di un generico elemento α non può essere maggiore di $q-1$. Inoltre $\alpha^m = 1$ se e solo se m è multiplo di n.

Siano α e β elementi di ordine n e m, con n ed m relativamente primi, e si voglia determinare l'ordine di $\alpha\beta$. Basta osservare che $(\alpha\beta)^k = 1$ implica $(\alpha\beta)^{km} = 1$ ovvero, poiché $\beta^{km} = 1$, $\alpha^{km} = 1$. L'esponente km deve essere multiplo di n e quindi k è multiplo di n. In modo del tutto analogo da $(\alpha\beta)^{kn} = 1$ si ottiene che k è multiplo di m. Quindi k è multiplo di nm. Poiché $(\alpha\beta)^{nm} = \alpha^{nm}\beta^{nm} = 1$, l'ordine di $\alpha\beta$ è mn.

Infine sia n l'ordine massimo degli elementi non nulli del campo GF(q), e sia α un elemento di ordine n. Si supponga che esista un elemento β non nullo di ordine d, con d non divisore di n. Le scomposizioni in fattori primi di n e d contengono (almeno) un termine p^a e p^b, con $b > a \geq 0$. L'elemento $\alpha' = \alpha^{p^a}$ ha ordine n/p^a che non contiene il fattore primo p. L'elemento $\beta' = \beta^{d/p^b}$ ha ordine p^b che non contiene altri fattori primi. Quindi n/p^a e p^b sono relativamente primi e $\alpha'\beta'$ ha ordine $np^{b-a} > n$ maggiore del massimo. Ne deriva che *tutti* gli elementi β_i non nulli del campo hanno ordine divisore di n, e soddisfano quindi la condizione $\beta_i^n = 1$, cioè sono radici dell'equazione $x^n - 1 = 0$. Quindi $x^n - 1$ contiene come fattori *tutti* i $q-1$ termini $(x - \beta_i)$ e ha grado non minore di $q-1$. Ne deriva che $n \geq q-1$, ovvero $n = q-1$.

4.1.7 Sequenze pseudocasuali

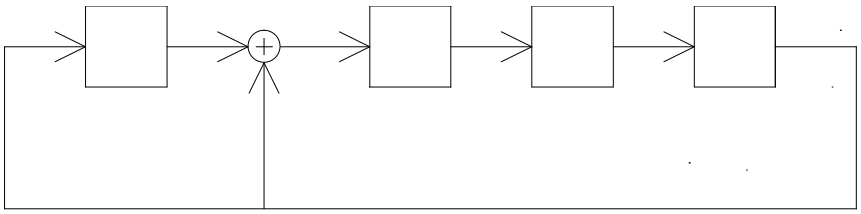

Figura 4.1: Generatore delle potenze di α in GF(16), ovvero di sequenze pseudocasuali di lunghezza 15

Dalla definizione delle successive potenze dell'elemento primitivo α del campo GF(16), ovvero moltiplicazione per α e riduzione modulo $p(\alpha) = \alpha^4 + \alpha + 1$, si individua il semplice circuito di Fig. 4.1 che le genera successivamente. I quattro registri contengono *da destra a sinistra* i coefficienti dei termini di grado *decrescente* da 3 a 0. Si riconoscono nei collegamenti in retroazione i coefficienti del polinomio $p(x)$. In generale, preso un polinomio primitivo in grado di generare GF(2^m), il circuito che genera successivamente le potenze dell'elemento primitivo α ha m registri e connessioni in retroazione in corrispondenza dei coefficienti non nulli di $p(x)$. Se si inizializzano i registri con una qualunque potenza di α, cioè con una generica m-pla non nulla, le potenze di α si ripetono con periodo $2^m - 1$.

Se si considera l'evoluzione nel tempo del contenuto di *uno qualsiasi* degli m registri si ha una sequenza con le seguenti proprietà:

- in un periodo di lunghezza 2^m-1 si hanno 2^{m-1} uni e $2^{m-1}-1$ zeri; infatti le potenze di α sono tutti gli elementi del campo, eccetto quello nullo
- la sequenza generata ha come periodo 2^m-1, e non un suo divisore[11], perché in tal caso non sarebbe rispettato il numero di uni e zeri in un ciclo completo di lunghezza 2^m-1
- il periodo 2^m-1 è il massimo possibile per un circuito con m registri, essendo 2^m-1 le configurazioni non nulle distinte
- la somma della sequenza e di una sua replica traslata (purché non di un multiplo di 2^m-1) dà la stessa sequenza con *fase* diversa. Basta infatti considerare due circuiti come quelli di Fig. 4.1 inizializzati con potenze *diverse* α^i e α^j, che producono quindi la stessa sequenza sfasata. Per linearità, la somma delle due sequenze può essere ottenuta inizializzando un *unico* registro con la somma $\alpha^i+\alpha^j$: si ottiene quindi la stessa sequenza, con fase ancora diversa
- all'uscita di tutti i registri si ha la *stessa* sequenza, con fasi diverse[12]. Esaminando il circuito *da destra a sinistra* la proprietà è evidente per i registri collegati direttamente. Dove ci sono sommatori l'ingresso *da sinistra* è la somma dell'ingresso *dal basso* e dell'ingresso al registro successivo: ma queste sono repliche sfasate dalla sequenza
- in un periodo le configurazioni di m bit consecutivi contenuti in un registro sono *tutte* le m-ple possibili, eccetto quella di m zeri. Infatti, detta s_k la sequenza, esaminando la Fig. 4.1 si vede che se $p(x)=x^m+\sum_{i=0}^{m-1}p_ix^i$ è il polinomio generatore del campo si ha $s_{k+m}=\sum_{i=0}^{m-1}p_is_{k+i}$. Se una stessa m-pla si ripetesse all'interno di un periodo si ripeterebbe anche s_k; se si avessero m zeri consecutivi la sequenza resterebbe definitivamente azzerata. Si noti che l'equazione che lega s_{k+m} agli s_{k+i} precedenti valori fornisce un diverso circuito, mostrato in Fig. 4.2, per generare la sequenza[13]
- se si rappresentano i livelli 1 e 0 della sequenza s_k con 1 e -1, o viceversa, l'autocorrelazione della sequenza vale 1 nell'origine e per traslazioni pari a un numero intero di periodi, e $-1/(2^m-1)$ altrimenti; per m grande la sequenza è praticamente incorrelata (bianca). Basta infatti confrontare le sequenze traslate (cioè *sommarle*, nella rappresentazione con uni e zeri) per vedere che si hanno $2^{m-1}-1$ valori coincidenti e 2^{m-1} opposti

[11] nel caso 2^m-1 non sia un numero primo

[12] nel caso particolare di un solo sommatore, come in Fig. 4.1, basterebbe partire dall'uscita del registro che precede il sommatore e procedere in senso antiorario

[13] se il polinomio $p(x)$ generatore del campo ha solo tre coefficienti non nulli, la Fig. 4.2 coincide con la Fig. 4.1, a meno di una permutazione dei registri; ma in generale la struttura che si ottiene è diversa

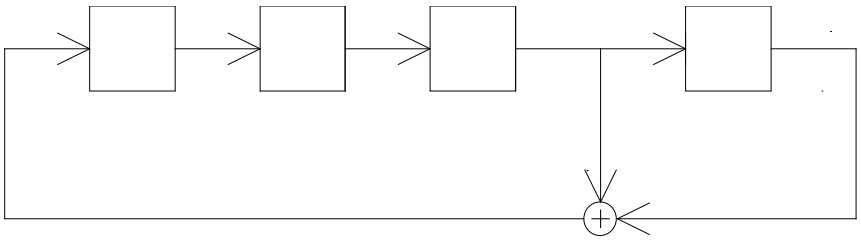

Figura 4.2: Forma alternativa del generatore di sequenze pseudocasuali di lunghezza 15

Se m è grande, come nei casi di maggior interesse pratico, la sequenza contiene zeri e uni quasi bilanciati ed appare come *casuale*; poiché la generazione *non* è casuale, la sequenza è detta *pseudocasuale* (o anche m-sequenza).

Si noti infine che nello stesso GF(2^m) polinomi primitivi diversi generano sequenze pseudocasuali diverse, anche se con proprietà analoghe. Il polinomio *reciproco* $p(x) = x^4 + x^3 + 1$ genera la sequenza con asse dei tempi ribaltato. Per $m = 5$ oltre a x^5+x^2+1 sono primitivi $x^5 + x^4 + x^3 + x^2 + 1$ e $x^5 + x^4 + x^2 + x + 1$, nonché i reciproci[14].

4.2 Proprietà specifiche dell'algebra dei campi finiti

Una proprietà fondamentale è che se α e β sono elementi generici di GF(p^m), con p primo, si ha dalla solita espansione binomiale della potenza p-esima

$$(\alpha + \beta)^p = \sum_0^p \binom{p}{i} \alpha^i \beta^{p-i} = \alpha^p + \beta^p \tag{4.25}$$

Infatti per $1 \leq i \leq p-1$ risulta[15]

$$\binom{p}{i} = \frac{p(p-1)(p-2)\cdots(p-i+1)}{1 \cdot 2 \cdot 3 \cdot i} = 0 \tag{4.26}$$

Si hanno poi le notevoli generalizzazioni:

$$(\alpha + \beta + \gamma)^p = ((\alpha + \beta) + \gamma)^p = (\alpha + \beta)^p + \gamma^p = \alpha^p + \beta^p + \gamma^p \tag{4.27}$$

$$(\alpha + \beta)^{p^2} = ((\alpha + \beta)^p)^p = (\alpha^p + \beta^p)^p = \alpha^{p^2} + \beta^{p^2} \tag{4.28}$$

e quindi, per qualsiasi numero di elementi e qualsiasi $k \geq 1$,

$$(\sum \alpha_i)^{p^k} = \sum \alpha_i^{p^k} \tag{4.29}$$

[14] vengono generate le sequenze 000010010110011111000110111010 1, 000011001001111101110001010110 1, 000011100110111111010001001010 11 e le reciproche

[15] il calcolo deve essere eseguito modulo p perché la somma di p uni dà zero! Il fattore primo p a numeratore non si cancella. Si ricordi anche l'esempio già visto in GF(2^m): $(x+1)^2 = x^2 + 1$

Se poi $\alpha \in \mathrm{GF}(p)$ è utile osservare che[16]

$$\alpha^p = \alpha^{p-1}\alpha = \alpha \tag{4.30}$$

$$\alpha^{p^2} = (\alpha^p)^p = \alpha^p = \alpha \tag{4.31}$$

e in generale, per ogni $k \geq 1$

$$\alpha^{p^k} = \alpha \tag{4.32}$$

In modo analogo si dimostra che se $\alpha \in \mathrm{GF}(q) = \mathrm{GF}(p^n)$ si ha, per ogni $k \geq 1$

$$\alpha^{q^k} = \alpha \tag{4.33}$$

Una notevole conseguenza di quanto sopra è che se β è un generico elemento di $\mathrm{GF}(p^m)$ e $f(x) = a_0 + a_1 x + \ldots$ è un generico polinomio[17] con coefficienti $a_i \in \mathrm{GF}(p)$ si ha la seguente relazione tra i valori del polinomio calcolato in β e β^p:

$$f^p(\beta) = (\sum a_i \beta^i)^p = \sum a_i^p \beta^{ip} = \sum a_i \beta^{ip} = f(\beta^p) \tag{4.34}$$

Un risultato ancora più generale, se $a_i \in \mathrm{GF}(q) = \mathrm{GF}(p^n)$ con n divisore di m in modo che a_i appartenga anche a $\mathrm{GF}(p^m)$, è

$$f^q(\beta) = (\sum a_i \beta^i)^q = \sum a_i^q \beta^{iq} = \sum a_i \beta^{iq} = f(\beta^q) \tag{4.35}$$

Nel caso particolare $f(\beta) = 0$ si ha anche $f(\beta^q) = 0$, $f(\beta^{q^2}) = 0, \ldots$ Quindi se β è una radice di un polinomio $f(x)$ con coefficienti in $\mathrm{GF}(q)$, anche β^q, $\beta^{q^2}, \ldots$ ne sono radici.

Ad esempio in $\mathrm{GF}(2^3)$ il polinomio con coefficienti binari $f(x) = x^3 + x + 1$ ha radice α (infatti $\alpha^3 + \alpha + 1 = 0$). Altre radici sono quindi α^2 e α^4 (si noti che le successive potenze si ripetono: $\alpha^8 = \alpha$, $\alpha^{16} = \alpha^2, \ldots$). Si verifica facilmente che $\alpha^6 + \alpha^2 + 1 = 0$ e $\alpha^{12} + \alpha^4 + 1 = \alpha^5 + \alpha^4 + 1 = 0$. In questo caso il polinomio $f(x)$ ha grado 3, e quindi non ha altre radici. Il polinomio $f(x)$, irriducibile in $\mathrm{GF}(2)$, è quindi fattorizzabile in $\mathrm{GF}(2^3)$:

$$x^3 + x + 1 = (x - \alpha)(x - \alpha^2)(x - \alpha^4) \tag{4.36}$$

[16] la proprietà è banale in GF(2), poiché α può valere solo 0 o 1; ma ad esempio in GF(17) si ha $\alpha^{17^k} = \alpha$ per qualsiasi α e k, e si può quindi affermare senza calcoli che 14^{289} diviso per 17 dà resto 14

[17] si faccia ben attenzione a non far confusione tra i diversi polinomi che via via si incontrano:

- polinomio $p(x)$ generatore del campo $\mathrm{GF}(q^m)$ estensione di $\mathrm{GF}(q)$, di grado m e con coefficienti in $\mathrm{GF}(q)$
- elementi del campo $\mathrm{GF}(q^m)$, rappresentabili sia come m-ple di elementi di $\mathrm{GF}(q)$ sia come polinomi di grado $m-1$ con coefficienti in $\mathrm{GF}(q)$
- polinomi generici $f(x)$, di grado qualsiasi e con coefficienti in $\mathrm{GF}(q)$ o in $\mathrm{GF}(q^m)$

Analogamente si può verificare che $x^3 + x^2 + 1$ ha come radice α^3. Le altre radici sono α^6 e $\alpha^{12} = \alpha^5$ e quindi

$$x^3 + x^2 + 1 = (x - \alpha^3)(x - \alpha^6)(x - \alpha^5) \tag{4.37}$$

Un polinomio irriducibile in GF(q), senza radici multiple, è quindi individuabile con *una sola (qualsiasi)* delle sue radici in GF(q^m).

Si usa dire che x^3+x+1 è il *polinomio minimo* di α (e naturalmente anche di α^2 e α^4): è il polinomio di grado minimo con coefficienti in GF(2) e con radice α (α^2, α^4). Analogamente $x^3 + x^2 + 1$ è il polinomio minimo di α^3, α^6 e α^5.

In GF(2^4) α (oppure α^2, α^4, α^8, $\alpha^{16} = \alpha$, ...) individuano il polinomio irriducibile, con coefficienti in GF(2),

$$(x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8) = x^4 + x + 1$$

Analogamente α^3 (oppure α^6, α^{12}, $\alpha^{24} = \alpha^9$, $\alpha^{48} = \alpha^3$, ...) individuano[18]

$$(x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9) = x^4 + x^3 + x^2 + x + 1$$

α^5 (oppure α^{10}, $\alpha^{20} = \alpha^5$, ...) individuano

$$(x - \alpha^5)(x - \alpha^{10}) = x^2 + x + 1$$

α^7 (oppure α^{14}, $\alpha^{28} = \alpha^{13}$, $\alpha^{56} = \alpha^{11}$, $\alpha^{112} = \alpha^7$, ...) individuano[19]

$$(x - \alpha^7)(x - \alpha^{14})(x - \alpha^{13})(x - \alpha^{11}) = x^4 + x^3 + 1$$

ed infine $\alpha^0 = 1$ individua

$$(x - 1) = x + 1$$

Quindi ogni polinomio con coefficienti in GF(q) (senza radici multiple) è individuabile da un sottoinsieme di radici: ad esempio le radici α^3, α^4 e α^5 individuano il polinomio $(x^4 + x^3 + x^2 + x + 1)(x^4 + x + 1)(x^2 + x + 1) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$.

L'equazione $x^{15}-1 = 0$ ha evidentemente come radici tutti gli elementi non nulli di GF(16). Basta infatti osservare che $(\alpha^n)^{15} = 1$ per ogni n. Si ha quindi la fattorizzazione[20]

$$x^{15} - 1 = x^{15} + 1 = (x - \alpha)(x - \alpha^2)(x - \alpha^3) \dots (x - \alpha^{15}) \tag{4.38}$$

Più in generale l'equazione $x^{q^m-1} - 1 = 0$ ha come radici tutti gli elementi non nulli di GF(q^m), ovvero

$$x^{q^m-1} - 1 = (x - \alpha)(x - \alpha^2)(x - \alpha^3) \dots (x - \alpha^{q^m-1}) \tag{4.39}$$

[18] per fare meno fatica a ridurre modulo 15: α^6, α^{12}, $\alpha^{24} = \alpha^9$, $\alpha^{18} = \alpha^3$, ...

[19] per fare meno fatica a ridurre modulo 15: α^{14}, $\alpha^{28} = \alpha^{13}$, $\alpha^{26} = \alpha^{11}$, $\alpha^{22} = \alpha^7$, ...

[20] in GF(2^m) sottrazione e somma sono equivalenti; quindi anziché $x^{15} - 1$ si usa scrivere $x^{15} + 1$

4.2.1 Rappresentazioni isomorfe di GF(q^m)

Nota: questo argomento può risultare un po' ostico ad una prima (o seconda!) lettura. Non è tuttavia indispensabile per proseguire, e può essere rimandato.

Come già visto, il campo GF(8) può essere generato dal polinomio $p(x) = x^3 + x + 1$ scegliendo α come elemento primitivo. Il polinomio generatore ha come radici anche α^2 e α^4, entrambi elementi primitivi. Se si pone $\gamma = \alpha^2$ e si confrontano le tabelle di addizione delle potenze di α e di quelle di γ si trova che sono *identiche*. Ad esempio risulta $\alpha+\alpha^6 = \alpha^5$ ed elevando al quadrato è immediato ottenere $\gamma + \gamma^6 = \gamma^5$. Gli elementi descritti dalle potenze di γ sono *permutati* rispetto alle potenze di α, ma le tabelle di addizione sono invariate: le due descrizioni del campo, con potenze di α oppure con potenze di $\gamma = \alpha^2$ sono isomorfe. La permutazione è così semplice che nulla sembra cambiare nelle operazioni nel campo. Una analoga permutazione isomorfa si ottiene ponendo $\gamma = \alpha^4$.

Il campo GF(8) ha anche altri elementi primitivi, ad esempio $\gamma = \alpha^3$, ma se ora si calcola $\gamma + \gamma^6$ si ottiene $\alpha^6 = \gamma^2 \neq \gamma^5$. Quindi nel campo generato da $p(x) = x^3 + x + 1$ se si ordinano gli elementi come potenze di α^3 si ottengono tabelle di addizione *diverse* da quelle con ordinamento secondo le potenze di α. Naturalmente si tratta solo di un effetto della permutazione degli elementi.

Volendo utilizzare $\gamma = \alpha^3$ come elemento primitivo, sarebbe naturale usare un polinomio $p(x)$ generatore del campo che avesse α^3 e le coniugate come radici, cioè

$$p(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12}) = x^3 + x^2 + 1 \tag{4.40}$$

Si tratta semplicemente del polinomio minimo di α^3. Se si genera il campo GF(8) con questo polinomio, e si indica con γ l'elemento primitivo, si ottiene

$$\gamma$$
$$\gamma^2$$
$$\gamma^3 = \gamma^2 + 1$$
$$\gamma^4 = \gamma^3 + \gamma = \gamma^2 + \gamma + 1$$
$$\gamma^5 = \gamma^3 + \gamma^2 + \gamma = \gamma + 1$$
$$\gamma^6 = \gamma^2 + \gamma$$
$$\gamma^7 = \gamma^3 + \gamma^2 = 1$$

Con questo *diverso* polinomio generatore si ottiene $\gamma+\gamma^6 = \gamma^2$. Le tabelle di addizione delle potenze di γ coincidono con quelle delle potenze di α^3 nella rappresentazione precedente, cioè si è ottenuta la permutazione desiderata.

Nella rappresentazione con polinomio generatore $p(x) = x^3 + x^2 + 1$ anche γ^3 è elemento primitivo, ed ha polinomio minimo x^3+x+1. E' facile verificare che le tabelle di addizione delle potenze di γ^3 nel campo generato da $p(x) = x^3 + x^2 + 1$ sono uguali a quelle di α

nel campo generato da $p(x) = x^3 + x + 1$. In definitiva, sia nel caso di un unico polinomio $p(x)$ sia con due diversi polinomi generatori del campo, la permutazione è particolarmente semplice e fa coincidere le tabelle di addizione se si ordinano gli elementi non nulli del campo secondo le potenze di elementi primitivi aventi lo stesso polinomio minimo.

4.2.2 Rappresentazione di GF(2^m) con altre basi

Nota: anche questo argomento può essere rimandato ad una successiva lettura.

La rappresentazione più comune degli elementi di GF(2^m) utilizza la *base polinomiale*: gli elementi del campo sono dati da combinazioni lineari degli elementi 1, $\alpha, \alpha^2, \ldots, \alpha^{m-1}$. I coefficienti della combinazione lineare formano la m-pla di bit che rappresenta l'elemento.

Tuttavia sono possibili, e talvolta più comode, altre rappresentazioni. Le conversioni di base sono facilitate se si usano basi duali l'una dell'altra.

Funzioni binarie lineari

Si consideri una generica funzione $f(y)$ definita sugli elementi di GF(2^m), che assuma solo valori binari (0 oppure 1) e sia *lineare*: $f(y + z) = f(y) + f(z)$.

Un esempio è il seguente: $f(y)$ coincide con una delle cifre binarie y_i nella rappresentazione polinomiale $y = \sum_{i=0}^{m-1} y_i \alpha^i$ $(i = 0, \ldots, m-1)$. Ricordando quanto visto nella Sez. 4.1.7 è evidente che questa funzione $f(y)$, calcolata successivamente per tutte le potenze di α, fornisce la sequenza pseudocasuale.

La condizione di linearità impone che la più generale funzione $f(y)$ sia combinazione lineare delle m cifre y_i:

$$f(y) = f(\sum_{i=0}^{m-1} y_i \alpha^i) = \sum_{i=0}^{m-1} c_i y_i \tag{4.41}$$

con coefficienti binari c_i arbitrari. Esistono evidentemente 2^m di tali funzioni, secondo i valori dei coefficienti c_i, di cui una identicamente nulla. Le $2^m - 1$ funzioni $f(y)$ non nulle sono combinazioni lineari di sequenze pseudocasuali con fasi diverse e quindi, calcolate per le potenze successive di α, forniscono una sequenza pseudocasuale. Le $2^m - 1$ funzioni $f(y)$ non nulle corrispondono alle $2^m - 1$ fasi possibili della sequenza pseudocasuale.

Una notevole proprietà delle sequenze pseudocasuali, qualunque ne sia la fase, e quindi una proprietà di ogni $f(y)$ è

$$f(\alpha^m y) = \sum_{i=0}^{m-1} p_i f(\alpha^i y) \tag{4.42}$$

dove i p_i sono i coefficienti binari del polinomio $p(x)$ generatore del campo, per $i = 0, 1, \ldots, m-1$.

Basi duali della base polinomiale

Fissata a piacere una delle $f(y)$ definite sopra, si cerchi (se esiste) un insieme di m elementi di GF(2^m) tali che si abbia *ortogonalità* tra questi e la base α^i ($i = 0, 1, \ldots, m-1$), rispetto alla funzione $f(y)$ scelta:

$$f(\alpha^i \mu_j) = \delta_{ij} = \begin{cases} 1 & j = i, \\ 0 & j \neq i \end{cases} \tag{4.43}$$

Di un generico elemento y del campo sono possibili la rappresentazione polinomiale

$$y = \sum_{i=0}^{m-1} y_i \alpha^i \tag{4.44}$$

e quella *duale*

$$y = \sum_{j=0}^{m-1} Y_j \mu_j \tag{4.45}$$

Usando per y la rappresentazione duale, e valutando $f(\alpha^j y)$ tenendo conto della linearità, si ottiene facilmente l'espressione dei coefficienti della rappresentazione duale

$$Y_j = f(\alpha^j y) \tag{4.46}$$

Moltiplicatore di *Berlekamp*

La rappresentazione duale è particolarmente utile per realizzare circuiti efficienti per la moltiplicazione di elementi di GF(2^m). Si voglia infatti calcolare il prodotto $c = ab$ utilizzando la rappresentazione polinomiale per a e quella duale per b e c. Si ha

$$C_j = f(\alpha^j ab) = f(\alpha^j \sum_{i=0}^{m-1} a_i \alpha^i b) = \sum_{i=0}^{m-1} a_i f(\alpha^{j+i} b) = \sum_{i=0}^{m-1} a_i B_{j+i} \tag{4.47}$$

Inoltre si ha, per le solite proprietà della sequenza pseudocasuale,

$$B_{j+m} = f(\alpha^{j+m} b) = \sum_{i=0}^{m-1} p_i f(\alpha^{j+i} b) = \sum_{i=0}^{m-1} p_i B_{j+i} \tag{4.48}$$

Le ultime due equazioni danno luogo ad un circuito molto efficiente per il calcolo *seriale* del prodotto (moltiplicatore di *Berlekamp*). Dati $B_0, B_1, \ldots, B_{m-1}$ i valori $B_m, \ldots, B_{2m-2}$ vengono calcolati *serialmente* mediante l'ultima equazione: si tratta del generatore di sequenze pseudocasuali, nella forma di Fig. 4.2, inizializzato con $B_0, B_1, \ldots, B_{m-1}$. I coefficienti C_j del prodotto vengono calcolati *serialmente* con m moltiplicatori binari (porte AND) riutilizzati per m volte.

Il moltiplicatore di *Berlekamp* è il moltiplicatore seriale che richiede il minimo di circuiteria logica. Lo si può modificare per ottenere un moltiplicatore *parallelo*, realizzando una rete combinatoria (di solito abbastanza semplice) che produca *tutti* i valori $B_m, \ldots, B_{2m-2}$ e moltiplicando per m il numero di porte AND (dovendo calcolare in parallelo $C_0, C_1, \ldots, C_{m-1}$).

Conversione di base

Volendo utilizzare il moltiplicatore di *Berlekamp* resta il problema della conversione di base. Nel caso di polinomio generatore del campo $p(x) = x^m + x^k + 1$ si può verificare che con una opportuna scelta della funzione $f(y)$ fra le $2^m - 1$ possibili la conversione di base si riduce alla quasi banale permutazione[21]

$$\mu_0 = \alpha^{k-1} \quad \mu_1 = \alpha^{k-2} \ldots \quad \mu_{k-1} = 1 \quad \mu_k = \alpha^{m-1} \ldots \quad \mu_{m-1} = \alpha^k \tag{4.49}$$

Si ottiene questo risultato, ma ci vuole un po' di pazienza, mostrando che è possibile scegliere la funzione $f(y)$ in modo che per $y = \alpha^{k-1}, \alpha^k, \alpha^{k+1}, \ldots, \alpha^{m+k-1}, \alpha^{m+k}, \ldots, \alpha^{2m-2}$ assuma i valori $1, 0, 0, \ldots, 1, 0, \ldots, 0$.

Nel caso di polinomio generatore del campo $p(x)$ con cinque coefficienti non nulli, come per $m = 8$, si può mostrare che oltre ad una permutazione occorrono due porte EXOR.

[21] operando serialmente una permutazione non è del tutto gratuita: occorre memorizzare alcuni dei coefficienti per restituirli nell'ordine giusto; operando in parallelo invece la permutazione non costa nulla

Capitolo 5

Codici a blocco e decodifica algebrica

5.1 Trasformata discreta di Fourier nei campi finiti

5.1.1 Definizione della trasformata discreta

In un campo finito è possibile definire la trasformata discreta di Fourier, con proprietà analoghe a quelle ben note, nel seguente modo: sia α un elemento[1] di ordine N di GF(p^m). La trasformata della N-pla di elementi di GF(p^m) $v_0, v_1, \ldots, v_{N-1}$ è[2]

$$V_j = \sum_{i=0}^{N-1} \alpha^{ij} v_i \qquad j = 0, 1, \ldots, N-1 \tag{5.1}$$

e la formula di antitrasformazione è

$$v_i = \frac{1}{N} \sum_{j=0}^{N-1} \alpha^{-ij} V_j \qquad i = 0, 1, \ldots, N-1 \tag{5.2}$$

Si noti che, a differenza dell'algebra dei numeri complessi, non si può scegliere N a piacere: infatti esistono radici N-esime dell'unità solo se N è un divisore di $p^m - 1$.

C'è anche da osservare che, nel termine $1/N$, N deve essere inteso modulo p. Ad esempio in GF(2^m) N è necessariamente dispari e quindi (modulo p) vale 1: quindi $1/N = 1$.

La dimostrazione della formula di antitrasformazione è del tutto analoga a quella solita, e qui viene riportata solo per chiarire meglio il ruolo di N. Trasformando e antitrasformando

[1] generalmente si indica con α l'elemento primitivo scelto per il campo; qui invece α indica un elemento generico, diverso da 0 o 1; se la notazione non piacesse si sostituisca *ovunque* α con un altro simbolo, ad esempio β

[2] la N-pla trasformata è anche esprimibile come prodotto della matrice (di *Vandermonde*) $A = [a_{ij}]$, con $a_{ij} = \alpha^{ij}$, per il vettore $[v_0\ v_1\ \cdots\ v_{N-1}]^T$

(ignorando per il momento il fattore $1/N$) si ha

$$\sum_{j=0}^{N-1} \alpha^{-ij} (\sum_{k=0}^{N-1} \alpha^{kj} v_k) = \sum_{k=0}^{N-1} v_k \sum_{j=0}^{N-1} \alpha^{(k-i)j} \tag{5.3}$$

Quando $k - i = 0$ la somma interna è la somma (con le regole del campo) di N uni, e quindi vale N modulo p. Si ottiene quindi il solo v_i, moltiplicato per N (mod p).

Tutti gli altri casi non danno contributo perché la somma interna è nulla. Infatti dalla semplice uguaglianza tra polinomi, facilmente verificabile,

$$x^N - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{N-1}) \tag{5.4}$$

ponendo $x = \alpha^r$ (con r generico, diverso da 0 modulo N) si ottiene

$$\alpha^{rN} - 1 = (\alpha^r - 1) \sum_{j=0}^{N-1} \alpha^{rj} \tag{5.5}$$

Poiché $\alpha^{rN} - 1 = (\alpha^N)^r - 1 = 0$ mentre $\alpha^r - 1 \neq 0$ si ha

$$\sum_{j=0}^{N-1} \alpha^{rj} = 0 \tag{5.6}$$

Nota: riesaminando la dimostrazione si può osservare che non è neppure necessario che la struttura algebrica su cui si opera sia un campo: basterebbe un anello, purché esista una radice N-ma dell'unità ed esista l'elemento inverso di N (che in un anello non è garantito). Tuttavia la trasformata discreta definita su un anello non sembra di grande interesse pratico.

Esempio: in GF(5), $\alpha = 2$ è primitivo e si ha $\alpha^2 = 4$, $\alpha^3 = 3$, $\alpha^4 = 1$. Quindi $N = 4$. Inoltre $1/4 = 4$ (infatti $4 \cdot 4 = 1$). Trasformando ad esempio la quaterna v=[1 2 3 4]^T si ottiene facilmente

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 3 \\ 2 \end{bmatrix} \tag{5.7}$$

e antitrasformando

$$v = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 4 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 4 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 4 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \tag{5.8}$$

5.1.2 Proprietà della trasformata discreta

In vista delle applicazioni alla codifica è importante osservare che se si associano alle N-ple $v_0, v_1, \ldots, v_{N-1}$ e $V_0, V_1, \ldots, V_{N-1}$ i polinomi

$$v(x) = v_0 + v_1 x + \ldots + v_{N-1} x^{N-1} \tag{5.9}$$

$$V(x) = V_0 + V_1 x + \ldots + V_{N-1} x^{N-1} \tag{5.10}$$

risulta evidentemente

$$V_j = v(\alpha^j) \tag{5.11}$$

$$v_i = \frac{1}{N} V(\alpha^{-i}) \tag{5.12}$$

e in particolare $V_j = 0$ implica che il polinomio $v(x)$ abbia uno zero in α^j, cioè che $v(x)$ abbia come fattore $(x - \alpha^j)$; analogamente se $v_i = 0$ il polinomio $V(x)$ si azzera in α^{-i}, cioè $V(x)$ ha come fattore $(x - \alpha^{-i})$.

Le proprietà della trasformata discreta definita su un campo di *Galois* sono del tutto analoghe alle usuali, e con analoga dimostrazione. Ad esempio la trasformata del prodotto $e_i = f_i g_i$ si ottiene dalla convoluzione circolare:

$$E_j = \sum_{i=0}^{N-1} \alpha^{ij} f_i \left(\frac{1}{N} \sum_{k=0}^{N-1} \alpha^{-ik} G_k\right) = \frac{1}{N} \sum_{k=0}^{N-1} G_k \left(\sum_{i=0}^{N-1} \alpha^{i(j-k)} f_i\right) = \frac{1}{N} \sum_{k=0}^{N-1} G_k F_{j-k} \tag{5.13}$$

dove $j - k$ è da intendere modulo N (si osservi che $\alpha^{i(j-k)} = \alpha^{i(N+j-k)}$). Altre proprietà da ricordare sono quelle relative alla rotazione (ciclica) delle sequenze:

$$\alpha^i v_i \longleftrightarrow V_{j+1} \tag{5.14}$$

$$v_{i-1} \longleftrightarrow \alpha^j V_j \tag{5.15}$$

Si noti anche, poiché $\alpha^i \neq 0 \ \forall i$, che i v_i non nulli restano tali ruotando la trasformata (e viceversa).

Infine una proprietà utile (meno nota delle precedenti) è: se almeno un $V_j \neq 0$, cioè se $V(x) \neq 0$, e se $V_{N-1} = V_{N-2} = \ldots = V_{N-K} = 0$, almeno $K + 1$ fra i v_i sono diversi da zero.

Dimostrazione: $V(x)$ ha grado $\leq N - K - 1$ e quindi, per il teorema fondamentale dell'algebra, ha al massimo $N - K - 1$ zeri. Ne deriva che non più di $N - K - 1$ fra i v_i sono nulli.

Corollario: non occorre che i V_j nulli siano nelle posizioni di grado più elevato; infatti ruotando ciclicamente la trasformata di k posizioni i valori v_i vengono moltiplicati per $\alpha^{ki} \neq 0$. Basta quindi che gli zeri siano in posizioni (ciclicamente) consecutive!

Altre proprietà interessanti valgono quando la trasformata è valutata in GF(q^m) ma i $v_i \in$ GF(q) (si ripensi alle simmetrie complesse coniugate della trasformata nel campo complesso di una N-pla reale):

$$v_i \in \mathrm{GF}(q) \Longleftrightarrow V_j^q = V_{jq} \tag{5.16}$$

dove jq è da intendere modulo N. Per dimostrare che $V_j^q = V_{jq}$ bastano passaggi elementari:

$$V_j^q = v^q(\alpha^j) = v(\alpha^{jq}) = V_{jq} \tag{5.17}$$

Dimostrare il risultato duale, che è comunque meno utile nelle applicazioni, non è altrettanto semplice.

5.2 Codici ciclici

Sia α un elemento di ordine N in GF(q^m). La N-pla $c_0, c_1, \ldots, c_{N-1}$ è per definizione parola di codice se la trasformata è nulla in un insieme prefissato di posizioni j_l: $C_{j_l} = 0$ ($l = 1, 2, \ldots$), ovvero se il polinomio $c(x)$ si azzera negli α^{j_l} corrispondenti. Le posizioni j_l degli zeri individuano il codice. Data l'equivalenza tra la N-pla $c_0, c_1, \ldots, c_{N-1}$ e il corrispondente polinomio $c(x) = c_0 + c_1 x + \ldots + c_{N-1} x^{N-1}$ si usa anche dire che il polinomio $c(x)$ è parola di codice.

Il numero N di cifre del codice, ovvero la lunghezza del codice, è l'ordine di α; la lunghezza massima $q^m - 1$ è ottenuta se α è elemento primitivo; si deve operare in un campo GF(q^m) con un numero di elementi maggiore della lunghezza N desiderata.

5.2.1 Polinomio generatore

Si può definire il codice in modo equivalente dicendo che ogni parola di codice $c(x)$ contiene come fattore $g(x)$, ovvero che è divisibile per $g(x)$, dove

$$g(x) = \prod (x - \alpha^{j_l}) \tag{5.18}$$

è chiamato *polinomio generatore* del codice[3]. Poiché $x^N - 1$ ha come radici *tutte* le potenze di α è evidente che $g(x)$ è un divisore di $x^N - 1$.

Nota: la N-pla ruotata ciclicamente c_{i-1} ha trasformata $\alpha^j C_j$, ed è quindi parola di codice se lo è c_i. Da tale proprietà deriva il nome dato alla classe di codici. Questa proprietà si dimostra facilmente anche *nel dominio dei tempi*. Infatti se $c(x)$ è una generica parola del codice, il polinomio corrispondente alla parola ruotata di una posizione è

$$xc(x) + c_{N-1}(1 - x^N) \tag{5.19}$$

[3]si osservi che non ha senso, con questa definizione, un polinomio generatore con radici ripetute; questo caso comunque non ha alcun interesse pratico

ed è evidentemente divisibile per $g(x)$.

Nota: il polinomio generatore $g(x)$ è una delle parole di codice. Sono parole di codice anche tutte le sue repliche cicliche, e tutte le combinazioni di queste. Si può quindi ritenere $g(x)$ il *prototipo* di tutte le parole di codice.

5.2.2 Polinomio di parità

Poiché $g(x)$ è un divisore di $x^N - 1$ esiste il *polinomio di parità* $h(x)$ tale che

$$h(x)g(x) = x^N - 1 \tag{5.20}$$

Il polinomio di parità svolge lo stesso ruolo della matrice di parità nella descrizione elementare dei codici a blocco. Per una generica parola di codice $c(x) = g(x)a(x)$ si ha

$$h(x)c(x) = h(x)g(x)a(x) = (x^N - 1)a(x) \tag{5.21}$$

Poiché $a(x)$ ha grado massimo $K - 1$, $(x^N - 1)a(x) = h(x)c(x)$ ha nulli i termini di grado $N - 1$, $N - 2$, ..., K. Si hanno quindi le $N - K$ *regole di parità* del codice:

$$\begin{aligned}
&h_0 c_{N-1} + h_1 c_{N-2} + h_2 c_{N-3} + \ldots + h_K c_{N-K-1} = 0 \\
&h_0 c_{N-2} + h_1 c_{N-3} + h_2 c_{N-4} + \ldots + h_K c_{N-K-2} = 0 \\
&\ldots \\
&h_0 c_K + h_1 c_{K-1} + h_2 c_{K-2} + \ldots + h_K c_0 = 0
\end{aligned} \tag{5.22}$$

Si tratta di una sola regola, applicata a segmenti diversi (di lunghezza $K + 1$) della parola di codice. Può essere interessante osservare che, fissate arbitrariamente K cifre $c_{N-1}, \ldots, c_{N-K}$, la prima equazione consente di determinare la cifra c_{N-K-1} e le altre via via le successive cifre. Da questa osservazione si ottiene facilmente una struttura per il codificatore, che può essere interessante per piccoli valori di K.

5.2.3 Codice duale

Viene detto *codice duale* quello che ha $h(x)$ come polinomio generatore e $g(x)$ come polinomio di parità. Così come la generica parola del codice è $c(x) = g(x)a(x)$, le parole del codice duale sono $d(x) = h(x)b(x)$. Si ha

$$c(x)d(x) = h(x)b(x)g(x)a(x) = b(x)a(x)(x^N - 1) \tag{5.23}$$

Poiché $a(x)$ ha grado massimo $K - 1$ e $b(x)$ ha grado massimo $N - K - 1$, il prodotto $b(x)a(x)(x^N - 1)$ ha nullo il termine di grado $N - 1$. Si ha quindi

$$c_0 d_{N-1} + c_1 d_{N-2} + c_2 d_{N-3} + \ldots + c_{N-1} d_0 = 0 \tag{5.24}$$

ovvero si ha *ortogonalità* tra una qualsiasi parola $c(x)$ del codice e una qualsiasi $d(x)$ del codice duale[4].

5.2.4 Struttura del codificatore

Se $c_i \in \mathrm{GF}(q^m)$, cioè l'alfabeto delle parole di codice è $\mathrm{GF}(q^m)$, non ci sono altri vincoli né sulla trasformata né sul polinomio generatore. Il codificatore può essere realizzato in uno dei seguenti modi:

- codifica nel *dominio delle frequenze* (non comune in pratica): si pone $C_{j_l} = 0$; si scelgono liberamente gli altri C_j secondo l'informazione da trasmettere; infine si calcola l'antitrasformata discreta c_i
- codifica *nel dominio dei tempi*: sia $N - K$ il numero delle radici, e quindi il grado, di $g(x)$; si pongano K cifre d'informazione $i_0, i_1, \ldots, i_{K-1}$ come coefficienti di un polinomio $i(x)$ di grado[5] $K - 1$; le parole di codice, di grado[6] $N - 1$, sono date da

  $$c(x) = g(x)i(x) \tag{5.25}$$

  Nota: la moltiplicazione di un polinomio generico $i(x)$ per un polinomio prefissato $g(x)$ si realizza semplicemente: in forma seriale è un filtro FIR con $N - K + 1$ coefficienti, che opera con l'algebra di $\mathrm{GF}(q^m)$. Si noti che le cifre d'informazione non vengono trasmesse *in chiaro*: il codice viene detto *non sistematico*.

  Nota: per il numero di cifre d'informazione e di cifre complessive vengono usati anche i simboli k e n.

- codifica *sistematica* nel dominio dei tempi (la più comune in pratica):

  $$c(x) = i(x)x^{N-K} - R\left[\frac{i(x)x^{N-K}}{g(x)}\right] \tag{5.26}$$

  dove $R[\cdot]$ è il resto della divisione, che ha grado massimo $N-K-1$. Avendo sottratto l'eventuale resto non nullo della divisione è evidente che $c(x)$ è divisibile per $g(x)$ e quindi parola di codice. I K coefficienti di grado più elevato di $c(x)$ coincidono con le cifre d'informazione, mentre i coefficienti del resto (che ha grado massimo $N-K-1$) rimangono nelle altre $N - K$ posizioni di grado più basso.

  Nota: la divisione per $g(x)$ ha la stessa complessità della moltiplicazione per $g(x)$; occorre semplicemente un circuito retroazionato con guadagni pari ai coefficienti del polinomio generatore.

[4] si noti che la parola del codice duale è ribaltata; viene chiamato *duale* anche il codice con parole ribaltate una seconda volta, con polinomio generatore con coefficienti ribaltati $x^K h(x^{-1})$

[5] $K - 1$ è il grado massimo possibile; evidentemente il grado risulta minore se $i_{K-1} = 0$

[6] vedi nota precedente

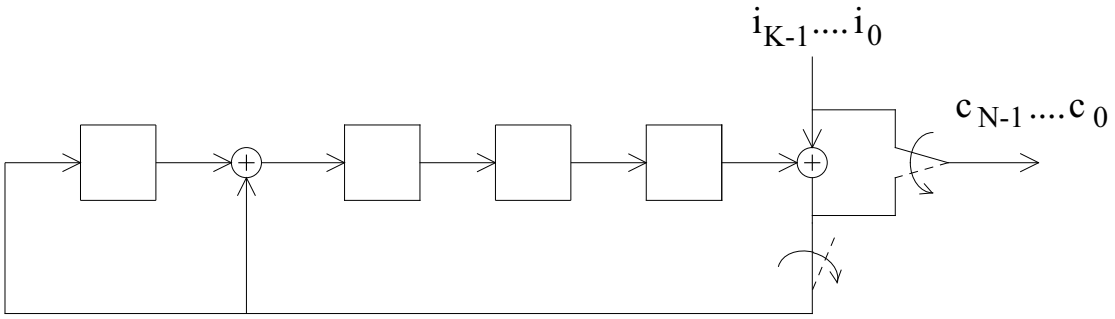

Figura 5.1: Codificatore per il codice con $g(x) = x^4 + x + 1$

In Fig. 5.1 è mostrato, ad esempio, il codificatore per $g(x) = x^4 + x + 1$. Inizialmente i registri sono azzerati. Seguendo passo passo le operazioni effettuate dal circuito si può vedere che dopo K *clock* negli $N - K$ registri è contenuto il resto[7]. Si apre quindi l'anello di retroazione e si invia sul canale il contenuto dei registri.

Nota: l'insieme di tutte le parole di codice, nella versione sistematica e non, è evidentemente lo stesso! le parole di codice vengono associate alle cifre d'informazione in modo diverso.

Nota: solitamente si trasmettono prima le cifre d'informazione e poi il resto; la convenzione usuale è di partire dal coefficiente di grado $N - 1$ di $c(x)$, cioè da i_{K-1}.

Se $c_i \in \mathrm{GF}(q)$, cioè se l'alfabeto delle parole di codice è limitato a $\mathrm{GF}(q)$, ci sono ulteriori vincoli sulla trasformata e quindi sul polinomio generatore. La codifica *nel dominio delle frequenze* deve rispettare *tutte* le condizioni $C_j^q = C_{jq}$. I corrispondenti vincoli sul polinomio generatore sono i seguenti: se α^j è radice di $g(x)$ anche α^{jq}, α^{jq^2}, ... devono essere radici. In questo modo $g(x)$ ha coefficienti in $\mathrm{GF}(q)$ e quindi scegliendo anche $i(x)$ con coefficienti in $\mathrm{GF}(q)$ si ottengono parole di codice con coefficienti in $\mathrm{GF}(q)$.

Nota: la struttura dei codificatori nel *dominio del tempo* non varia. Tuttavia il grado di $g(x)$ aumenta perché oltre alle radici *desiderate* occorre introdurne altre.

Infine conviene chiarire perché codici basati su un elemento α di ordine $N \leq q^m - 1$ di $\mathrm{GF}(q^m)$ hanno lunghezza N, e non superiore. Se si accettasse, ad esempio, la parola di codice $x^{N+M} - x^M = x^M(x^N - 1)$, evidentemente divisibile per $g(x)$, si avrebbero parole di codice con due soli simboli non nulli. La distanza minima del codice sarebbe $d = 2$, ritenuta troppo piccola per essere di interesse pratico.

5.2.5 Modificazioni dei codici (binari)

Un codice può essere *esteso* aggiungendo una cifra di parità complessiva: un bit somma modulo 2 di tutti i bit d'informazione e di parità. Tutte le parole di codice hanno quindi

[7]suggerimento: si consideri il caso di un solo bit d'informazione i_{K-l} ($l = 1, \ldots, K$) diverso da zero; per sovrapposizione degli effetti ...

un numero pari di uni. La distanza minima del codice, se era dispari come spesso avviene, aumenta di uno. Il codice non è più ciclico.

Un codice ciclico con distanza minima dispari può essere *espurgato* moltiplicando il polinomio generatore per il fattore $x+1$. Aumentando di uno il grado di $g(x)$ si riduce di uno K. Si vede facilmente che tutte le parole del codice espurgato hanno un numero pari di uni, e quindi la distanza minima è pari, e quindi aumentata di uno. Infatti risulta $c(1) = 0$ e quindi la somma delle cifre del codice è nulla, cosa possibile solo con un numero pari di uni. Il codice espurgato è ciclico.

Infine il codice può essere *accorciato*. Si azzerano i bit d'informazione nelle prime b posizioni. Ovviamente non si trasmettono questi dati, e si ottiene quindi un codice con $K' = K - b$ ed $N' = N - b$. Il codice accorciato non è ciclico.

5.2.6 Alcune proprietà dei codici ciclici

Il polinomio $c(x) = 0$ è sempre parola di codice.

La somma (o differenza) di due parole di codice è una parola di codice: il codice è lineare.

Due diverse parole di codice differiscono quindi in almeno tante posizioni quanti sono (al minimo) i simboli non nulli delle parole di codice, esclusa $c(x) = 0$. Questa viene detta *distanza minima* del codice.

E' noto che un codice con distanza minima d può essere utilizzato per rivelare errori in $d-1$ (o meno) posizioni, oppure per correggere fino a $(d-1)/2$ errori se d è dispari e $d/2-1$ se d è pari.

Fondamentale è il *BCH bound*: se $C_j = 0$ in $d-1$ posizioni (ciclicamente) consecutive la distanza minima del codice è almeno d. Infatti ogni parola di codice non nulla ha trasformata con almeno $d-1$ zeri consecutivi, e quindi contiene almeno d elementi non nulli.

5.3 Codici *BCH*

Codici introdotti da *Bose*, *Ray-Chauduri* e *Hocquengem* (1960):

$c_i \in \mathrm{GF}(q)$

$C_i \in \mathrm{GF}(q^m)$

$C_{j_0} = C_{j_0+1} = \ldots = C_{j_0+d-2} = 0 \quad (j_0 \text{ arbitrario})$

E' garantita la distanza minima d. In taluni casi, non frequenti, l'effettiva distanza minima risulta maggiore di quella *progettata*.

Nota: la trasformata ha altri elementi nulli, a causa del vincolo $C_{jq} = C_j^q$: $C_{j_0 q} = C_{j_0 q^2} = \ldots = 0$; $C_{(j_0+1)q} = C_{(j_0+1)q^2} = \ldots = 0$; ...

N	K	d	$g(x)$
15	11	3	23
	7	5	721
	5	7	2467
	1	15	77777
31	26	3	45
	21	5	3551
	16	7	107657
	11	11	5423325
	...	...	...
63	57	3	103
	51	5	12471
	45	7	1701317
	39	9	...
	36	11	...
	...	...	...

N	K	d	$g(x)$
127	120	3	211
	113	5	41567
	106	7	11554743
	99	9	...
	92	11	...
	...	...	...
255	247	3	435
	239	5	267543
	231	7	156720655
	223	9	...
	215	11	...
	207	13	...
	199	15	...
	191	17	...
	...	...	...

Tabella 5.1: Alcuni codici *BCH* binari

5.3.1 Codici *BCH* binari

Il caso $q = 2$ e α primitivo è il più frequente nelle applicazioni. Vediamo un semplice esempio in GF(16)=GF(2^4):

$N = 15 \quad j_0 = 1 \quad d = 7$

Radici:

α (e quindi anche $\alpha^2, \alpha^4, \alpha^8$)

α^3 (e quindi anche$\alpha^6, \alpha^{12}, \alpha^{24} = \alpha^9$)

α^5 (e quindi anche α^{10})

Polinomio generatore:

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)(x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^9)(x-\alpha^5)(x-\alpha^{10}) =$$
$$= (x^4+x+1)(x^4+x^3+x^2+x+1)(x^2+x+1) = x^{10}+x^8+x^5+x^4+x^2+x+1$$

Quindi $N - K = 10$ e $K = 5$ (5 bit d'informazione e 10 di parità). La distanza minima del codice risulta effettivamente pari a 7.

I parametri di alcuni codici *BCH* binari sono riportati in Tab. 5.1. Solitamente il numero di cifre di parità richiesto da un codice con $N = 2^m - 1$ per ottenere $d = 2t + 1$, e quindi in grado di correggere t errori, è mt. In alcuni casi risulta minore.

Nella Tab. 5.1 i polinomi generatori sono dati in *ottale*. Ad esempio per $N = 15$ e $K = 5$ il numero ottale 2467, convertito in binario, dà 10100110111. Questi sono, nell'ordine, i coefficienti del polinomio generatore $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ già trovato in precedenza. In modo analogo si ricava dalla tabella, ad esempio, che per $N = 31$ e $K = 21$ il polinomio generatore è $g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$.

5.3.2 Codici di *Hamming*

I codici *BCH* includono come caso particolare i *codici di Hamming*. Questi sono infatti i *BCH* correttori di un solo errore. Il polinomio generatore $g(x)$ ha come radici α e le sue potenze α^2, α^4, ... richieste per avere coefficienti binari. Si riconosce facilmente che il polinomio generatore $g(x)$ dei codici di *Hamming* non è altro che il polinomio $p(x)$ generatore del campo. Operando in GF(2^m), e quindi con $N = 2^m - 1$, il polinomio generatore ha grado m. Ciò equivale a dire che tra N e K intercorre la relazione

$$N = 2^{N-K} - 1 \tag{5.27}$$

5.3.3 Codici *BCH* non primitivi

Se la radice N-esima dell'unità utilizzata per la costruzione del codice non è elemento primitivo del campo la lunghezza N del codice non è $2^m - 1$, ma un suo divisore. In tal caso il codice ciclico è detto *non primitivo*. Il caso più notevole di codice *BCH* non primitivo è il codice di *Golay*, ottenuto in GF(2^{11}) con $\beta = \alpha^{89}$, che ha ordine 23. Scegliendo β come radice del polinomio generatore si hanno anche le radici β^2, β^4, β^8, β^{16}, $\beta^{32} = \beta^9$, β^{18}, $\beta^{36} = \beta^{13}$, $\beta^{26} = \beta^3$, β^6 e β^{12} (poi le radici si ripetono). Il codice ha quindi $N = 23$ e $K = 12$. Vi sono quattro potenze consecutive di β, e quindi $d \geq 5$. In questo caso è noto che la vera distanza minima è $d = 7$, e quindi il codice corregge fino a 3 errori[8]. Il polinomio generatore, a conti fatti, risulta

$$g(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1 \tag{5.28}$$

(oppure si può usare il polinomio con i coefficienti ribaltati).

5.3.4 Codici *BCH* non binari

Nel caso, meno frequente nelle applicazioni, di dimensione dell'alfabeto dei simboli $q \neq 2$ l'unica differenza è che la radice α implica che siano radici di $g(x)$ anche α^q, α^{q^2}, ...

Si noti che q può essere sia un numero primo sia potenza di un numero primo. Un semplice esempio con $q = 4$ in GF(16) è il seguente:

$$N = 15 \quad j_0 = 1 \quad d = 5$$

[8] anzi questo codice è famosissimo perché corregge *tutti* e *soli* gli errori semplici, doppi e tripli; in tempi in cui questa proprietà sembrava importante il codice fu definito *perfetto*

Radici:

α (e quindi anche α^4)

α^2 (e quindi anche α^8)

α^3 (e quindi anche α^{12})

Polinomio generatore:

$$g(x) = (x-\alpha)(x-\alpha^4)(x-\alpha^2)(x-\alpha^8)(x-\alpha^3)(x-\alpha^{12}) =$$
$$= x^6 + \alpha^{10}x^5 + x^4 + x^3 + \alpha^5x^2 + \alpha^5x + 1 = x^6 + \beta^2x^5 + x^4 + x^3 + \beta x^2 + \beta x + 1$$

Infatti si è già visto[9] che gli elementi di GF(4) sono 0, 1, $\beta = \alpha^5$ e $\beta^2 = \alpha^{10}$. Quindi $N - K = 6$ e $K = 9$ (9 cifre *quaternarie* d'informazione e 6 di parità).

Per ottenere distanza $d = 7$ si aggiungerebbero le radici α^5 (si noti che $\alpha^{20} = \alpha^5$) e α^6 (quindi anche $\alpha^{24} = \alpha^9$). Il polinomio generatore avrebbe grado 9. Tuttavia se si sceglie $j_0 = 0$ la radice $a^0 = 1$ sostituisce le due radici α^6 e α^9. Il polinomio generatore ha quindi grado 8, e si ha $K = 7$. Analogo risultato si ottiene con $j_0 = 5$ oppure $j_0 = 10$. Nel caso dei codici *BCH* non binari questa situazione può presentarsi, di tanto in tanto.

5.4 Codici *Reed-Solomon*

Nei codici *Reed-Solomon* (1960) c_i e C_j sono tratti dallo stesso alfabeto GF(q). Per ottenere la distanza minima d si pone:

$$C_{j_0} = C_{j_0+1} = \ldots = C_{j_0+d-2} = 0 \quad (j_0 \text{ arbitrario}) \tag{5.29}$$

Non ci sono altri vincoli. Il polinomio generatore è

$$g(x) = (x-\alpha^{j_0})\ldots(x-\alpha^{j_0+d-2}) \tag{5.30}$$

ed ha grado $d-1$. Il codice ha $d-1$ cifre di parità e $N-d+1$ d'informazione.

Esempio in GF(16):

$$N = 15 \quad j_0 = 1 \quad d = 5$$
$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) = x^4 + \alpha^{13}x^3 + \alpha^6x^2 + \alpha^3x + \alpha^{10}$$

Quindi $N - K = 4$ e $K = 11$: 11 cifre d'informazione e 4 di parità. Si noti che non sono cifre binarie; ciascuna cifra è un elemento di GF(16) e quindi può essere considerata un *byte* di 4 bit.

[9] si era anche affermato che il polinomio irriducibile generatore di GF(16) come estensione di GF(4) è $p(x) = x^2 + x + \beta$; ora è facile verificarlo: $p(x)$ deve avere come radice α, e quindi α^4; dunque $p(x) = (x-\alpha)(x-\alpha^4) = x^2 + x + \beta$

Nei codici *Reed-Solomon* (*RS*) la distanza minima risulta sempre uguale a quella *progettata*, ed anzi si dimostra che non possono esistere codici con distanza maggiore di $N - K + 1$.

I codici *RS* sono interpretabili anche come *BCH* non binari, con $q = q^m$. Essendo $m = 1$ ogni radice del polinomio generatore non ne implica altre. Tuttavia i codici *RS* sono stati introdotti *indipendentemente* dai *BCH*.

Infine se $q = 2^m$ e si interpreta un simbolo dell'alfabeto GF(q) come una m-pla di bit, i codici *Reed-Solomon* hanno $m(2^m - 1)$ bit. Più bit errati concentrati nello stesso *byte* di m bit valgono *un solo errore*. Questa caratteristica è particolarmente apprezzabile in caso di canali binari che introducano pacchetti di errori, anziché errori isolati.

5.5 Decodifica algebrica (codici *BCH* e *RS*)

La trasformata di Fourier della parola di codice trasmessa è nulla nelle $d - 1$ posizioni consecutive $j_0, j_0 + 1, \ldots, j_0 + d - 2$. Se la N-pla ricevuta è

$$v_i = c_i + e_i \tag{5.31}$$

dove e_i è il valore dell'errore (eventuale) in posizione i-esima, risulta disponibile una *finestra spettrale* di lunghezza $d - 1$ attraverso cui osservare la sola trasformata dell'errore:

$$V_j = E_j \quad j = j_0, j_0 + 1, \ldots, j_0 + d - 2 \tag{5.32}$$

Sia $e_i \neq 0$ nelle posizioni i_1, i_2, …, i_ν, dove ν è il numero degli errori. Sia anche $\nu \leq t$, dove t è il potere correttore del codice[10]. Si può quindi scrivere un sistema di equazioni

$$E_j = \sum_{n=1}^{\nu} e_{i_n} \alpha^{j i_n} \quad j = j_0, j_0 + 1, \ldots, j_0 + d - 2 \tag{5.33}$$

lineare nelle incognite e_{i_n} (valori degli errori) ma *non lineare* nelle incognite i_n (posizioni degli errori).

5.5.1 Polinomio locatore degli errori

Per semplificare la soluzione si preferisce individuare le posizioni degli errori attraverso il *polinomio locatore degli errori* definito come

$$\Lambda(x) = \prod_{n=1}^{\nu} (1 - x\alpha^{i_n}) = \Lambda_0 + \Lambda_1 x + \ldots + \Lambda_\nu x^\nu \tag{5.34}$$

Il polinomio locatore si annulla, per costruzione, in α^{-i_1}, α^{-i_2}, …, α^{-i_ν}. Quindi determinare il polinomio e calcolarne le radici equivale a determinare il numero degli errori e a localizzarne le posizioni[11]. Si noti che $\Lambda_0 = 1$, per cui il polinomio $\Lambda(x)$ ha ν coefficienti

[10] calcolato sulla base della distanza *progettata*, che in alcuni casi è minore della vera distanza minima

[11] talvolta si usa la definizione $\Lambda(x) = \prod_{n=1}^{\nu} (x - \alpha^{i_n})$; il polinomio ha come radici α^{i_1}, α^{i_2}, …, α^{i_ν}

incogniti. Anche ν è a priori incognito.

Ci si può chiedere se sarebbe accettabile un polinomio locatore con radici spurie, in posizioni senza errore. Non farebbe gran danno, perché poi si troverebbe nullo il *valore* dell'errore. Tuttavia sarebbe una complicazione inutile. Si cerca sempre (se esiste) il polinomio di grado minimo che corrisponde a *tutte e sole* le posizioni degli errori.

Poichè il polinomio locatore ha radici $\alpha^{-i_1}, \alpha^{-i_2}, \ldots, \alpha^{-i_\nu}$ la corrispondente antitrasformata λ_i è nulla in tutte (e sole) le posizioni degli errori. Si ha quindi

$$\lambda_i e_i = 0 \quad i = 0, 1, \ldots, N-1 \tag{5.35}$$

Trasformando si ottiene

$$\sum_{j=0}^{N-1} \Lambda_j E_{k-j} = \sum_{j=0}^{\nu} \Lambda_j E_{k-j} = 0 \quad k = 0, 1, \ldots, N-1 \tag{5.36}$$

o anche, poiché $\Lambda_0 = 1$,

$$\sum_{j=1}^{\nu} \Lambda_j E_{k-j} = -E_k \quad k = 0, 1, \ldots, N-1 \tag{5.37}$$

Queste N equazioni sono *lineari* nelle incognite $\Lambda_1, \ldots, \Lambda_\nu$. Le equazioni che corrispondono a $k = j_0 + \nu, \ldots, j_0 + d - 2$, per un totale di $d - 1 - \nu$ equazioni, contengono solo valori di E_{k-j} *noti*.

L'insieme di queste equazioni, esprimibili anche in altre forme equivalenti, va sotto il nome di *key equation*. Si tratta infatti del punto chiave per la decodifica algebrica.

Se $\nu \leq t$ ci sono equazioni a sufficienza per determinare il polinomio locatore. Resta solo da mostrare che, scelte ν *equazioni consecutive*, queste sono linearmente indipendenti e quindi il sistema ha un'unica soluzione[12]. Come già detto, il numero di errori ν non è noto *a priori* e quindi viene determinato esaminando *tutte* le equazioni disponibili e valutando quante risultano indipendenti.

[12]Per dimostrare l'indipendenza si può procedere, con un po' di pazienza, come segue: preso un k qualsiasi, l'antitrasformata della riga di coefficienti $E_k, E_{k-1}, \ldots, E_{k+1}$ è (come si può ricavare con un uso attento delle proprietà della trasformata discreta)

$$a_i = \frac{1}{N} \sum_{j=0}^{N-1} E_{k-j} \alpha^{-ij} = \frac{1}{N} \sum_{l=0}^{N-1} E_l \alpha^{-i(k-l)} = \alpha^{-ik} e_{-i} \tag{5.38}$$

ed ha quindi solo ν elementi non nulli. Analogamente si trova che le antitrasformate delle righe successive di coefficienti sono date dagli stessi a_i, moltiplicati rispettivamente per α^{-i}, α^{-2i}, ...
Se una combinazione lineare di ν righe di coefficienti potesse essere nulla, sarebbe nulla anche la corrispondente antitrasformata. Ma ciò è evidentemente impossibile, perché *nel dominio del tempo* (e considerando solo gli e_i non nulli) i coefficienti della combinazione sono dati dalla matrice *di Vandermonde*

$$V = \begin{bmatrix} \alpha^{i_1 k} & \alpha^{i_2 k} & \ldots & \alpha^{i_\nu k} \\ \alpha^{i_1 (k+1)} & \alpha^{i_2 (k+1)} & \ldots & \alpha^{i_\nu (k+1)} \\ \alpha^{i_1 (k+2)} & \alpha^{i_2 (k+2)} & \ldots & \alpha^{i_\nu (k+2)} \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix}$$

5.5.2 Una dimostrazione alternativa del *BCH bound*

Può essere interessante utilizzare il polinomio locatore per dimostrare in altro modo il *BCH bound*. Si supponga che esista una parola di codice $c(x)$ con $\nu < d$ cifre non nulle, nelle posizioni i_1, i_2, ..., i_ν. Si definisca un *polinomio locatore* della parola $c(x)$

$$\Lambda(x) = \prod_{n=1}^{\nu}(1 - x\alpha^{i_n}) = \Lambda_0 + \Lambda_1 x + \ldots + \Lambda_\nu x^\nu \tag{5.39}$$

Poiché $\lambda_i c_i = 0$ in tutte le N posizioni, si ottiene

$$\sum_{j=1}^{\nu} \Lambda_j C_{k-j} = -C_k \quad k = 0, 1, \ldots, N-1 \tag{5.40}$$

Se $\nu < d$, essendo $C_j = 0$ in $d-1$ posizioni consecutive *tutti* i C_j risultano nulli. Dunque non esiste una parola di codice non nulla con meno di d cifre non nulle.

5.5.3 Valutazione dei valori degli errori

Determinato il polinomio locatore, si può procedere in più modi. Un primo metodo utilizza recursivamente le restanti equazioni

$$E_k = -\sum_{j=1}^{\nu} \Lambda_j E_{k-j} \quad k = j_0 + d - 1, j_0 + d, \ldots \tag{5.41}$$

per completare il calcolo della trasformata degli errori; quindi si antitrasforma. Il metodo è conveniente se si dispone di tecniche efficienti per il calcolo della antitrasformata. Si noti che non si devono determinare le radici del polinomio locatore degli errori.

Un altro metodo consiste nella ricerca delle radici del polinomio locatore, e quindi delle posizioni i_n degli errori. Nel caso di codici binari si può poi assumere $e_{i_n} = 1$ in ognuna delle posizioni. Per i codici non binari, una volta note le posizioni degli errori si risolve un sistema di ν equazioni *lineari*, scelte tra le $d-1$ disponibili,

$$E_j = \sum_{n=1}^{\nu} e_{i_n} \alpha^{j i_n} \tag{5.42}$$

che ha come incognite i *valori* degli errori e_{i_n}. Il metodo è utilizzabile solo per piccoli valori di ν.

---

che non è mai singolare. Poiché infine nel dominio del tempo la combinazione lineare non ha più di ν elementi non nulli, nel dominio delle frequenze non possono trovarsi ν elementi consecutivi nulli. Quindi l'indipendenza vale anche considerando solo ν colonne consecutive, anziché N

Un terzo metodo (forse il più comune) risulterà più chiaro operando nel solo *dominio del tempo*. Risulta comodo definire l'insieme delle *sindromi* $S_0 = E_{j_0}$, $S_1 = E_{j0+1}$, ..., $S_{2t-1} = E_{j_0+2t-1}$. Si ha poi

$$S_i = E_{j_0+i} = \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n(j_0+i)} \tag{5.43}$$

Si definisce poi il polinomio $S(x)$, detto anche polinomio delle sindromi[13],

$$S(x) = \sum_{i=0}^{2t-1} S_i x^i \tag{5.44}$$

In presenza di ν errori nelle posizioni i_n si ha

$$S(x) = \sum_{i=0}^{2t-1} \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n(j_0+i)} x^i = \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n j_0} \sum_{i=0}^{2t-1} (\alpha^{i_n} x)^i \tag{5.45}$$

Poiché se $\beta \neq 1$ anche in un campo finito si ha $\displaystyle\sum_{i=0}^{2t-1} \beta^i = \frac{1-\beta^{2t}}{1-\beta}$ e osservando che $1 - \alpha^{i_n} x$ è un fattore del polinomio locatore, si può moltiplicare per $\Lambda(x)$, ottenendo

$$S(x)\Lambda(x) = \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n j_0} (1 - (\alpha^{i_n} x)^{2t}) \prod_{j \neq n} (1 - \alpha^{i_j} x) \tag{5.46}$$

Se in questa uguaglianza si considerano solo i termini di grado minore di $2t$ si semplifica l'espressione senza perdere nulla: i termini di grado superiore sono sostanzialmente gli stessi, ripetuti. Si pone

$$\Omega(x) = S(x)\Lambda(x) \mod x^{2t} = \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n j_0} \prod_{j \neq n} (1 - \alpha^{i_j} x) \tag{5.47}$$

da cui si vede che $\Omega(x)$ ha grado $\leq \nu - 1$. Quindi tutti i termini dal grado ν al grado $2t - 1$ del prodotto $S(x)\Lambda(x)$ sono nulli. Si tratta della *key equation* già vista nel dominio della frequenza. Infatti il prodotto dei polinomi equivale alla convoluzione dei relativi coefficienti.

Si risolve la *key equation* con il metodo preferito, e si trovano il polinomio locatore $\Lambda(x)$ e le posizioni i_n degli errori. Poi si calcola

$$\Omega(x) = S(x)\Lambda(x) \mod x^{2t} \tag{5.48}$$

[13] c'è chi preferisce numerare le sindromi da 1 a $2t$, modificando di conseguenza le equazioni

e si osserva che dall'uguaglianza

$$\Omega(x) = \sum_{n=1}^{\nu} e_{i_n} \alpha^{i_n j_0} \prod_{j \neq n} (1 - \alpha^{i_j} x) \tag{5.49}$$

è possibile estrarre il valore e_{i_n} dell'errore in posizione i_n. Infatti ponendo $x = \alpha^{-i_n}$ si annullano tutte le produttorie tranne quella desiderata, e si ha

$$\Omega(\alpha^{-i_n}) = e_{i_n} \alpha^{i_n j_0} \prod_{j \neq n} (1 - \alpha^{i_j - i_n}) \tag{5.50}$$

C'è poi un metodo ingegnoso per semplificare il calcolo dell'ultima produttoria rimasta. Basta osservare che la derivata rispetto a x di $\Lambda(x)$, calcolata in α^{-i_n}, vale[14]

$$\Lambda'(\alpha^{-i_n}) = -\alpha^{i_n} \prod_{j \neq n} (1 - \alpha^{i_j - i_n}) \tag{5.51}$$

per ottenere infine

$$e_{i_n} = -\frac{\Omega(\alpha^{-i_n})}{\alpha^{i_n (j_0 - 1)} \Lambda'(\alpha^{-i_n})} \tag{5.52}$$

Nel calcolo della derivata rispetto a x del polinomio $\Lambda(x)$ si utilizzano le usuali regole di derivazione, ma con le regole del campo finito per i coefficienti. Ad esempio in GF(2^m) la derivata di x^n, anziché l'usuale nx^{n-1}, vale x^{n-1} se n è dispari e zero se n è pari. Quindi nel calcolo della derivata di $\Lambda(x)$ si considerano solo i termini di grado *dispari*.

In molti casi pratici $j_0 = 1$, e quindi a denominatore si ha solo $\Lambda'(\alpha^{-i_n})$. Nel caso, non infrequente, di $j_0 = 0$ a denominatore si ha $x\Lambda'(x)$, valutato in $x = \alpha^{-i_n}$. Si può osservare che

$$x\Lambda'(x) = \frac{d}{dx}(x\Lambda(x)) - \Lambda(x) \tag{5.53}$$

e inoltre che $\Lambda(\alpha^{-i_n}) = 0$. Valutando la derivata di $x\Lambda(x)$ si trovano i soli coefficienti di grado *pari* di $\Lambda(\alpha^{-i_n})$.

5.5.4 Alcune considerazioni pratiche

Esclusi i casi semplicissimi, la ricerca delle radici del polinomio locatore viene fatta per tentativi (ricerca di *Chien*): si provano *tutte* le N posizioni, valutando successivamente $\Lambda(\alpha^{-i})$ per $i = N-1, N-2, \ldots, 0$ (seguendo l'ordinamento delle cifre trasmesse), ovvero $\Lambda(\alpha), \Lambda(\alpha^2), \ldots, \Lambda(\alpha^N) = \Lambda(1)$. Nel calcolo si riutilizzano per quanto possibile i risultati

[14] $\frac{d}{dx}(x-x_1)(x-x_2)(x-x_3)\ldots = (x-x_2)(x-x_3)\ldots + (x-x_1)(x-x_3)\ldots + (x-x_1)(x-x_2)\ldots + \ldots$
Ponendo $x = x_1$ resta solo il primo termine

precedenti. Si consideri ad esempio un termine come $\Lambda_3 x^2$: avendo già calcolato al passo i-esimo $\Lambda_3(\alpha^i)^2$ basta moltiplicare per α^2 per ottenere $\Lambda_3(\alpha^{i+1})^2$.

Quando si trova una radice, nel caso non binario si valuta anche il valore e_{i_n} dell'errore. Per il calcolo di $\Lambda'(\alpha^{-i_n})$ e di $\Omega(\alpha^{-i_n})$ si usano tecniche analoghe.

Il calcolo delle sindromi risulta spesso la parte più pesante. Si può procedere in più modi.

La generica sindrome $E_j = v(\alpha^j)$ è pari al resto della divisione tra $v(x)$ e $(x-\alpha^j)$. Può essere calcolata in N passi con un circuito retroazionato che comprende un registro accumulatore e un moltiplicatore per α^j secondo la regola di *Horner* (o Ruffini):

$$\begin{aligned} v(\alpha^j) &= v_{N-1}(\alpha^j)^{N-1} + v_{N-2}(\alpha^j)^{N-2} + v_{N-3}(\alpha^j)^{N-3} + \ldots = \\ &= (\ldots((v_{N-1}\alpha^j + v_{N-2})\alpha^j + v_{N-3}) + \ldots) + v_0 \end{aligned} \tag{5.54}$$

Può essere conveniente calcolare *prima* il resto $r(x)$ della divisione di $v(x)$ per il polinomio generatore $g(x)$ e poi $r(\alpha^j)$. Infatti si ha $v(x) = q(x)g(x) + r(x)$, dove $q(x)$ è il quoziente. Poiché $g(\alpha^j) = 0$ si ottiene

$$E_j = v(\alpha^j) = r(\alpha^j) \tag{5.55}$$

Il vantaggio è notevole nel caso di codici binari, perché la divisione di $v(x)$ per $g(x)$ richiede solo semplici operazioni in GF(2). Una variante è la seguente: se $M_j(x)$ è il polinomio minimo di α^j, si divide $v(x)$ per $M_j(x)$, ottenendo un resto $r(x)$, e poi si valuta $r(\alpha^j)$.

Si possono calcolare le sindromi E_j anche come componenti della trasformata discreta della N-pla ricevuta, se si dispone di una implementazione veloce della trasformata di Fourier.

Conviene anche ricordare che vale la proprietà $E_j^q = E_{jq}$, utile tutte le volte che $q \neq q^m$. Ad esempio nel caso di codici binari $E_2 = E_1^2$, $E_4 = E_2^2, \ldots$ Ciò consente di valutare solo una parte delle sindromi, e trovare le altre di conseguenza.

5.5.5 Un semplice esempio

Si consideri un codice correttore di due errori, con $j_0 = 1$, e quindi con radici α, α^2, α^3 e α^4. Calcolate le sindromi $S_0 = v(\alpha), \ldots, S_3 = v(\alpha^4)$ se queste non sono tutte nulle si risolve il sistema di equazioni (*key equation*)

$$\begin{aligned} S_0\Lambda_2 + S_1\Lambda_1 &= S_2 \\ S_1\Lambda_2 + S_2\Lambda_1 &= S_3 \end{aligned} \tag{5.56}$$

La soluzione, ottenuta con uno qualsiasi dei metodi tradizionali, è

$$\Lambda_1 = \frac{S_1S_2 + S_0S_3}{S_0S_2 + S_1^2} \quad \Lambda_2 = \frac{S_2^2 + S_1S_3}{S_0S_2 + S_1^2} \tag{5.57}$$

se risulta $S_0S_2 + S_1^2 \neq 0$. Altrimenti le due equazioni risultano coincidenti (c'è un solo errore!) e si ottiene

$$\Lambda_1 = \frac{S_2}{S_1} \quad \Lambda_2 = 0 \tag{5.58}$$

Caso binario (codici BCH)

Si ha $S_1 = S_0^2$ e $S_3 = S_0^4$. Quindi si ottiene

$$\Lambda_1 = S_0 \quad \Lambda_2 = S_0^2 + \frac{S_2}{S_0} \tag{5.59}$$

se $S_2 \neq S_0^3$. Altrimenti risulta

$$\Lambda_1 = S_0 \quad \Lambda_2 = 0 \tag{5.60}$$

Non occorre determinare i valori degli errori.

Caso non binario (codici *Reed-Solomon*)

Si calcola anche $\Omega(x) = S(x)\Lambda(x) \bmod x^4$. Si ottiene

$$\Omega_0 = S_0 \quad \Omega_1 = S_1 + S_0\Lambda_1 \tag{5.61}$$

Si ha poi $\Lambda'(x) = \Lambda_1$ e quindi

$$e_{i_n} = \frac{\Omega_0 + \Omega_1\alpha^{-i_n}}{\Lambda_1} \tag{5.62}$$

5.6 Soluzione della *key equation*

Nel caso di pochi errori si può risolvere direttamente il sistema di equazioni lineari. Per codici correttori di molti errori sono disponibili l'*algoritmo di Euclide* e quello di *Berlekamp* o *Berlekamp-Massey*.

5.6.1 Algoritmo di Euclide

L'algoritmo di Euclide per la ricerca del massimo comun divisore di due polinomi fornisce un metodo efficiente per la soluzione della *key equation*

$$\Lambda(x)S(x) = \Omega(x) \bmod x^{2t} \tag{5.63}$$

dove t è il potere correttore del codice, $S(x) = S_0 + S_1 x + \cdots + S_{2t-1}x^{2t-1}$ è il polinomio delle sindromi, $\Lambda(x)$ e $\Omega(x)$ sono i polinomi locatore e valutatore degli errori. Si cerca una soluzione, se esiste, con $\deg \Omega(x) < \deg \Lambda(x) \leq t$. La *key equation* equivale a

$$\Phi(x)x^{2t} + \Lambda(x)S(x) = \Omega(x) \tag{5.64}$$

Per semplicità nel seguito si sottintenderà nei polinomi $\Phi(x)$, $\Lambda(x)$ e $\Omega(x)$ la dipendenza da x. E' immediato verificare che se si pone

$$\begin{aligned} \Phi_{-1} &= 1 \\ \Lambda_{-1} &= 0 \\ \Omega_{-1} &= x^{2t} \end{aligned} \tag{5.65}$$

si ha

$$\Phi_{-1}x^{2t} + \Lambda_{-1}S = \Omega_{-1} \tag{5.66}$$

e analogamente ponendo

$$\begin{aligned} \Phi_0 &= 0 \\ \Lambda_0 &= 1 \\ \Omega_0 &= S \end{aligned} \tag{5.67}$$

si ha

$$\Phi_0 x^{2t} + \Lambda_0 S = \Omega_0 \tag{5.68}$$

Le due soluzioni non hanno tuttavia il grado desiderato. Dividendo successivamente i polinomi Ω_{i-2} e Ω_{i-1} ($i = 1, \ldots$) si ottengono i quozienti

$$q_i = \frac{\Omega_{i-2}}{\Omega_{i-1}} \tag{5.69}$$

di grado $\deg q_i = \deg \Omega_{i-2} - \deg \Omega_{i-1} \geq 1$, e i resti

$$\Omega_i = \Omega_{i-2} - q_i \Omega_{i-1} \tag{5.70}$$

di grado $\deg \Omega_i \leq \deg \Omega_{i-1}$ che decresce ad ogni passo fino ad annullarsi, in $2t-1$ passi al massimo[15]. Se poi si definiscono

$$\Lambda_i = \Lambda_{i-2} - q_i \Lambda_{i-1} \tag{5.71}$$

[15]Se si procedesse fino a questo punto l'ultimo resto non nullo sarebbe il massimo comun divisore dei due polinomi iniziali (x^{2t} e S nel caso qui considerato); l'algoritmo proposto da Euclide per la ricerca del massimo comun divisore di due numeri interi funziona allo stesso modo per i polinomi

e

$$\Phi_i = \Phi_{i-2} - q_i \Phi_{i-1} \tag{5.72}$$

si ottiene una successione di polinomi Ω_i, Λ_i e Φ_i soluzioni della *key equation*. Infatti

$$\begin{aligned} &\Phi_i x^{2t} + \Lambda_i S = \Phi_{i-2} x^{2t} + \Lambda_{i-2} S - q_i(\Phi_{i-1} x^{2t} + \Lambda_{i-1} S) = \\ &= \Omega_{i-2} - q_i \Omega_{i-1} = \Omega_i \end{aligned} \tag{5.73}$$

Poiché $\deg q_i \geq 1$, il grado di Λ_i è determinato da $q_i \Lambda_{i-1}$ e cresce ad ogni passo:

$$\deg \Lambda_i = (\deg \Omega_{i-2} - \deg \Omega_{i-1}) + \deg \Lambda_{i-1} > \deg \Lambda_{i-1} \tag{5.74}$$

Da questa si ottiene, recursivamente,

$$\deg \Lambda_i + \deg \Omega_{i-1} = \deg \Lambda_{i-1} + \deg \Omega_{i-2} = 2t \tag{5.75}$$

Poiché il grado di Ω_i decresce ad ogni passo, si ha anche

$$\deg \Lambda_i + \deg \Omega_i < 2t \tag{5.76}$$

Si procede con l'algoritmo fino a quando, per la prima volta,

$$\deg \Omega_i < t \tag{5.77}$$

Poiché $\deg \Omega_{i-1} \geq t$ si ha anche

$$\deg \Lambda_i \leq t \tag{5.78}$$

Questa potrebbe essere la soluzione desiderata[16]. E comunque se si procedesse oltre si avrebbe $\deg \Lambda_{i+1} = 2t - \deg \Omega_i > t$, che non è certamente la soluzione. Resta da mostrare che una soluzione con $\deg \Omega < \deg \Lambda \leq t$, se esiste, coincide con quella trovata al passo i-esimo. Si osservi che

$$\Phi_0 \Lambda_{-1} - \Phi_{-1} \Lambda_0 = -1 \tag{5.79}$$

e che quindi, recursivamente,

$$\begin{aligned} &\Phi_i \Lambda_{i-1} - \Phi_{i-1} \Lambda_i = (\Phi_{i-2} - q_i \Phi_{i-1}) \Lambda_{i-1} - \Phi_{i-1}(\Lambda_{i-2} - q_i \Lambda_{i-1}) = \\ &= -(\Phi_{i-1} \Lambda_{i-2} - \Phi_{i-2} \Lambda_{i-1}) = (-1)^{i+1} \end{aligned} \tag{5.80}$$

Dunque i polinomi Φ_i e Λ_i non hanno fattori comuni. Sia ora

$$\Phi x^{2t} + \Lambda S = \Omega \tag{5.81}$$

[16]Si noti che l'esecuzione dell'algoritmo di Euclide non richiede l'aggiornamento dei polinomi Φ_i, che occorre solo per dimostrare che il metodo funziona correttamente

la soluzione desiderata della *key equation*, con $\deg\Omega < \deg\Lambda \leq t$ e sia

$$\Phi_i x^{2t} + \Lambda_i S = \Omega_i \tag{5.82}$$

la soluzione trovata dall'algoritmo di Euclide al primo valore di i per cui risulta $\deg\Omega_i \leq \deg\Omega$. Si osservi che $\deg\Omega_{i-1} > \deg\Omega_i$, e quindi $\deg\Lambda_i < 2t - \deg\Omega$. Moltiplicando le (5.81) e (5.82) rispettivamente per Λ_i e Λ si ottiene

$$\Lambda_i\Phi x^{2t} + \Lambda_i\Lambda S = \Lambda_i\Omega \tag{5.83}$$

$$\Lambda\Phi_i x^{2t} + \Lambda\Lambda_i S = \Lambda\Omega_i \tag{5.84}$$

Notando che

$$\deg(\Lambda_i\Omega) = \deg\Lambda_i + \deg\Omega < 2t \tag{5.85}$$

$$\deg(\Lambda\Omega_i) = \deg\Lambda + \deg\Omega_i < 2t \tag{5.86}$$

e confrontando i termini di ugual grado nelle (5.83) e (5.84) si ottiene

$$\Lambda_i\Omega = \Lambda\Omega_i \tag{5.87}$$

$$\Lambda_i\Phi = \Lambda\Phi_i \tag{5.88}$$

e poiché Λ_i e Φ_i sono polinomi primi tra loro

$$\Lambda = \lambda(x)\Lambda_i \tag{5.89}$$

$$\Phi = \lambda(x)\Phi_i \tag{5.90}$$

e infine, dalla (5.81),

$$\Omega = \lambda(x)\Omega_i \tag{5.91}$$

Dunque l'algoritmo di Euclide determina la soluzione desiderata, a meno di un eventuale fattore comune $\lambda(x)$ che non viene individuato. Nel caso di correzione di soli errori $\lambda(x)$ deve però essere una costante. Infatti Λ e Ω non hanno fattori comuni; altrimenti si avrebbero radici del polinomio locatore in cui il polinomio valutatore è nullo, cioè errori nulli.

5.6.2 Algoritmo di *Berlekamp-Massey*

La soluzione della *key equation*

$$E_k = -\sum_{j=1}^{\nu} \Lambda_j E_{k-j} \tag{5.92}$$

equivale a determinare il registro a scorrimento della minor lunghezza possibile (retroazionato con guadagni Λ_j) in grado di riprodurre le prime $2t$ sindromi (ed eventualmente tutte le altre). Si descriverà una possibile formulazione dell'algoritmo, senza dimostrarne l'ottimalità (compito tutt'altro che banale).

Le inizializzazioni sono:

$n = 0$	iterazione corrente
$L = 0$	lunghezza del registro corrente
$\Lambda(x) = 1$	polinomio locatore (registro) corrente
Λ_i	coefficiente i-esimo di $\Lambda(x)$
ν	grado del polinomio locatore corrente
$D(x) = x$	termine correttivo (se il registro corrente fallisce)

Si calcola la *discrepanza*, ovvero l'eventuale errore commesso dal registro attuale nel prevedere la prossima sindrome:

$$d = \sum_{i=0}^{\nu} \Lambda_i S_{n-i} \tag{5.93}$$

Se $d_n \neq 0$ si calcola il nuovo $\Lambda(x)$:

$$\Lambda^*(x) = \Lambda(x) - dD(x) \quad \text{(polinomio locatore modificato)} \tag{5.94}$$

Se inoltre $n \geq 2L$ si devono modificare anche $D(x)$ e L:

$$D(x) = \frac{\Lambda(x)}{d} \quad \text{(termine correttivo aggiornato, usando il } \Lambda(x) \text{ precedente)} \tag{5.95}$$

$$L = n - L + 1 \quad \text{(lunghezza del registro aggiornato)} \tag{5.96}$$

A questo punto si aggiorna effettivamente $\Lambda(x)$

$$\Lambda(x) = \Lambda^*(x) \tag{5.97}$$

Infine si incrementa n (unica operazione da eseguire nel caso $d = 0$). L'algoritmo è concluso se $n = 2t$. Altrimenti si prosegue, dopo aver posto

$$D(x) = xD(x) \quad \text{(spostamento di una posizione)} \tag{5.98}$$

L'algoritmo di *Berlekamp-Massey*, a differenza di quello di Euclide, non prevede di calcolare $\Omega(x)$ (che non è richiesto per i codici binari). Se tuttavia si desidera calcolare anche $\Omega(x)$ *durante le iterazioni*[17], come nell'algoritmo originale di *Berlekamp*, si pone inizialmente $\Omega(x) = 0$ e $A(x) = -1$ e si aggiornano entrambi questi polinomi con le stesse regole utilizzate per $\Lambda(x)$ e $D(x)$.

Nel caso binario si può dimostrare che la discrepanza d è nulla per tutti i valori dispari di n. Si può quindi saltare metà delle iterazioni, semplicemente moltiplicando $D(x)$ per x^2.

[17] si potrebbe calcolare $\Omega(x)$ alla fine: $\Omega(x) = S(x)\Lambda(x) \bmod x^{2t}$

5.6.3 Possibili controlli finali sulla soluzione

Sono possibili diversi controlli sul superamento della capacità di correzione del codice. Questi consentono, quasi sempre, di evitare correzioni sbagliate (cioè di introdurre ulteriori errori). Alcuni possibili controlli sono:

- il numero di radici del polinomio locatore deve coincidere con il grado
- nel caso di codici binari, se si calcolano i valori degli errori si deve trovare $e_{i_n} = 1$ in ogni posizione
- la recursione $E_k = -\sum_{j=1}^{\nu} \Lambda_j E_{k-j}$, dopo un giro completo, deve ridare i valori di partenza $E_{j_0}, \ldots, E_{j_0+d-2}$
- se si calcolano sia $\Lambda(x)$ sia $\Omega(x)$ deve essere $\deg \Omega(x) < \deg \Lambda(x) \leq t$
- se la distanza d *progettata* del codice è *pari* è disponibile una equazione in più, che non aumenta il potere correttore t del codice ma può essere utilizzata come ulteriore controllo; in tal caso nelle equazioni e negli algoritmi basta sostituire $d-1$ (dispari) a $2t$

5.6.4 Correzione di errori e cancellazioni

In taluni casi alcuni simboli ricevuti sono *cancellati*, o perché effettivamente non ricevuti o perché talmente incerti che si preferisce non tentare di interpretarli. Ai fini della decodifica si attribuisce ai simboli cancellati un valore provvisorio, arbitrario (ad esempio nullo). Se si hanno e *cancellazioni* nelle posizioni $j_1, j_2 \ldots, j_e$ si sono introdotti fino a e errori addizionali, da determinare. A differenza dei veri errori, tuttavia, sono già note le posizioni.

Se si definisce il *polinomio locatore delle cancellazioni*

$$\Gamma(x) = \prod_{n=1}^{e} (1 - x\alpha^{j_n}) = \Gamma_0 + \Gamma_1 x + \ldots + \Gamma_e x^e \tag{5.99}$$

il prodotto $\Psi(x) = \Lambda(x)\Gamma(x)$ individua tutte le posizioni in cui possono esservi errori, ed è detto *polinomio locatore degli errori e delle cancellazioni*. Ripercorrendo senza modifiche il percorso che porta al polinomio $\Omega(x)$ valutatore degli errori, con le sole sostituzioni di $\Lambda(x)\Gamma(x)$ a $\Lambda(x)$ e di errori e cancellazioni ai soli errori, si trova la *key equation*[18]

$$\Omega(x) = S(x)\Lambda(x)\Gamma(x) \bmod x^{d-1} \tag{5.100}$$

Le incognite $\Lambda(x)$ e $\Omega(x)$ hanno grado rispettivamente ν e $\leq \nu + e - 1$. Quindi tutti i termini dal grado $\nu + e$ al grado $d-2$ del prodotto $S(x)\Lambda(x)\Gamma(x)$ sono nulli. Si tratta di

[18] per maggior generalità non si suppone che d sia dispari, e quindi si sostituisce $d-1$ a $2t$

$d-1-e-\nu$ equazioni, che consentono di determinare i ν coefficienti incogniti di $\Lambda(x)$ se[19] $2\nu+e \leq d-1$. Ai fini della decodifica *due cancellazioni* equivalgono ad *un errore*. Ciò non sorprende se si pensa che per correggere un errore occorre determinarne posizione e valore, mentre di una cancellazione basta il valore. Ecco quindi perché è preferibile *cancellare* il simbolo ricevuto, nei casi troppo incerti.

Ai fini della soluzione della *key equation* basta calcolare il prodotto $S(x)\Gamma(x)$ e considerarlo come un *polinomio delle sindromi modificato*. Poiché nella *key equation* i termini di grado $\geq d-1$ sono ininfluenti, si può definire il polinomio delle sindromi modificato come $T(x) = S(x)\Gamma(x) \bmod x^{d-1}$.

Infine per il calcolo dei *valori* degli errori e delle cancellazioni si procede normalmente. Ovviamente al posto del polinomio locatore degli errori $\Lambda(x)$ si utilizza il polinomio $\Psi(x)$ locatore degli errori e delle cancellazioni.

Errori e cancellazioni con l'algoritmo di Euclide

Si calcola il polinomio $\Gamma(x)$ a partire dalle posizioni note delle cancellazioni, poi le sindromi modificate $T(x) = S(x)\Gamma(x) \bmod x^{d-1}$. Nelle inizializzazioni si pone $\Lambda_0(x) = \Gamma(x)$, $\Omega_{-1}(x) = x^{d-1}$ e $\Omega_0(x) = T(x)$. Si procede normalmente fino a quando, per la prima volta, $\deg \Omega < (d-1+e)/2$.

Errori e cancellazioni con l'algoritmo di *Berlekamp-Massey*

Si calcola il polinomio $\Gamma(x)$ a partire dalle posizioni note delle cancellazioni, poi le sindromi modificate $T(x) = S(x)\Gamma(x) \bmod x^{d-1}$. Si possono poi usare le sindromi modificate $T(x)$, procedendo normalmente, fino a $n = d-1$, senza altre modifiche.

Tuttavia si può dimostrare che con le inizializzazioni $n = e$ e $\Lambda(x) = D(x) = \Gamma(x)$ l'algoritmo di *Berlekamp-Massey* calcola direttamente il polinomio $\Psi(x)$ locatore di errori e cancellazioni *senza* richiedere esplicitamente le sindromi modificate $T(x)$, e cioè utilizzando come sindromi $S(x)$. Tuttavia questa versione non riesce a calcolare anche $\Omega(x)$, che viene determinato alla fine dalla relazione $\Omega(x) = S(x)\Psi(x) \bmod x^{d-1}$.

[19] il risultato è evidente anche *nel dominio delle frequenze*: $S(x)$ contiene $d-1$ valori noti (consecutivi) della trasformata dell'errore, mentre del prodotto $S(x)\Gamma(x)$ sono noti solo $d-1-e$ coefficienti. Infatti al prodotto dei polinomi corrisponde la convoluzione dei coefficienti; essendo e il grado di $\Gamma(x)$, e coefficienti del prodotto sono *inquinati* da valori *non noti* della trasformata dell'errore

Capitolo 6

Complementi sui codici a blocco

6.1 Decodifica a massima verosimiglianza dei codici a blocco

Il calcolo esaustivo delle 2^K correlazioni con tutti i segnali è proponibile solo per codici molto semplici. Un metodo generale, almeno in linea di principio, per decodificare in modo ottimale i codici a blocco è di far corrispondere le parole di codice ai rami di un traliccio, come per i codici convoluzionali.

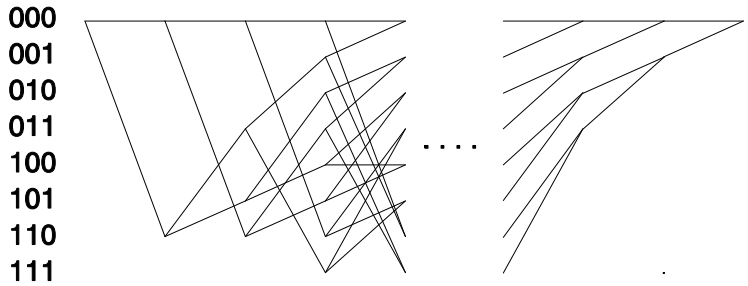

Figura 6.1: Traliccio del codice di *Hamming* (7,4)

Per semplicità di esposizione ci si limiterà ai codici ciclici, eventualmente accorciati, il cui codificatore ha una struttura come quella di Fig. 5.1, corrispondente al codice di *Hamming* con $N = 15$ e $K = 11$. Definendo *stato* il contenuto degli $N - K$ registri è facile tracciare il diagramma a traliccio del codice, con un numero di stati pari a 2^{N-K}. Per semplicità in Fig. 6.1 è mostrato il traliccio a otto stati del codice di *Hamming* (7,4). Ovviamente ad ogni transizione di stato corrisponde un solo bit trasmesso. Il ramo superiore corrisponde alla trasmissione di uno zero per le transizioni uscenti dagli stati ..0 e di un uno per gli stati ..1. Gli ultimi passi corrispondono all'emissione delle cifre di parità. In totale si hanno N passi.

Il traliccio di un codice a blocco non ha una struttura così regolare come quella dei codici

convoluzionali. Si può ugualmente utilizzare l'algoritmo di Viterbi per la ricerca del percorso nel traliccio, cioè della parola di codice, a massima verosimiglianza. E' evidente che ci si deve limitare a codici con un piccolo numero di cifre di parità, perché il numero di stati sia trattabile.

6.2 Trasformata di *Hadamard*

In molti casi il calcolo delle prestazioni di un codice a blocco richiede la conoscenza della distribuzione dei pesi delle parole del codice. Le proprietà della trasformata di *Hadamard* consentono di ottenere un notevole teorema che collega la distribuzione dei pesi delle parole di un codice a blocco a quella del codice duale. Se il *rate* $R = K/N$ del codice è maggiore di $1/2$, come quasi sempre avviene, risulta più facile l'indagine sul codice duale. Quando non sia possibile ottenere né la distribuzione dei pesi del codice né quella del duale si ricorre ad approssimazioni.

La trasformata di *Hadamard* consente anche di ottenere, quando siano note (e non troppo numerose) le parole del codice duale, una notevole espressione per le probabilità a posteriori di ciascun bit della parola di un codice a blocco. Tali probabilità sono particolarmente utili per la decodifica iterativa di codici prodotto (una particolare forma di *turbo codici*).

Siano $\mathbf{u}$ e $\mathbf{v}$ N-ple binarie, e si definisca il prodotto scalare

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=0}^{N-1} u_i v_i \tag{6.1}$$

valutato con l'algebra binaria di GF(2). Il prodotto scalare vale 0 (nel qual caso $\mathbf{u}$ e $\mathbf{v}$ si dicono *ortogonali*) oppure 1.

Se si considerano tutte le 2^N possibili N-ple $\mathbf{u}$ e $\mathbf{v}$, la matrice di *Hadamard* di dimensione $2^N \cdot 2^N$ con elementi pari a ± 1, data da[1]

$$H = (-1)^{\mathbf{u}\cdot\mathbf{v}} \tag{6.2}$$

ha tutte le righe e le colonne ortogonali (nel senso usuale; la somma algebrica dei prodotti è nulla). Infatti se si considerano due *diverse* righe di H, ottenute ad esempio da $\mathbf{u}^{(m)}$ e $\mathbf{u}^{(l)}$ è facile riconoscere che il prodotto scalare $(\mathbf{u}^{(m)} + \mathbf{u}^{(l)}) \cdot \mathbf{v}$ vale 0 per metà dei $\mathbf{v}$ e 1 per l'altra metà[2]. Quindi metà dei prodotti scalari $\mathbf{u}^{(m)} \cdot \mathbf{v}$ e $\mathbf{u}^{(l)} \cdot \mathbf{v}$ coincidono e metà sono diversi. Analogo risultato si ottiene per le colonne. Il prodotto scalare di una riga o colonna di H per sé stessa vale 2^N, e quindi si ha

$$HH^T = H^T H = 2^N I \tag{6.3}$$

[1] è consuetudine sottintendere gli indici che enumerano $\mathbf{u}$ e $\mathbf{v}$ da 1 a 2^N; scelto un ordinamento qualsiasi delle N-ple $\mathbf{u}$ e $\mathbf{v}$, sarebbe più corretto scrivere $h_{mn} = (-1)^{\mathbf{u}^{(m)}\cdot\mathbf{v}^{(n)}}$ $(m, n = 1 \ldots 2^N)$

[2] esiste almeno una componente non nulla di $\mathbf{u}^{(m)} + \mathbf{u}^{(l)}$; sia j la sua posizione nella N-pla; metà delle N-ple $\mathbf{v}$ hanno $v_j = 0$ e metà $v_j = 1$, quindi metà dei prodotti scalari danno 0 e metà danno 1 (o viceversa)

Se $\mathbf{u}$ è l'insieme di tutte le N-ple e $f(\mathbf{u})$ una funzione qualsiasi[3] di $\mathbf{u}$ si ha

$$H^T H f(\mathbf{u}) = 2^N f(\mathbf{u}) \tag{6.4}$$

E' quindi naturale definire trasformata di *Hadamard* (o di *Fourier*, o di *Walsh*) il prodotto $Hf(\mathbf{u})$; la moltiplicazione per H^T e per la costante $1/2^N$ costituisce la formula di antitrasformazione. Si usa scrivere la trasformata nella forma[4]

$$F(\mathbf{v}) = \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{v}} f(\mathbf{u}) \tag{6.5}$$

e la corrispondente formula di antitrasformazione come

$$f(\mathbf{u}) = \frac{1}{2^N} \sum_{\mathbf{v}} (-1)^{\mathbf{u}\cdot\mathbf{v}} F(\mathbf{v}) \tag{6.6}$$

Per il calcolo effettivo, si può mostrare che esiste una trasformata di *Hadamard* veloce (FHT) con struttura simile alla FFT.

La trasformata di *Hadamard* ha una notevole proprietà, relativamente ai codici lineari: se si considerano le N-ple $\mathbf{u} \in C$ e $\mathbf{v} \in C'$, dove C' è il codice duale[5] di C, si ha

$$\sum_{\mathbf{u}\in C} f(\mathbf{u}) = \frac{1}{|C'|} \sum_{\mathbf{v}\in C'} F(\mathbf{v}) \tag{6.7}$$

dove $|C'|$ è il numero di parole del codice C'. Infatti si ha

$$\sum_{\mathbf{v}\in C'} F(\mathbf{v}) = \sum_{\mathbf{v}\in C'} \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{v}} f(\mathbf{u}) = \sum_{\mathbf{u}} f(\mathbf{u}) \sum_{\mathbf{v}\in C'} (-1)^{\mathbf{u}\cdot\mathbf{v}} \tag{6.8}$$

Basta poi osservare che per ogni $\mathbf{u} \in C$ si ha $\mathbf{u} \cdot \mathbf{v} = 0$, e quindi $f(\mathbf{u})$ viene sommato $|C'|$ volte; se invece $\mathbf{u} \notin C$, $\mathbf{u} \cdot \mathbf{v}$ vale 0 metà delle volte e 1 l'altra metà, e quindi la somma interna è nulla.

Nella (6.7) si possono ovviamente scambiare C e C', poiché C è il codice duale di C'.

6.3 Teorema di *MacWilliams*

Un esempio notevole di applicazione di questa proprietà è il teorema di *MacWilliams*, che consente di determinare la funzione enumeratrice dei pesi di un codice da quella del duale. Se per la generica N-pla $\mathbf{u}$ si pone

$$f(\mathbf{u}) = x^{N-w(\mathbf{u})} y^{w(\mathbf{u})} \tag{6.9}$$

[3] è solo richiesto che $f(\mathbf{u})$ sia definita per ogni N-pla $\mathbf{u}$ e che siano definiti somma e differenza; per il resto la funzione è del tutto arbitraria

[4] il prodotto $Hf(\mathbf{u})$ fornisce tutti i valori di $F(\mathbf{v})$; l'espressione $F(\mathbf{v}) = \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{v}} f(\mathbf{u})$ fornisce il valore di $F(\mathbf{v})$ per una specifica N-pla $\mathbf{v}$

[5] il codice duale C' contiene tutte e sole le N-ple ortogonali alle parole del codice C

dove $w(\mathbf{u})$ è il numero di uni contenuto nella N-pla $\mathbf{u}$, si ha

$$F(\mathbf{v}) = \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{v}} x^{N-w(\mathbf{u})} y^{w(\mathbf{u})} = \sum_{u_0=0}^{1} \cdots \sum_{u_{N-1}=0}^{1} \prod_{i=0}^{N-1} (-1)^{u_i v_i} x^{1-w(u_i)} y^{w(u_i)} = \\ = \prod_{i=0}^{N-1} \sum_{u_i=0}^{1} (-1)^{u_i v_i} x^{1-u_i} y^{u_i} \tag{6.10}$$

L'ultima somma vale $x+y$ tutte le volte che $v_i = 0$ e $x-y$ tutte le volte che $v_i = 1$. Quindi

$$F(\mathbf{v}) = (x+y)^{N-w(\mathbf{v})}(x-y)^{w(\mathbf{v})} \tag{6.11}$$

Infine si ottiene

$$\sum_{\mathbf{u}\in C} x^{N-w(\mathbf{u})} y^{w(\mathbf{u})} = \frac{1}{|C'|} \sum_{\mathbf{v}\in C'} (x+y)^{N-w(\mathbf{v})}(x-y)^{w(\mathbf{v})} \tag{6.12}$$

Ora basta riconoscere che le due somme enumerano i pesi delle parole dei codici C' e C. Infatti indicando con A_i il numero di parole del codice C di peso i e con $A(x,y) = \sum_i A_i x^{N-i} y^i$ la funzione enumeratrice del codice C, ed analogamente con B_i il numero di parole del codice C' di peso i e con $B(x,y) = \sum_i B_i x^{N-i} y^i$ la funzione enumeratrice del codice C' si può riscrivere la (6.12) nella forma

$$A(x,y) = 2^{-(N-K)} B(x+y, x-y) \tag{6.13}$$

dove K è il numero di cifre d'informazione del codice C, oppure nella forma

$$B(x,y) = 2^{-K} A(x+y, x-y) \tag{6.14}$$

Benché sia più comodo lavorare con polinomi omogenei, una delle variabili della coppia x, y è ridondante; infatti se si pone

$$A(x) = \sum_i A_i x^i \tag{6.15}$$

e

$$B(x) = \sum_i B_i x^i \tag{6.16}$$

si ricava facilmente

$$B(x) = 2^{-K}(1+x)^N A(\frac{1-x}{1+x}) \tag{6.17}$$

oppure

$$A(x) = 2^{-(N-K)}(1+x)^N B(\frac{1-x}{1+x}) \tag{6.18}$$

6.3.1 Esempi

Il codice duale del codice di *Hamming* di lunghezza N ha come parole la N-pla nulla e tutte le versioni ruotate ciclicamente di un periodo della sequenza pseudocasuale di lunghezza N. Si ha quindi

$$B(x) = 1 + Nx^{\frac{N+1}{2}} \tag{6.19}$$

da cui si ottiene, utilizzando il teorema di *MacWilliams*,

$$A(x) = \frac{1}{N+1}\left[(1+x)^N + N(1+x)^{\frac{N-1}{2}}(1-x)^{\frac{N+1}{2}}\right] \tag{6.20}$$

Ad esempio per $N = 7$ si ottiene, svolgendo le potenze,

$$A(x) = 1 + 7x^3 + 7x^4 + x^7 \tag{6.21}$$

ma il risultato si complica rapidamente all'aumentare di N. Ad esempio per $N = 15$ si ha

$$A(x) = 1+35x^3+105x^4+168x^5+280x^6+435x^7+435x^8+280x^9+168x^{10}+105x^{11}+\ldots \tag{6.22}$$

Anche la funzione generatrice dei codici duali dei *BCH* correttori di due e tre errori è nota.

Nei codici estesi tutti i pesi i dispari crescono di una unità: è facile vedere che $A(x)$ viene sostituito da $[(1+x)A(x) + (1-x)A(-x)]/2$. Ad esempio è facile verificare che i codici di *Hamming* estesi hanno funzione generatrice

$$A(x) = \frac{1}{2N}\left[(1+x)^N + (1-x)^N + 2(N-1)(1-x^2)^{\frac{N}{2}}\right] \tag{6.23}$$

6.3.2 Distribuzione approssimata dei pesi

Quando non risulta possibile determinare esattamente la distribuzione dei pesi si ricorre ad approssimazioni. La più comune è assumere la stessa distribuzione che si avrebbe generando in modo casuale le parole del codice (con simboli equiprobabili). Si utilizza l'informazione sulla distanza minima, se disponibile, ponendo $A_0 = 1$, $A_1, \ldots, A_{d-1} = 0$. Per gli altri valori di i si calcola il numero di N-ple distinte di peso i e si divide per q^{N-K}, dove q è la dimensione dell'alfabeto. Infatti il codice utilizza solo q^K delle q^N N-ple disponibili. Il risultato è quindi

$$A_i \approx \binom{N}{i}(q-1)^i q^{-(N-K)} \tag{6.24}$$

Si noti che $\sum A_i \approx q^K$, come deve essere. Nel caso binario ($q = 2$) i pesi A_i sono quindi proporzionali ai coefficienti binomiali. Le tabelle 6.1 e 6.2 mostrano questa approssimazione binomiale a confronto con il risultato esatto per il codice di *Hamming* (127,120) e per il codice *BCH* (127,113).

i	A_i	A_i^B
3	2667	$2.604 \cdot 10^3$
4	82677	$8.074 \cdot 10^4$
5	$1.984 \cdot 10^6$	$1.986 \cdot 10^6$
6	$4.035 \cdot 10^7$	$4.039 \cdot 10^7$
...	...	...
63	$9.356 \cdot 10^{34}$	$9.356 \cdot 10^{34}$
...	...	...

Tabella 6.1: Confronto tra distribuzione esatta A_i dei pesi e approssimazione binomiale A_i^B per il codice di *Hamming* (127,120)

i	A_i	A_i^B
5	16002	$1.552 \cdot 10^4$
6	325374	$3.155 \cdot 10^5$
7	$5.456 \cdot 10^6$	$5.454 \cdot 10^6$
8	$8.183 \cdot 10^7$	$8.181 \cdot 10^7$
...	...	...
63	$7.309 \cdot 10^{32}$	$7.309 \cdot 10^{32}$
...	...	...

Tabella 6.2: Confronto tra distribuzione esatta A_i dei pesi e approssimazione binomiale A_i^B per il codice *BCH* (127,113)

6.3.3 Distribuzione dei pesi dei codici *Reed-Solomon*

La distribuzione dei pesi è nota per tutti i codici *Reed-Solomon*, anche accorciati. Se q è la dimensione dell'alfabeto e $d = N - K + 1$ è la distanza minima del codice, oltre alla parola nulla si hanno[6]

$$A_i = \binom{N}{i}(q-1)\sum_{j=0}^{i-d}(-1)^j\binom{i-1}{j}q^{i-d-j} \tag{6.25}$$

parole di peso $i \geq d$.

Ad esempio il codice *Reed-Solomon* (255,239) ha ben $3.387 \cdot 10^{28}$ parole non nulle di peso minimo, pari a $d = 17$. Se le 256^{239} parole del codice fossero scelte a caso, se ne otterrebbero in media $3.182 \cdot 10^{28}$ di peso 17.

[6]la dimostrazione non è semplice, e viene omessa

6.3.4 Codici binari utilizzati come rivelatori

Si consideri, per semplicità, un codice *binario* con distanza minima d e si supponga che sia utilizzato per la sola rivelazione di errori. Si supponga il canale binario simmetrico, ovvero con probabilità P di errore indipendente dal bit trasmesso. Si suppongano inoltre gli errori indipendenti. Ai fini del calcolo delle prestazioni del codice si può supporre che sia stata trasmessa la parola di tutti zeri. La probabilità di decodifica corretta coincide con la probabilità di zero errori, ovvero[7]

$$P_{cd} = (1-P)^N \tag{6.26}$$

Se A_i è la distribuzione dei pesi delle parole del codice la probabilità di mancata rivelazione di errori è data dalla probabilità che venga ricevuta una qualsiasi parola non nulla del codice, ovvero

$$P_{icd} = \sum_{i=d}^{N} A_i P^i (1-P)^{N-i} \tag{6.27}$$

Si noti che è richiesta la conoscenza dei pesi delle parole del codice, a meno che sia $P = 1/2$ (il canale peggiore possibile). In tal caso si ha

$$P_{icd} = \sum_{i=d}^{N} A_i \frac{1}{2^N} = 2^{-(N-K)} - 2^{-N} \approx 2^{-(N-K)} \tag{6.28}$$

relazione quasi ovvia se si pensa che gli $N-K$ bit di parità scelti casualmente (e indipendentemente) devono coincidere con quelli calcolati dalle cifre d'informazione (casuali). Per molti codici il caso $P = 1/2$ è il peggiore dal punto di vista della probabilità di mancata rivelazione, e quindi $2^{-(N-K)}$ è una utile maggiorazione. Ad esempio con $N-K = 32$ bit di parità si avrebbe $P_{icd} \leq 2.3 \cdot 10^{-10}$ per *qualunque* rapporto segnale-rumore. Tuttavia sono noti codici (in particolare accorciati) per cui P_{icd} può superare anche di diversi ordini di grandezza $2^{-(N-K)}$ (per $P < 1/2$) e quindi occorre maggior cautela.

Se è nota la funzione generatrice $B(x)$ dei pesi del codice duale conviene osservare che si ottiene facilmente

$$P_{icd} = (1-P)^N \left[A\left(\frac{P}{1-P}\right) - 1 \right] \tag{6.29}$$

e dal teorema di *MacWilliams* si ha anche

$$A\left(\frac{P}{1-P}\right) = 2^{-(N-K)} (1-P)^{-N} B(1-2P) \tag{6.30}$$

e quindi infine l'espressione

$$P_{icd} = 2^{-(N-K)} B(1-2P) - (1-P)^N \tag{6.31}$$

[7] i pedici *cd*, *icd* ed *ed* significano *correct decoding*, *incorrect decoding* ed *error detection*

particolarmente apprezzabile perché non richiede di determinare esplicitamente gli A_i.

Infine la probabilità P_{ed} di rivelazione di errori (e quindi, solitamente, di richiesta di ritrasmissione) si ottiene per differenza:

$$P_{ed} = 1 - P_{cd} - P_{icd} \tag{6.32}$$

6.3.5 Codici utilizzati come rivelatori e correttori

Il calcolo delle prestazioni dei codici utilizzati anche (o solo) come correttori di errori è ancora basato sulla conoscenza dei pesi delle parole del codice. In questa sezione si considerano codici anche non binari, e canali che commettono errori indipendenti. Tuttavia si considerano solo canali *simmetrici* che commettono errore a favore dei $q-1$ simboli errati con *uguale probabilità*. Con questa ipotesi, purtroppo non sempre verificata, se P è la probabilità d'errore sul canale, la probabilità di ricevere una specifica N-pla contenente k errori è data dalla semplice espressione

$$P(k) = \left(\frac{P}{q-1}\right)^k (1-P)^{N-k} \tag{6.33}$$

che dipende solo dal numero di errori, mentre per un canale non simmetrico dipenderebbe anche dalla specifica N-pla errata.

Il codice può essere utilizzato per correggere fino a $C \leq t$ errori, con $2t+1 \leq d$. Se $C < t$ la restante ridondanza viene usata per ridurre la probabilità di mancata rivelazione. Naturalmente ciò ha senso solo se accettare blocchi con errori è gravemente penalizzante, e se è possibile chiedere la ritrasmissione del pacchetto. Se invece si deve comunque produrre un blocco di K simboli decodificati conviene evidentemente utilizzare la piena capacità di correzione del codice ($C = t$).

La probabilità di decodifica corretta coincide con la probabilità di un numero di errori non superiore a C, ovvero

$$P_{cd} = \sum_{k=0}^{C} \binom{N}{k} P^k (1-P)^{N-k} \tag{6.34}$$

La probabilità di mancata rivelazione richiede un calcolo molto più complesso. Indicando ancora con $k \leq N$ il numero di errori, si decodifica in modo errato se la N-pla ricevuta è a distanza $s \leq C$ da una parola di codice non nulla, il cui peso i può variare da d a N. Si sommeranno dunque le probabilità $P(k)$ per tutti i possibili valori di k, s e i. Resta da calcolare il numero delle N-ple di peso k a distanza s da una qualsiasi parola di codice di peso i. Evidentemente k deve essere compreso tra $i-s$ e $i+s$. Inoltre la N-pla ricevuta può essere a distanza s da *non più di una* parola di codice. Infatti le parole di codice distano tra loro di almeno $d > 2t \geq 2s$. Dunque basta contare quante parole di codice hanno peso i ed in quanti modi diversi modificandone s simboli si possono ottenere parole

di peso k. Tutte le N-ple così ottenute sono distinte, ciascuna ha probabilità $P(k)$ e tutte producono errori non rivelati. Il risultato del calcolo, non semplice, è[8]

$$P_{icd} = \sum_{i=d}^{N} \sum_{s=0}^{C} \sum_{k=i-s}^{i+s} \sum_{h=h_1}^{h_2} A_i \left(\frac{P}{q-1}\right)^k (1-P)^{N-k} \binom{N-i}{h} (q-1)^h \binom{i}{i+h-k} \binom{k-h}{s+k-i-2h} (q-2)^{s+k-i-2h} \tag{6.35}$$

dove

$$h_1 = \max(0, k-i) \quad h_2 = \frac{s+k-i}{2} \tag{6.36}$$

che nel caso binario si semplifica in[9]

$$P_{icd} = \sum_{i=d}^{N} \sum_{s=0}^{C} \sum_{h=0}^{s} A_i P^{2h+i-s} (1-P)^{N-2h-i+s} \binom{N-i}{h} \binom{i}{s-h} \tag{6.37}$$

Infine la probabilità P_{ed} di rivelazione di errori si ottiene per differenza:

$$P_{ed} = 1 - P_{cd} - P_{icd} \tag{6.38}$$

6.3.6 Probabilità d'errore all'uscita del decodificatore

Infine si voglia valutare la probabilità che i simboli decodificati siano errati. Se è prevista la ritrasmissione conviene distinguere tra blocchi consegnati al destinatario con errori non rivelati, con probabilità P_{icd}, e blocchi per i quali sia riconosciuta la presenza di errori non correggibili, con probabilità P_{ed}. Infatti in molti casi P_{icd} risulta molto minore di P_{ed}. Se invece non è possibile la ritrasmissione anche gli errori rivelati (ma non correggibili) devono essere tenuti in conto.

Errori non rivelati

Per tener conto degli errori non rivelati basta moltiplicare ciascun termine della somma che fornisce P_{icd} per i/N (si assume che gli i errori nella parola decodificata cadano con

[8] si noti che se il codice è usato solo come rivelatore ($C = 0$) l'espressione si semplifica notevolmente; si ottiene, in accordo con il risultato già visto per il caso binario,

$$P_{icd} = \sum_{i=d}^{N} A_i \left(\frac{P}{q-1}\right)^i (1-P)^{N-i}$$

[9] anche questa espressione si riduce, per $C = 0$, al risultato già visto

uguale frequenza nelle cifre d'informazione e di parità; ciò è vero per codici ciclici, e molto vicino al vero anche per codici non ciclici).

Ad alto rapporto segnale-rumore il caso più probabile è di avere il minor numero possibile $d - C$ di errori sul canale, ottenere una parola a distanza C da una parola errata, e quindi avere d errori nella parola decodificata. Se tutti gli altri termini sono effettivamente trascurabili si ottiene

$$P_e \approx \frac{d}{N} P_{icd} \tag{6.39}$$

Errori rivelati

Alla probabilità d'errore a valle del decodificatore dovuta agli errori non rivelati occorre aggiungere (se non è possibile la ritrasmissione) gli errori contenuti nei blocchi con errori rivelati, ma non correggibili. Come semplice approssimazione ad alto rapporto segnale-rumore si possono considerare *tutte* le configurazioni di $k > t$ errori, supponendo che queste producano k simboli errati sul totale di N (nella maggior parte dei casi la presenza di errori viene rivelata e quindi il decodificatore non ne aggiunge altri). Si ottiene quindi (nel caso binario)

$$P_e \approx \sum_{k=t+1}^{N} \frac{k}{N} \binom{N}{k} P^k (1-P)^{N-k} \approx \frac{t+1}{N} \binom{N}{t+1} P^{t+1} (1-P)^{N-t-1} \tag{6.40}$$

Si noti che in questa approssimazione non è richiesta la conoscenza dei pesi A_i delle parole del codice.

6.4 Algoritmo di *Hartmann* e *Rudolph*

Un altro esempio notevole di applicazione della (6.7) è l'algoritmo proposto da *Hartmann* e *Rudolph* per la minimizzazione della probabilità d'errore sui bit nella decodifica *soft* di codici a blocco (o anche convoluzionali)[10]. L'algoritmo è basato sul codice duale[11]. Si vedrà dapprima l'algoritmo originale, e poi come sia possibile estenderlo per ottenere una *uscita soft* dal decodificatore, cioè una misura della affidabilità dei bit decisi, utilizzabile come ingresso *soft* di un successivo decodificatore negli schemi di codifica concatenata.

[10] *Hartmann* e *Rudolph* riconoscono nel loro contributo che la minimizzazione della probabilità d'errore sui bit anziché sul blocco non sembra fornire rilevanti miglioramenti delle prestazioni; pongono invece l'accento sul fatto che per codici a blocco la complessità di elaborazione è proporzionale a 2^{N-K} anziché 2^K (se si calcolassero tutte le correlazioni)

[11] anche per un codice convoluzionale, con *rate* b/n, si può definire un duale, con *rate* $(n-b)/n$; l'algoritmo duale è particolarmente conveniente se il *rate* è elevato

Supponendo 0 e 1 equiprobabili a priori, se $\mathbf{r}$ è il vettore ricevuto la decisione ottimale sul bit m-esimo è 0 se il segno della *differenza*

$$D = P(c_m = 0/\mathbf{r}) - P(c_m = 1/\mathbf{r}) \equiv p(\mathbf{r}/c_m = 0) - p(\mathbf{r}/c_m = 1) \equiv \sum_{\mathbf{c}} p(\mathbf{r}/\mathbf{c})(-1)^{c_m} \quad (6.41)$$

è positivo. Definendo la funzione

$$f(\mathbf{u}) = p(\mathbf{r}/\mathbf{u})(-1)^{u_m} \quad (6.42)$$

e osservando che si può scrivere

$$(-1)^{u_m} = \prod_{j=0}^{N-1} (-1)^{u_j \delta_{jm}} \quad (6.43)$$

dove

$$\delta_{jm} = \begin{cases} 1 & j = m \\ 0 & \text{altrimenti} \end{cases} \quad (6.44)$$

si ha, per un canale senza memoria,

$$\begin{aligned} F(\mathbf{v}) &= \sum_{\mathbf{u}} (-1)^{\mathbf{u}\cdot\mathbf{v}} p(\mathbf{r}/\mathbf{u})(-1)^{u_m} = \sum_{u_0=0}^{1} \cdots \sum_{u_{N-1}=0}^{1} \prod_{j=0}^{N-1} (-1)^{u_j v_j} p(r_j/u_j)(-1)^{u_j \delta_{jm}} = \\ &= \prod_{j=0}^{N-1} \sum_{u_j=0}^{1} p(r_j/u_j)(-1)^{u_j (v_j + \delta_{jm})} \equiv \prod_{j=0}^{N-1} P_j^{v_j \oplus \delta_{jm}} \end{aligned} \quad (6.45)$$

Nell'ultimo passaggio si è diviso per $\prod_{j=0}^{N-1} (p(r_j/0) + p(r_j/1))$ e si è posto $P_j = \frac{p(r_j/0) - p(r_j/1)}{p(r_j/0) + p(r_j/1)}$; inoltre si è indicata con $\oplus$ la somma binaria (modulo 2). In tal modo tutti i termini del prodotto per cui risulta $v_j \oplus \delta_{jm} = 0$ valgono 1. Infine si ottiene

$$D \equiv \sum_{\mathbf{c}} p(\mathbf{r}/\mathbf{c})(-1)^{c_m} \equiv \sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j \oplus \delta_{jm}} \quad (6.46)$$

Se interessa solo la miglior decisione su c_m non occorre altro. Basta valutare la (6.46) per ogni m, e decidere in base ai segni.

Se invece si desidera calcolare una misura dell'affidabilità della decisione, è evidente che valutando in modo analogo $\sum_{\mathbf{c}} p(\mathbf{r}/\mathbf{c})$, e ignorando le stesse costanti moltiplicative, si ottiene, per la *somma* delle probabilità,

$$S \equiv \sum_{\mathbf{c}} p(\mathbf{r}/\mathbf{c}) \equiv \sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j} \quad (6.47)$$

Tenendo conto anche di eventuali probabilità a priori $P(c_m = 0)$ e $P(c_m = 1)$ non bilanciate si ha

$$\frac{P(c_m = 0/\mathbf{r})}{P(c_m = 1/\mathbf{r})} = \frac{P(c_m = 0)}{P(c_m = 1)} \frac{p(\mathbf{r}/c_m = 0)}{p(\mathbf{r}/c_m = 1)} = \frac{P(c_m = 0)}{P(c_m = 1)} \frac{p(r_m/c_m = 0)}{p(r_m/c_m = 1)} \frac{p(\overline{\mathbf{r}}/c_m = 0)}{p(\overline{\mathbf{r}}/c_m = 1)} \quad (6.48)$$

dove $\overline{\mathbf{r}}$ differisce da $\mathbf{r}$ perché si pone $r_m = 0$ (e quindi $P_m = 0$)[12]. Si ottiene il prodotto di tre termini, che derivano rispettivamente dalle probabilità a priori, dal solo campione ricevuto relativo al bit m-esimo, e da tutti gli altri campioni (escluso l'm-esimo) tenendo ovviamente conto delle regole del codice. Si ha poi

$$\frac{p(\overline{\mathbf{r}}/c_m = 0)}{p(\overline{\mathbf{r}}/c_m = 1)} = \frac{S + D}{S - D} = \frac{\sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j} + \sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j \oplus \delta_{jm}}}{\sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j} - \sum_{\mathbf{v} \in C'} \prod_{j=0}^{N-1} P_j^{v_j \oplus \delta_{jm}}} \quad (6.49)$$

e quindi, ricordando che $P_m = 0$,

$$\frac{p(\overline{\mathbf{r}}/c_m = 0)}{p(\overline{\mathbf{r}}/c_m = 1)} = \frac{\sum_{\mathbf{v}: v_m = 0} \prod_{j \neq m} P_j^{v_j} + \sum_{\mathbf{v}: v_m = 1} \prod_{j \neq m} P_j^{v_j}}{\sum_{\mathbf{v}: v_m = 0} \prod_{j \neq m} P_j^{v_j} - \sum_{\mathbf{v}: v_m = 1} \prod_{j \neq m} P_j^{v_j}} = \frac{\sum_{\mathbf{v} \in C'} \prod_{j \neq m} P_j^{v_j}}{\sum_{\mathbf{v} \in C'} (-1)^{v_m} \prod_{j \neq m} P_j^{v_j}} \quad (6.50)$$

Si osservi che in ogni prodotto occorre considerare solo i P_j corrispondenti a $v_j = 1$.

Infine è utile considerare i logaritmi. Si definisce *valore algebrico a posteriori* la quantità

$$L_m = \log \frac{P(c_m = 0/\mathbf{r})}{P(c_m = 1/\mathbf{r})} \quad (6.51)$$

che è somma di tre termini: *valore algebrico a priori* (o *componente a priori*)

$$a_m = \log \frac{P(c_m = 0)}{P(c_m = 1)} \quad (6.52)$$

valore algebrico di canale (detto anche *componente sistematica*)

$$\lambda_m = \log \frac{p(r_m/c_m = 0)}{p(r_m/c_m = 1)} \quad (6.53)$$

e *valore algebrico estrinseco* (o *componente estrinseca*)

$$e_m = \log \frac{\sum_{\mathbf{v}: v_m = 0} \prod_{j \neq m} P_j^{v_j} + \sum_{\mathbf{v}: v_m = 1} \prod_{j \neq m} P_j^{v_j}}{\sum_{\mathbf{v}: v_m = 0} \prod_{j \neq m} P_j^{v_j} - \sum_{\mathbf{v}: v_m = 1} \prod_{j \neq m} P_j^{v_j}} = \log \frac{\sum_{\mathbf{v} \in C'} \prod_{j \neq m} P_j^{v_j}}{\sum_{\mathbf{v} \in C'} (-1)^{v_m} \prod_{j \neq m} P_j^{v_j}} \quad (6.54)$$

[12]il lettore può verificare che, con qualche passaggio in più, si ottiene lo stesso risultato finale anche senza estrarre il contributo di r_m a questo punto

Per il canale gaussiano con ingresso binario è facile verificare che il valore algebrico di canale è proporzionale a r_m:

$$\lambda_m = \log \frac{\exp(-(r_m - 1)^2/2\sigma_n^2)}{\exp(-(r_m + 1)^2/2\sigma_n^2)} = \frac{2}{\sigma_n^2} r_m \tag{6.55}$$

Osservando che

$$\log \frac{x + y}{x - y} = 2 \tanh^{-1}(y/x) \tag{6.56}$$

si può esprimere la componente estrinseca anche come

$$e_m = 2 \tanh^{-1} \frac{\sum_{\mathbf{v}:v_m=1} \prod_{j \neq m} P_j^{v_j}}{\sum_{\mathbf{v}:v_m=0} \prod_{j \neq m} P_j^{v_j}} \tag{6.57}$$

Volendo esprimere P_j in termini di valori algebrici di canale è infine facile verificare che

$$P_j = \frac{\exp(\lambda_j) - 1}{\exp(\lambda_j) + 1} = \tanh(\lambda_j/2) \tag{6.58}$$

Come esempio elementare si può considerare un codice a parità semplice, il cui duale contiene solo la parola di tutti zeri e quella di tutti uni. La componente estrinseca è

$$e_m = \log \frac{1 + \prod_{j \neq m} \tanh(\lambda_j/2)}{1 - \prod_{j \neq m} \tanh(\lambda_j/2)} = 2 \tanh^{-1} \prod_{j \neq m} \tanh(\lambda_j/2) \tag{6.59}$$

La componente estrinseca relativa a c_m è interpretabile come informazione derivante da tutti gli altri campioni ricevuti r_j ($j \neq m$) attraverso le regole di parità del codice (si ricordi che gli uni delle parole del codice duale corrispondono a tutte le possibili regole di parità del codice).

Le componenti a priori a_j e quelle di canale λ_j sono equivalenti a tutti gli effetti, e possono essere sommate. Si noti però che per poter sommare correttamente occorre conoscere il rapporto segnale-rumore.

Una alternativa all'algoritmo di *Hartmann* e *Rudolph* per la decodifica bit per bit di un codice a blocco è l'algoritmo di *Bahl*. Infatti si può rappresentare il codice mediante un traliccio, come ad esempio quello di Fig. 6.1, e a questo applicare l'algoritmo di *Bahl*.

Capitolo 7

Turbo codici

I turbo codici, proposti negli anni '90, sul canale gaussiano e fino a probabilità d'errore dell'ordine di $10^{-5} \div 10^{-6}$ consentono prestazioni vicine alla capacità (entro $0.5 \div 1$ dB) con complessità accettabile.

L'idea originale è la *concatenazione parallela* di due codici: il blocco di cifre d'informazione è codificato con due *diversi* codici sistematici; si trasmettono le cifre d'informazione (una sola volta) e i due blocchi di cifre di parità. In pratica il modo più semplice per ottenere il secondo codice è usare lo stesso primo codice, alimentandolo con le cifre d'informazione in ordine permutato: anche le cifre di parità così ottenute sono combinazioni lineari delle cifre d'informazione, ma completamente diverse dalle prime parità.

Fondamentale per ottenere le ottime prestazioni dei turbo codici con ricevitori di complessità non eccessiva è la tecnica di decodifica iterativa: si decodificano alternativamente i due codici, con continuo scambio di informazione *soft* aggiornata. Questa tecnica, benchè non sia la decodifica ottimale del codice concatenato, consente di ottenere risultati molto buoni e soprattutto di gestire codici con dimensione del blocco molto grande, come raccomandato dalla teoria dell'informazione.

Sono stati proposti e studiati anche turbo codici con *concatenazione serie*. Si tratta della forma tradizionale di concatenazione, descritta nel Cap. 2: i bit d'informazione sono codificati da un codice *esterno*; *tutti* i bit così ottenuti, anche quelli di parità del primo codice, sono permutati da un *interleaver* e nuovamente codificati da un codice *interno*. Contrariamente al caso di concatenazione parallela non occorre, ed anzi può non essere conveniente, che i codici componenti siano sistematici. La novità rispetto alle tradizionali tecniche di decodifica dei codici concatenati è la decodifica iterativa, simile a quella dei turbo codici con concatenazione parallela.

In linea di massima con la concatenazione serie si ha un peggioramento della convergenza, con una perdita di qualche decimo di dB, ma si ottengono prestazioni asintotiche migliori.

7.1 Decodifica iterativa

In ricezione si attivano alternativamente i decodificatori dei due codici componenti, in modo da utilizzare ad ogni passo l'informazione prodotta dall'altro decodificatore al passo precedente. In pratica se i codici componenti sono uguali uno stesso modulo è in grado di decodificare entrambi i codici; cambia semplicemente l'ordinamento delle cifre d'informazione e quindi delle informazioni a posteriori prodotte dal decodificatore.

Per illustrare l'algoritmo iterativo di decodifica si farà riferimento ai concetti di *valori algebrici a priori*, *valori algebrici di canale* e *componenti estrinseche* discussi a proposito dell'algoritmo di *Hartmann* e *Rudolph* nella Sez. 6.4.

7.1.1 Concatenazione parallela

Si consideri la concatenazione parallela di due codici. Al primo passo il primo decodificatore[1] riceve in ingresso i valori algebrici a priori dei bit d'informazione[2] e i valori algebrici di canale, relativi sia ai bit d'informazione sia a quelli di parità del primo codice. Il modulo decodificatore produce le componenti estrinseche per tutti i bit d'informazione. Non solo gli ingressi, quindi, ma anche le uscite sono valori reali: il modulo è *Soft-In-Soft-Out* (SISO)[3].

Dopo questa prima decodifica i bit d'informazione non sono più zeri e uni equiprobabili: il modulo SISO ha valutato per ciascun bit d'informazione il soddisfacimento delle regole di parità del primo codice ed ha assegnato un corrispondente valore algebrico.

Il secondo decodificatore riceve in ingresso, ovviamente in ordine permutato, le componenti estrinseche prodotte al passo precedente, che sono utilizzate come valori algebrici a priori dei bit d'informazione. Il decodificatore riceve anche i valori algebrici di canale relativi ai bit d'informazione, già utilizzati nella prima decodifica (ma ora nella versione permutata), e ai bit di parità del secondo codice. Da questi ingressi produce i nuovi valori algebrici a posteriori, ovvero stime aggiornate (sulla base delle regole del secondo codice) delle probabilità dei bit d'informazione. Sottraendo gli ingressi si ottengono le componenti estrinseche relative al secondo codice.

Si procede poi con una seconda iterazione, in cui anche il primo decodificatore riceve le componenti estrinseche prodotte al passo precedente (riordinate secondo la permutazione inversa). Ogni passo SISO è quindi alimentato dalla nuova informazione estrinseca prodotta dall'altro modulo[4].

[1] per la concatenazione parallela ha poca importanza quale dei due codici sia decodificato per primo; se i due codici componenti sono uguali, come spesso accade, la scelta è del tutto arbitraria

[2] quasi sempre nulli, poiché solitamente non vi è alcuna informazione a priori sulle probabilità dei bit d'informazione

[3] se il modulo SISO decodificatore calcolasse i valori algebrici a posteriori anziché le componenti estrinseche, basterebbe poi sottrarre i valori algebrici a priori e di canale

[4] da cui il nome *turbo* dato all'algoritmo di decodifica, e per estensione ai codici pensati per tale tecnica

Ad ogni passo si potrebbero prendere decisioni *hard* sui bit d'informazione, in base al segno dei valori algebrici a posteriori (ovvero della somma delle componenti estrinseche e di canale in ingresso e delle componenti estrinseche in uscita). In pratica si procede per un numero prefissato di iterazioni, eventualmente terminando in anticipo qualora i due decodificatori abbiano già raggiunto un accordo sulle decisioni *hard* o sia comunque soddisfatto un adeguato criterio di arresto.

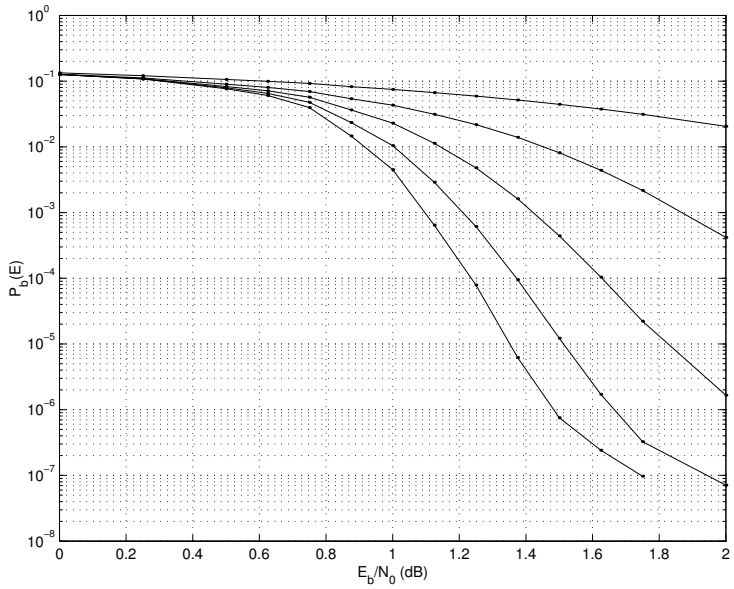

Figura 7.1: Prestazioni di un turbo codice, dalla prima alla quinta iterazione

L'utilità delle iterazioni risulta evidente dalla Fig. 7.1, che mostra le prestazioni dalla prima alla quinta iterazione di un codice a 8 stati con *rate* complessivo $R = 1/2$ e dimensione del blocco di 4224 bit d'informazione[5]. E' chiaro che senza iterazioni le prestazioni sarebbero decisamente deludenti, pur utilizzando una decodifica *soft*. Quante iterazioni siano convenienti dipende sia dai codici componenti sia dal rapporto segnale-rumore, e può essere almeno in parte previsto con una tecnica che sarà descritta nella Sez. 7.3.

7.1.2 Concatenazione serie

Nel caso di concatenazione serie occorre qualche piccola modifica all'algoritmo iterativo. Anzitutto occorre osservare che i campioni ricevuti dal canale sono relativi ai bit del codice interno, che è quindi il primo da decodificare. Alla prima iterazione non si hanno valori algebrici a priori dei bit d'informazione. Si deve iniziare la decodifica calcolando i valori algebrici a posteriori dei bit d'informazione del codice interno.

Poiché i bit d'informazione del codice interno sono anche i bit di codice del codificatore esterno, i valori algebrici appena calcolati (permutati) sono utilizzati dal decodificatore

[5]è uno dei codici dello standard UMTS

del codice esterno come valori algebrici in ingresso. Il decodificatore esterno calcola i valori algebrici a posteriori, *non* dei *bit d'informazione* ma dei *bit di codice*. Infatti tutti questi sono i bit d'informazione del codice interno, e i relativi valori algebrici (permutati) saranno utilizzati come componenti a priori nella successiva iterazione. Se ad esempio il codice esterno è sistematico si calcolano i valori algebrici a posteriori non solo dei bit d'informazione ma anche di quelli di parità; in generale interessano i valori algebrici a posteriori di tutti i bit *codificati*.

Si procede allo stesso modo, con l'unica differenza che a partire dalla seconda iterazione il decodificatore interno riceve in ingresso queste componenti a priori, che quindi devono poi essere sottratte dall'uscita (in modo del tutto analogo alle componenti estrinseche nella concatenazione parallela).

7.2 Codici componenti e permutazione

Come codici componenti la soluzione più classica prevede convoluzionali sistematici recursivi. Si considerino per semplicità codici convoluzionali *tail-biting* con *rate* $R = 1/2$. Parole di codice con un solo uno d'informazione contengono un grande numero di uni nelle parità di ciascuno dei due codici e quindi hanno peso certamente elevato[6]. Nel caso di due uni d'informazione è possibile che il primo produca uno stato diverso da quello nullo e che il secondo faccia tornare allo stato nullo. In tal caso il numero di uni nelle cifre di parità può essere limitato, e quindi il peso complessivo della parola può essere piccolo. Per aumentare il numero di uni nelle cifre di parità si preferisce usare polinomi a denominatore primitivi, in modo da rendere massimo il periodo della sequenza pseudocasuale eccitata da un uno d'informazione. Se ad esempio il codice componente ha 16 stati, e se il denominatore è primitivo, la sequenza nata da un uno d'informazione può terminare solo se un successivo uno d'informazione entra a distanza di un multiplo di 15 bit. Per ogni ciclo di sequenza pseudocasuale vengono generati otto uni di parità. Ovviamente perché si possa avere una piccola distanza minima nel codice concatenato occorre che lo stesso fenomeno si ripeta nella versione permutata.

L'analisi è più complessa quando si considerino tre o più uni nel blocco d'informazione. Si cerca di combattere la possibilità di parole di peso basso progettando l'*interleaver* che realizza la permutazione in modo che uni vicini nel blocco d'informazione siano il più possibile distanti nella sequenza permutata, e viceversa. L'esperienza mostra che è possibile tenere gli uni a distanza circa pari ad almeno $S = \sqrt{K/2}$, dove K è il numero di bit d'informazione. Il parametro S è detto *spread* della permutazione.

La permutazione ottenuta può essere più o meno soddisfacente, e quindi è consigliabile generarne molte e scegliere la migliore[7].

[6] nel caso dei codici terminati si potrebbero invece avere parole di peso piccolo se il singolo uno d'informazione fosse nelle ultime posizioni del blocco (anche nella versione permutata)

[7] in questo modo la permutazione, e la sua inversa, sono *tabelle* e non regole matematiche; se si riesce a trovare una buona regola matematica si risparmia memoria

Una buona permutazione aumenta la distanza minima del codice, e quindi migliora le prestazioni asintotiche. I turbo codici mostrano spesso una differenza notevole di prestazioni nelle regioni rispettivamente di alte e basse probabilità d'errore. Nella prima regione, detta *waterfall* o talvolta *cliff*, le curve di probabilità d'errore in funzione del rapporto segnale-rumore sono estremamente ripide e l'effetto della permutazione è di norma trascurabile. Oltre si ha un cambio di pendenza, spesso così evidente che si parla addirittura di un *floor*. In questa regione le prestazioni sono dominate dai concorrenti a minore distanza, che a loro volta dipendono molto dalla permutazione. Naturalmente per scegliere la permutazione più adatta occorrono strumenti per valutare almeno i primi termini dello spettro delle distanze del turbo codice, se si vogliono evitare lunghe simulazioni con ciascuno degli *interleaver* candidati.

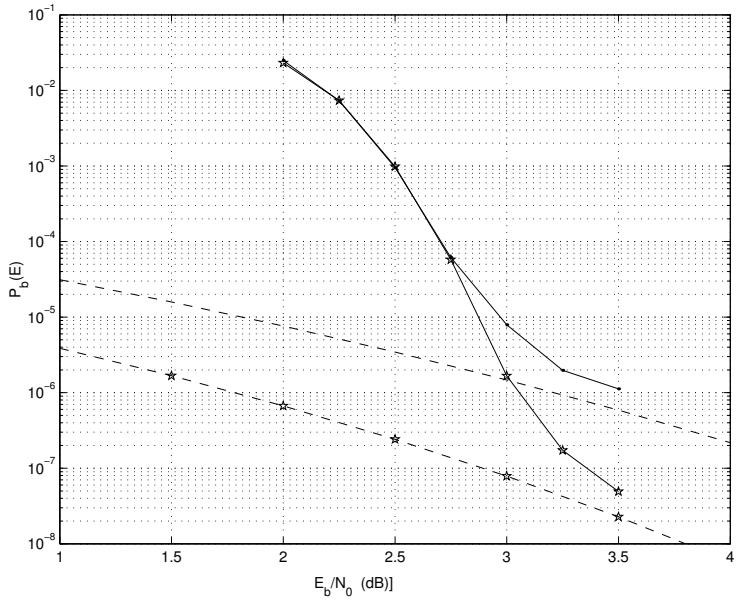

Figura 7.2: Prestazioni di uno stesso turbo codice con due diversi *interleaver*

La Fig. 7.2 mostra due esempi di *floor* per lo stesso turbo codice con blocco di 5006 bit d'informazione e *rate* complessivo $R = 3/4$, ottenuto da codici componenti a 8 stati con *rate* 6/7 con due diversi *interleaver*. Il primo fornisce una permutazione casuale; il secondo è migliore perché garantisce un adeguato *spread*. Sono anche mostrate approssimazioni asintotiche della probabilità di errore basate sulla valutazione dei primi termini dello spettro delle distanze del turbo codice (si veda la Sez. 7.4).

Nel caso di concatenazione serie il codice interno è solitamente un convoluzionale sistematico recursivo con piccola memoria, spesso di *rate* alto o addirittura unitario[8], mentre può essere conveniente un codice esterno convoluzionale non recursivo.

[8]il più semplice convoluzionale recursivo con *rate* $R = 1$ ha funzione di trasferimento $\frac{1}{1+D}$ cioè è un semplice accumulatore

7.2.1 Perforazione

Poiché le cifre d'informazione sono trasmesse una sola volta, la concatenazione parallela di due codici convoluzionali con *rate* $R = 1/2$ dà un codice con *rate* complessivo $R = 1/3$. Volendo ottenere *rate* più alti si hanno due possibilità: ricorrere alla perforazione oppure usare codici componenti con *rate* maggiore. Più tradizionale è la prima soluzione: il primo turbo codice, proposto nel 1993, perforava alternativamente un bit della prima e della seconda sequenza di parità, cancellando quindi metà dei bit di parità, per ottenere un *rate* complessivo $R = 1/2$. Perforazioni più consistenti[9] permettono di ottenere *rate* molto più elevati, se richiesti[10]. Tuttavia l'esperienza ha mostrato che non è facile tenere sotto controllo la distanza minima del codice, e quindi ottenere buone prestazioni asintotiche, quando si perfora pesantemente (in particolare se la perforazione è sostanzialmente casuale, come spesso avviene per semplicità di realizzazione).

Sembra preferibile, per ottenere *rate* elevati e buone prestazioni asintotiche, ricorrere a codici componenti con *rate* tale da non richiedere perforazione (o una modesta perforazione).

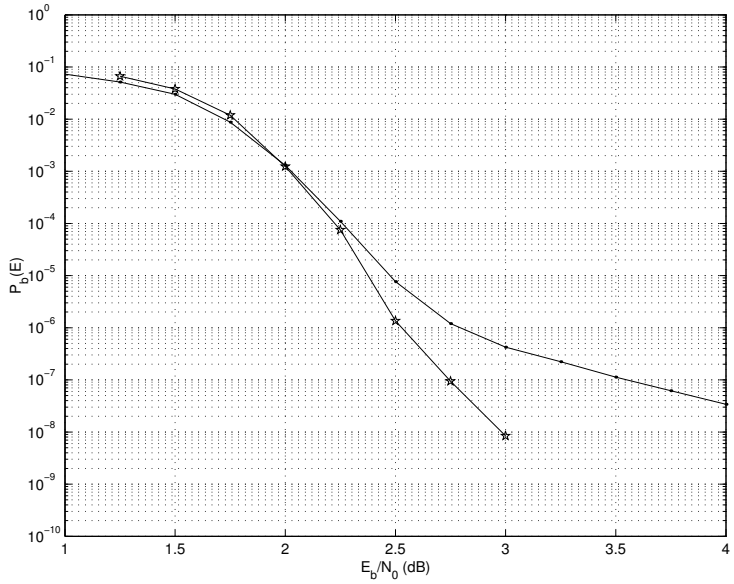

Figura 7.3: Prestazioni di due turbo codici con *rate* complessivo 2/3, ottenuti con e senza perforazione

La Fig. 7.3 mostra ad esempio le prestazioni dopo 10 iterazioni di due turbo codici con blocco di 1504 bit d'informazione e *rate* complessivo $R = 2/3$. I codici componenti hanno 8

[9] sembra accertato che si ottengono prestazioni migliori perforando solo bit di parità, e mai bit d'informazione

[10] non si dimentichi tuttavia che *rate* complessivi molto elevati sono penalizzati dalla trasmissione binaria; si riveda in proposito la Fig. 1.2, che mostra che per *rate* superiori a $0.75 \div 0.8$ si dovrebbero usare modulazioni a quattro livelli anziché binarie

stati e *rate* rispettivamente 1/2 e 4/5. Nel primo caso si ha una consistente perforazione[11] e una notevole degradazione delle prestazioni asintotiche.

7.2.2 Concatenazione di più codici

Ci si potrebbe porre la domanda se sia conveniente concatenare tre o più codici, mantenendo la semplicità della decodifica turbo iterativa. La risposta sembra essere che le prestazioni peggiorano in modo sensibile, come mostra l'esempio di Fig. 7.4. I codici hanno *rate* complessivo 1/2 e sono ottenuti dalla concatenazione di due oppure tre componenti a 16 stati, con *rate* 2/3 e 3/4 rispettivamente. La dimensione del blocco è 5994 bit d'informazione. Sono anche mostrate le prestazioni attese nel caso di blocco di dimensione molto elevata, valutate con la tecnica che sarà descritta nella Sez. 7.3.

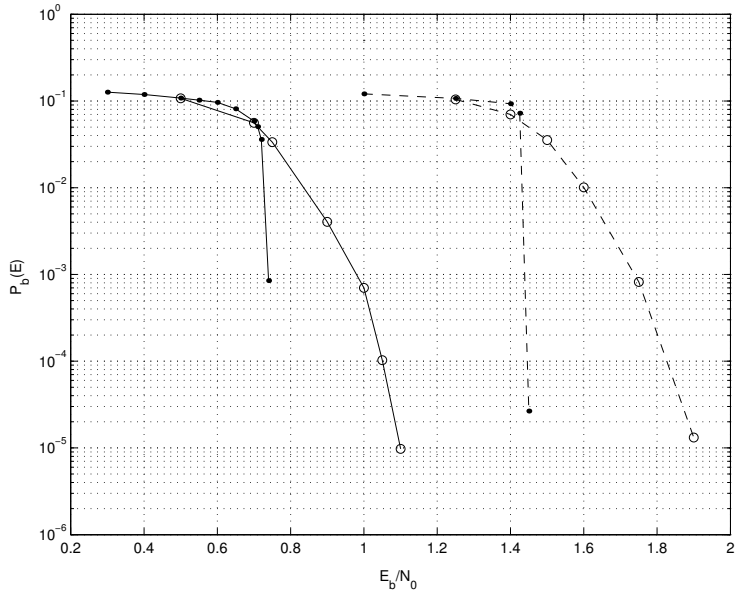

Figura 7.4: Prestazioni di due turbo codici con *rate* complessivo 1/2 e blocco di 5994 bit d'informazione, ottenuti concatenando due codici con *rate* 2/3 (curve continue) oppure tre codici con *rate* 3/4 (curve tratteggiate)

Non si deve concludere che codici ottenuti concatenando tre codici componenti siano intrinsecamente peggiori di quelli con due soli componenti. Vi sono anzi forti indizi che siano migliori. La conclusione sembra piuttosto essere che nel caso di tre o più codici risulti inefficiente la tecnica di decodifica iterativa.

A questo proposito conviene osservare che un generico codice a blocco può essere visto come concatenazione parallela di $N - K$ codici a parità singola (su sottoinsiemi diversi di cifre d'informazione). Tuttavia ogni tentativo di decodifica turbo, un bit di parità per volta,

[11] anche questo è uno dei codici dello standard UMTS

non ha finora dato risultati apprezzabili. Probabilmente ciò è anche dovuto al fatto che le equazioni di parità di un codice di *Hamming* o di un *BCH* coinvolgono un numero elevato di bit. La decodifica iterativa sembra essere efficiente solo se sono disponibili equazioni di parità con poche variabili, come avviene per i codici descritti nella prossima sezione.

Codici *Low Density Parity Check*

L'unica notevole eccezione è un tipo particolare di codici a blocco, non sistematici, proposti negli anni '60. Si tratta dei codici *Low Density Parity Check* (LDPC), definiti da matrici di parità con il minor numero possibile di uni. Un semplicissimo esempio, di dimensione troppo piccola per essere interessante in pratica, è il seguente:

$$H = \left[\begin{array}{cccccccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \tag{7.1}$$

Vi sono cinque uni in ogni riga e tre in ogni colonna[12]. Ogni equazione di parità coinvolge cinque bit di codice e ogni bit interviene solo in tre regole di parità. Si può verificare che solo 10 delle 12 righe sono linearmente indipendenti (anche se per maggior simmetria tutte le righe vengono utilizzate nella decodifica), per cui la dimensione del codice è $N = 20$, $K = 10$.

La particolarità dei codici LDPC risulta molto più evidente se si immagina, ad esempio, una matrice di parità con 300 righe e 500 colonne ma ancora con soli cinque uni per riga e tre per colonna[13].

I codici LDPC sono stati trascurati per decenni, probabilmente sia perché non sistematici (ciò rende l'associazione tra parole di codice e bit d'informazione più complessa) sia perché con i valori della dimensione del blocco relativamente piccoli che era possibile simulare a quei tempi non risultavano evidenti le potenzialità per grandi valori di N. I codici LDPC sono stati riscoperti solo dopo i turbo i codici, con cui ora rivaleggiano in prestazioni. La tecnica di decodifica è molto simile, e per certi aspetti addirittura meno complessa perché i

[12] tre uni per colonna sembra essere la scelta migliore: due uni non danno buoni codici; quattro o più uni danno codici probabilmente migliori, ma meno adatti alla decodifica iterativa

[13] se il codice fosse costruito affidandosi al caso, come descritto nel Cap. 2, ogni regola di parità coinvolgerebbe circa 250 bit e ogni bit d'informazione parteciperebbe a circa 150 equazioni di parità

codici componenti (corrispondenti alle righe di H) sono semplicissimi. La convergenza dei codici LDPC può essere ulteriormente migliorata rendendo irregolare, in modo controllato, il numero di uni nelle equazioni di parità.

Turbo codici a blocco

Presentano un certo interesse anche i turbo codici a blocco (*Block Turbo Codes*: BTC). Si tratta dei codici prodotto già descritti nel Cap. 2, decodificati in modo *soft* e iterativo. Se si eliminano le parità delle parità si ottiene una concatenazione parallela. Se invece si mantengono le parità delle parità la concatenazione è serie (con codici componenti sistematici): come nel caso dei codici componenti convoluzionali si ottengono prestazioni peggiori per quanto riguarda la convergenza, ma migliori asintoticamente. Nel caso di concatenazione serie si possono scambiare i ruoli dei due codici (interno ed esterno, o viceversa) e quindi la decodifica può iniziare da uno qualsiasi dei due. Inoltre si possono calcolare i valori algebrici a posteriori per tutti i bit, di informazione e di parità, in ciascuno dei due decodificatori. Questa variante sembra dare un piccolo guadagno nella convergenza delle iterazioni.

La differenza rispetto ai turbo codici con componenti convoluzionali è che non si hanno solo due codici, ma tanti quante sono le righe e le colonne. Le cifre di parità di una riga o di una colonna sono combinazioni lineari dei soli bit d'informazione contenuti nella riga o colonna stessa, anziché di *tutti* i bit d'informazione. Ciò ha due effetti: da un lato non si ottengono le stesse prestazioni dei normali turbo codici; d'altra parte ad ogni iterazione la decodifica di tutte le righe (o colonne) può essere effettuata da altrettanti decodificatori SISO in parallelo, non essendovi alcuna interazione tra di essi. Ciò consente una notevole parallelizzazione dei circuiti di decodifica e quindi velocità elevate. Un altro punto a favore di questi codici è che è relativamente facile determinare la distanza minima del turbo codice, e la relativa molteplicità, e quindi ottenere una buona stima delle prestazioni asintotiche.

Un altro vantaggio è che si ottengono facilmente *rate* elevati senza dover fare uso della perforazione, usando codici componenti con piccola ridondanza e quindi anche facilmente decodificabili con l'algoritmo di *Hartmann* e *Rudolph* descritto nel Cap. 6. Risulta invece difficile ottenere codici con basso *rate* e facilmente decodificabili.

Un punto a sfavore è la scarsa flessibilità nella scelta di *rate* e dimensione del blocco. Si può tuttavia aumentare la dimensione del blocco utilizzando il prodotto di tre o più codici.

Alcuni esempi di prestazioni di codici BTC sono mostrati nelle Fig. 7.5, 7.6 e 7.7. I primi due sono ottenuti dalla concatenazione serie rispettivamente di due *Hamming* estesi (64,57) e di tre *Hamming* estesi (32,26). La dimensione del blocco è quindi di circa 3250 e 17500 bit d'informazione, ed il *rate* complessivo è circa 0.79 e 0.57. Il terzo codice (senza parità delle parità, cioè con concatenazione parallela) è ottenuto dal prodotto di tre *Hamming* (63,57). Il blocco è di circa 185000 bit d'informazione e il *rate* è 0.76.

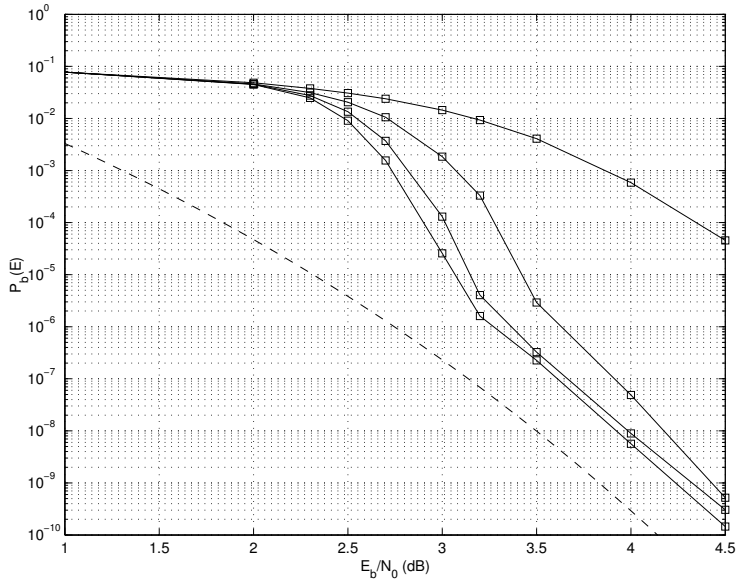

Figura 7.5: Prestazioni di turbo codice a blocco, con due *Hamming* estesi (64,57) come codici componenti, dalla prima alla quarta iterazione

7.3 Convergenza della decodifica iterativa

Fin dai primi turbo codici proposti è risultata evidente la buona convergenza delle tecniche di decodifica iterativa, ma mancava una convincente spiegazione dei motivi per cui ciò

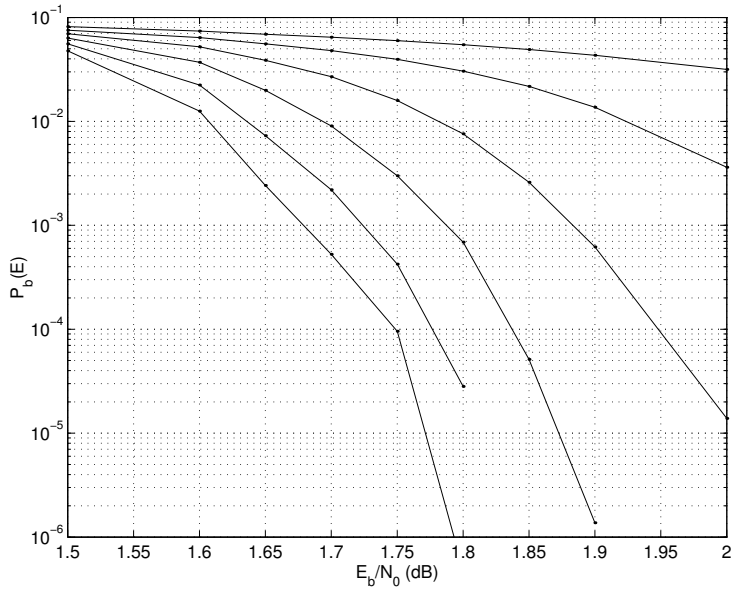

Figura 7.6: Prestazioni di turbo codice a blocco, con tre *Hamming* estesi (32,26) come codici componenti, dalla prima alla sesta iterazione

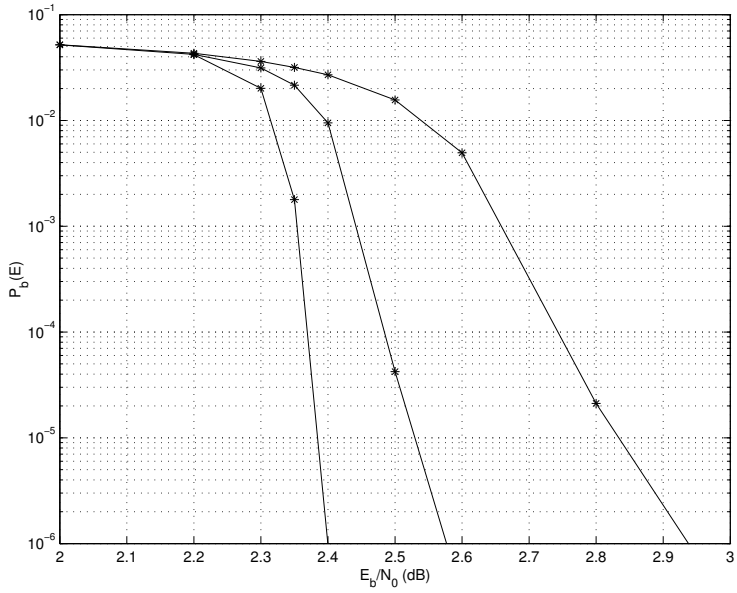

Figura 7.7: Prestazioni di turbo codice a blocco, con tre *Hamming* (63,57) come codici componenti, alla terza, quinta e ottava iterazione

accade. Ora sono invece disponibili strumenti abbastanza semplici in grado di predire con buona accuratezza la velocità di convergenza delle iterazioni, o la eventuale non convergenza a basso rapporto segnale-rumore.

La più accurata fra le varie tecniche proposte finora studia l'evoluzione con le iterazioni della informazione mutua tra i bit di informazione e le componenti estrinseche all'uscita dei decodificatori. La teoria assume che la dimensione del blocco sia molto grande, e fornisce risultati particolarmente precisi in questo caso ma comunque utili anche per blocchi di poche migliaia di bit.

Si parte da alcune osservazioni sperimentali, ottenute simulando la decodifica di un gran numero di blocchi codificati:

- se la dimensione del blocco è sufficientemente elevata l'informazione estrinseca prodotta al passo precedente, e applicata all'ingresso del decodificatore come informazione a priori, rimane per molte iterazioni praticamente incorrelata con i valori algebrici di canale; una spiegazione intuitiva di questo fatto è che l'informazione estrinseca relativa al bit m-esimo prodotta in un generico passo di decodifica non dipende dal valore algebrico di canale, né da quello a priori, dello stesso bit m-esimo; è solo attraverso i successivi scambi di informazione tra i decodificatori che il valore algebrico di canale di un bit ritorna (molto attenuato) all'ingresso m-esimo come componente a priori

- le componenti estrinseche hanno densità di probabilità gaussiana (ciò è vero soprattutto all'aumentare delle iterazioni); un tentativo di spiegazione intuitiva è basato sul

fatto che nel calcolo delle componenti estrinseche si deve sommare un gran numero di contributi (anche se nel modulo SISO non mancano altre operazioni diverse dalla somma)

Accettate queste ipotesi, si deve seguire l'evoluzione di un opportuno parametro di qualità all'uscita dei decodificatori, interpretando le componenti estrinseche via via prodotte come segnali sempre più correlati con i bit d'informazione effettivamente trasmessi e quindi sempre più utili al successivo passo di decodifica. Come parametro di qualità si potrebbe considerare il rapporto segnale-rumore. Tuttavia risultano più precise le stime basate sull'informazione mutua tra bit d'informazione e componenti estrinseche.

Se z è una generica variabile casuale ottenuta sommando una variabile binaria $x = \pm 1$ e un disturbo gaussiano n con varianza σ_n^2 e si considera il corrispondente valore algebrico

$$\lambda = \log \frac{p(z/x=1)}{p(z/x=-1)} = \log \frac{\exp(-(z-1)^2/2\sigma_n^2)}{\exp(-(z+1)^2/2\sigma_n^2)} = \frac{2z}{\sigma_n^2} = \frac{2}{\sigma_n^2}(x+n) \tag{7.2}$$

è facile verificare che tra valor medio e varianza del valore algebrico, condizionati a $x = 1$ (oppure $x = -1$), vale la semplice relazione

$$\sigma_\lambda^2 = 2E[\lambda] \tag{7.3}$$

Uno solo dei due parametri è sufficiente per individuare la densità di probabilità condizionata del valore algebrico.

Tali risultati valgono, con le ipotesi viste sopra, sia per i valori algebrici di canale λ sia per le componenti estrinseche e, e quindi anche per i valori algebrici a priori a applicati all'ingresso dei decodificatori, che sono ottenuti dalla permutazione delle componenti estrinseche calcolate al passo precedente.

E' immediato riconoscere che l'informazione mutua $I(X, A)$ tra un dato binario x e il corrispondente valore algebrico a priori a è uguale alla capacità di un canale con ingresso binario e rumore gaussiano, e quindi si calcola come descritto nel Cap. 1 (Fig. 1.2; $M = 2$). Basta infatti dare al rapporto segnale-rumore σ_x^2/σ_n^2 il valore $E^2[a]/\sigma_a^2 = E[a]/2$.

Per valutare l'informazione mutua $I(X, E)$ tra un dato binario x e la corrispondente componente estrinseca e si simula un numero adeguato di volte il *singolo* decodificatore. Si dovrebbero simulare parole di codice con zeri e uni equiprobabili, ma è facile verificare che si ottiene lo stesso risultato trasmettendo ampiezze x tutte positive (corrispondenti alla parola di tutti zeri). Si applicano agli ingressi campioni ricevuti dal canale, con un valore prefissato di E_b/N_0, e valori algebrici a priori con valor medio $E[a]$ e varianza $\sigma_a^2 = 2E[a]$.

Si calcola poi l'informazione mutua $I(X, E) = H(E) - H(E/X)$. Basta utilizzare la densità di probabilità $\hat{p}(e/x = 1)$ stimata dall'istogramma dei risultati della simulazione, e la corrispondente densità di probabilità non condizionata

$$\hat{p}(e) = \frac{1}{2}\,\hat{p}(e/x=1) + \frac{1}{2}\,\hat{p}(-e/x=1) \tag{7.4}$$

che si sarebbe ottenuta con dati equiprobabili. Infine in modo del tutto analogo alla (1.60) si ha

$$I(X,E) = \int_e \hat{p}(e/x=1) \log \frac{\hat{p}(e/x=1)}{\hat{p}(e)}\, de \tag{7.5}$$

Poiché la variabile x è discreta, ed anzi nella simulazione assume solo il valore $x = 1$, si può anche evitare il calcolo dell'istogramma delle componenti estrinseche utilizzando l'espressione equivalente $I(X,E) = H(X) - H(X/E)$, ovvero

$$I(X,E) = 1 + E\left[\log P(x=1/e)\right] = 1 + \frac{1}{N_b}\sum_{n=1}^{N_b} \log \frac{\exp(e_n)}{\exp(e_n)+1} \tag{7.6}$$

Il valor medio $E[\log P(x=1/e)]$ è stimato mediante la media aritmetica degli N_b valori e_n ottenuti dalla simulazione e la probabilità $P(x=1/e_n)$ è espressa in funzione del valore algebrico e_n. Se il decodificatore calcola dapprima la probabilità $P(x=1/e_n)$ e la converte poi nella componente estrinseca e_n si può risparmiare questa doppia conversione. Anche nella (7.6) si è assunto che sia sempre $x = 1$. Se per qualche ragione si preferisse simulare parole di codice corrispondenti a zeri e uni d'informazione equiprobabili, nella (7.6) si utilizzerebbero le probabilità $P(x/e_n)$ relative ai bit d'informazione effettivamente trasmessi, cioè anche

$$P(x=-1/e_n) = \frac{1}{\exp(e_n)+1} \tag{7.7}$$

Ripetendo l'operazione con diversi valori di $E[a]$, o equivalentemente di $I(X,A)$, si può tracciare il grafico dell'informazione mutua $I(X,E)$ relativa all'uscita estrinseca del decodificatore in funzione dell'informazione mutua $I(X,A)$ relativa alla componente a priori in ingresso. Si ottiene un diverso grafico per ciascun rapporto segnale-rumore E_b/N_0. Esempi di caratteristiche EXIT (*Extrinsic Information Transfer*) di trasferimento dell'informazione estrinseca sono mostrati in Fig. 7.8, per un codice con *rate* 2/3 a 16 stati, a valori di E_b/N_0 pari a 0.5, 0.75 e 1 dB.

Poiché le uscite e di un decodificatore, permutate, sono applicate al passo successivo all'ingresso a dell'altro, si segue facilmente l'evoluzione dell'informazione mutua se si tracciano sullo stesso grafico le caratteristiche di trasferimento dei due codici *con gli assi scambiati*. Nel caso di codici uguali si tratta di una riflessione speculare rispetto alla bisettrice del grafico mostrata in Fig. 7.8. Si parte da $I(X,A) = 0$ e si valuta $I(X,E)$, cioè $I(X,A)$ per il secondo decodificatore. Si procede allo stesso modo fino a ottenere informazione mutua uguale a 1 (e quindi probabilità di errore nulla, persino senza più utilizzare i valori algebrici di canale) oppure fino a quando le due curve si incrociano (nel qual caso l'effetto delle iterazioni si arresta prima di aver annullato la probabilità di errore). In definitiva si riesce a prevedere a quale rapporto segnale-rumore E_b/N_0 le due curve non si intersecano, e quindi le iterazioni turbo convergono a probabilità d'errore teoricamente nulla (in pratica molto bassa).

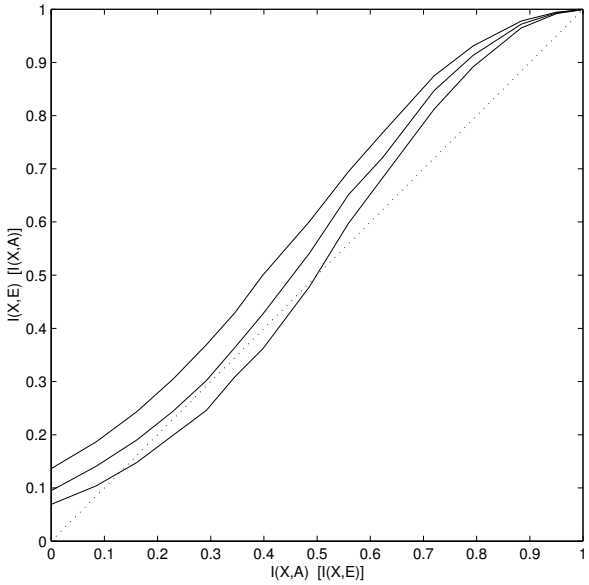

Figura 7.8: Caratteristiche EXIT di un codice a 16 stati con *rate* 2/3 a $E_b/N_0 = 0.5$, 0.75 e 1 dB

La Fig. 7.9 mostra l'evoluzione dell'informazione mutua ad $E_b/N_0 = 0.75$ dB. Si noti che invece dell'andamento teorico è tracciato l'andamento (medio) effettivo, stimato simulando la decodifica turbo di molti blocchi. La dimensione del blocco d'informazione è 32768 bit. Si osservi che a causa della dimensione finita del blocco l'informazione mutua resta lontana

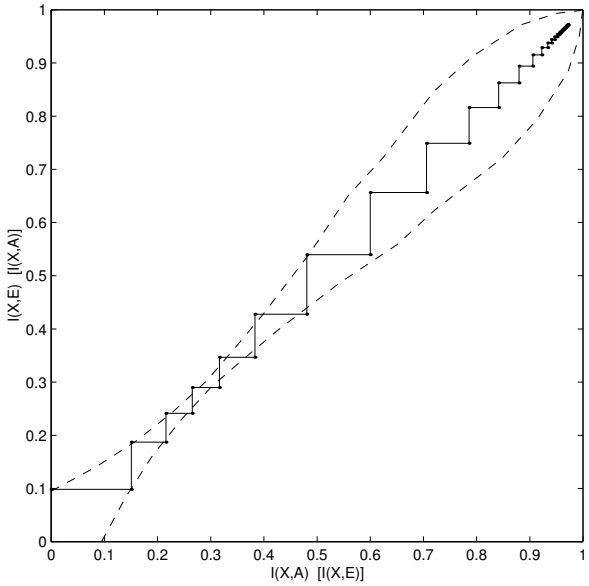

Figura 7.9: Evoluzione teorica ed effettiva (media) dell'informazione mutua a $E_b/N_0 = 0.75$ dB, con blocchi di 32768 bit d'informazione

da uno, e quindi la probabilità d'errore non tende a zero.

La Fig. 7.10 mostra l'andamento della probabilità di errore con le iterazioni. Si notano sia un rallentamento della convergenza intorno alla terza e quarta iterazione sia la convergenza più rapida alle iterazioni successive, previsti dalla teoria. Le caratteristiche EXIT non possono invece predire il comportamento dopo sette o otto iterazioni, dovuto alla crescente correlazione tra ingressi e uscite, non tenuta in conto dalla teoria. Con blocchi di dimensione più grande si avrebbe un buon accordo per un maggior numero di iterazioni.

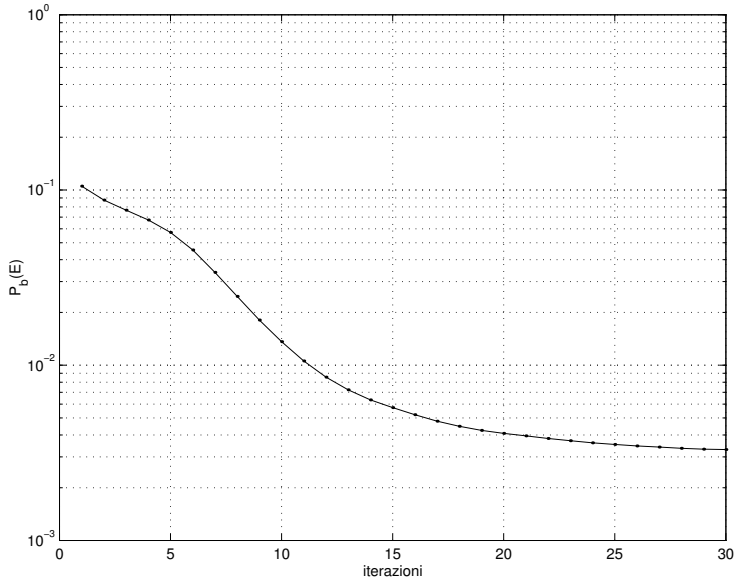

Figura 7.10: Probabilità d'errore in funzione delle iterazioni a $E_b/N_0 = 0.75$ dB, con blocchi di 32768 bit d'informazione

Dalla Fig. 7.8 si vede che a $E_b/N_0 = 0.5$ dB le iterazioni permetterebbero di raggiungere un'informazione mutua di soli 0.13 bit, valore per cui $I(X, E)$ risulta uguale a $I(X, A)$.

Se si tracciano le caratteristiche EXIT di codici con diverso numero di stati si osserva che in generale quelli con *minor* numero di stati convergono a più basso rapporto segnale-rumore (un risultato trovato sperimentalmente fin dai primi turbo codici[14]).

Nel caso di concatenazione serie si hanno alcune piccole differenze. Per il codice interno si deve valutare la caratteristica di trasferimento dell'informazione mutua tra componente a priori e componente estrinseca (che in questo caso è data dalla differenza tra valori algebrici a posteriori e componenti a priori). Per il codice esterno si calcola la caratteristica di trasferimento tra valori algebrici in ingresso e valori algebrici a posteriori (dei bit di codice). Poiché mancano gli ingressi provenienti dal canale si ha *una sola* caratteristica di trasferimento del codice esterno, indipendente dal rapporto segnale-rumore E_b/N_0.

[14]in compenso le prestazioni asintotiche dei codici con minor numero di stati sono generalmente peggiori

7.4 Prestazioni asintotiche dei turbo codici

Si possono ottenere stime abbastanza attendibili delle prestazioni di un turbo codice ad alto rapporto segnale-rumore se sono noti almeno i primi termini dello spettro delle distanze. Occorre tuttavia supporre che, almeno in questa regione, la decodifica iterativa abbia prestazioni non troppo diverse da quelle di un ipotetico ricevitore a massima verosimiglianza. Pur non essendo stato possibile finora ottenere argomenti decisivi a supporto di questa congettura, i confronti tra le simulazioni e le previsioni basate sullo spettro delle distanze incoraggiano ad utilizzare questo strumento, almeno per una prima valutazione.

Non si deve dimenticare che la simulazione di eventi rari come gli errori nella regione di *floor* è estremamente costosa in termini di tempo di calcolo e deve essere riservata ad un insieme limitato di soluzioni da confrontare. Se ad esempio si sono generati centinaia o migliaia di *interleaver* e si vuole scegliere il migliore, uno strumento che consenta di selezionare pochi casi da simulare è benvenuto.

Le probabilità che il blocco contenga errori e che i bit d'informazione siano errati, dopo la decodifica a massima verosimiglianza, sono approssimabili con

$$P(E) = \sum_d n_d Q\left(\sqrt{\frac{2E_b}{N_0}\frac{K}{N}d}\right) \tag{7.8}$$

e

$$P_b(E) = \frac{1}{K}\sum_d w_d Q\left(\sqrt{\frac{2E_b}{N_0}\frac{K}{N}d}\right) \tag{7.9}$$

dove d indica la generica distanza, K è il numero di bit d'informazione e N la dimensione del blocco, n_d è il numero dei concorrenti a distanza d e w_d il corrispondente numero complessivo dei bit d'informazione errati. Ad alto rapporto segnale-rumore si possono troncare le somme a pochi termini.

Negli ultimi anni sono stati proposti alcuni metodi che consentono in molti casi pratici di valutare i primi termini dello spettro delle distanze di un turbo codice. La complessità cresce fortemente con la distanza minima del codice. Attualmente si riescono a trattare solo blocchi di dimensione dell'ordine di poche migliaia.

7.5 Altre applicazioni della elaborazione iterativa

La tecnica iterativa per la decodifica dei turbo codici scompone un codice concatenato, troppo complesso per essere decodificato in modo ottimo, nei suoi due componenti e valuta le probabilità a posteriori considerando alternativamente solo i vincoli dell'uno e dell'altro codice. Il singolo modulo SISO ha una complessità limitata, ed è quindi possibile una valutazione esatta delle probabilità a posteriori, ad esempio con l'algoritmo di *Bahl*. I due

moduli si scambiano ad ogni passo la *nuova* informazione *soft* che ciascuno ha potuto ottenere. Se il procedimento converge si determinano in questo modo buone approssimazioni delle vere probabilità a posteriori del codice complessivo.

In questo processo è fondamentale l'*interleaver*, che rende quasi indipendenti le singole decodifiche anche in molte iterazioni successive. Si noti che l'*interleaver* determina la dimensione N del blocco codificato, solitamente piuttosto grande come raccomanda la teoria dell'informazione (compatibilmente con il ritardo tollerabile).

Possono esistere molte altre situazioni simili, che non derivano dalla concatenazione di due codici. Si consideri ad esempio un codificatore seguito da un modulatore con memoria. E' troppo complesso determinare le probabilità a posteriori dei bit d'informazione tenendo conto contemporaneamente dei vincoli del codice e del modulatore. Invece la determinazione delle probabilità a posteriori considerando le sole regole del modulatore è possibile con algoritmi noti, di complessità accettabile. La decodifica turbo (nella forma adatta alla concatenazione serie) può essere applicata con successo a questa situazione. L'*interleaver* rende molto grande la dimensione del blocco, senza incremento eccessivo della complessità del ricevitore.

La concatenazione di un codice e di un canale con memoria, separati da un opportuno *interleaver* può essere trattata in modo analogo. In tal caso il tradizionale *equalizzatore* è sostituito da un modulo SISO che calcola le probabilità a posteriori dei bit trasmessi sul canale.

Esistono molte altre situazioni simili, a cui si può applicare con successo l'elaborazione turbo.

Capitolo 8

Codici per costellazioni multilivello

Nel progetto dei codici per costellazioni multilivello è necessario tenere conto delle diverse distanze geometriche prodotte dai diversi bit della costellazione.

Si consideri ad esempio una costellazione 8PAM in cui i livelli trasmessi sono proporzionali a -7, -5, -3, -1, 1, 3, 5 e 7 e si supponga che i punti della costellazione siano associati a terne di bit secondo il *mapping* binario naturale: 000, 001, 010, 011, 100, 101, 110, 111. Il bit meno significativo (che diremo bit in posizione 1) dà una distanza geometrica che può evidentemente essere molto minore di quella del bit più significativo (terzo bit). Infatti cambiando valore al terzo bit (e lasciando invariati gli altri) si ha una distanza tra i livelli (al quadrato) pari a 64, mentre cambiando solo il primo bit la distanza al quadrato può essere anche solo 4. Non vi è dubbio quindi che il codice debba essere progettato tenendo conto di tali proprietà di distanza.

E' anche immediato verificare che le distanze cambiano con il *mapping*. E' evidente ad esempio che se si usa il *mapping* di *Gray* 000, 001, 011, 010, 110, 111, 101, 100 il progetto del codice cambia completamente.

Per un lungo periodo il problema non ha ricevuto grande attenzione, e i tentativi di utilizzare i codici binari con costellazioni multilivello non hanno dato risultati soddisfacenti. Poi sono state proposte due possibili soluzioni:

- codici binari diversi per i diversi bit della costellazione; possono risultare comodi sia il *mapping* binario naturale sia quello di *Gray*, e non mancano esempi con altri *mapping*; solitamente si considera un blocco di N simboli (BCM: *Block Coded Modulation*); i diversi bit della costellazione usano codici della stessa lunghezza N ma con diversa ridondanza: i bit meno protetti dal rumore richiedono codici con *rate* più bassi

- un unico codice che opera congiuntamente su bit diversi della costellazione, appositamente progettato per rendere massima la minima distanza geometrica tra segnali concorrenti; solitamente si sono considerati codici convoluzionali (TCM: *Trellis Coded Modulation*); poiché anche nel caso binario la ricerca dei buoni codici convoluzionali è sostanzialmente esaustiva, anche per le costellazioni multilivello si sono provati mol-

tissimi codici e selezionati i migliori (che ovviamente sono diversi dai migliori codici per la trasmissione binaria); il *mapping* naturale rende particolarmente comoda la ricerca dei codici, e quindi viene fissato a priori

Con entrambe le soluzioni è frequente che i bit maggiormente protetti (ad esempio il terzo bit della costellazione 8PAM) non siano codificati, cioè che per i livelli più alti basti un codice con *rate* $R = 1$.

La teoria dell'informazione ovviamente dà il solito suggerimento di aumentare quanto possibile, compatibilmente con complessità e ritardo, la dimensione del blocco di simboli codificato. Quali suggerimenti offre invece sulla scelta dei *rate* dei codici?

8.1 Capacità delle costellazioni multilivello

Considerando ancora come esempio la costellazione 8PAM, l'informazione mutua tra la terna di bit x_1, x_2, x_3 in ingresso[1] e l'uscita y è data da

$$I(X_1X_2X_3, Y) = E\left[\log \frac{p(y/x_1x_2x_3)}{p(y)}\right] \tag{8.1}$$

dove per semplificare la notazione si è indicato con $E[\cdots]$ il valor medio rispetto alle variabili binarie x_1, x_2 e x_3 e alla variabile continua y:

$$E[\cdots] = \sum_{x_1}\sum_{x_2}\sum_{x_3} P(x_1x_2x_3) \int_y p(y/x_1x_2x_3)\,[\cdots]\,dy \tag{8.2}$$

In questa forma non si vede alcuna influenza del *mapping*: l'informazione mutua, e quindi la capacità qualora si ottimizzino le probabilità dei punti della costellazione, dipende solo dall'insieme degli otto segnali e dal modello del rumore. Moltiplicando e dividendo per opportune densità di probabilità si può modificare la (8.1) nel seguente modo:

$$\begin{aligned} I(X_1X_2X_3, Y) &= E\left[\log\left(\frac{p(y/x_1)}{p(y)}\frac{p(y/x_1x_2)}{p(y/x_1)}\frac{p(y/x_1x_2x_3)}{p(y/x_1x_2)}\right)\right] = \\ &= I(X_1, Y) + I(X_2, Y/X_1) + I(X_3, Y/X_1X_2) \end{aligned} \tag{8.3}$$

L'interpretazione che si può dare di questa espressione è che l'informazione mutua, e quindi la capacità, è scomponibile in somma di tre termini relativi al solo primo bit, al secondo bit supponendo noto il primo, e al terzo bit supponendo noti i primi due.

Se la lunghezza N del blocco è elevata, ciascun livello può trasmettere ad un ritmo prossimo alla propria capacità. Quindi per ciascun bit si userà un codice con *rate* poco inferiore alla

[1] specificare la terna $x_1x_2x_3$ è del tutto equivalente a dare il punto trasmesso della costellazione; nel seguito x_1 è il bit meno significativo

capacità del bit stesso condizionata ai bit dei livelli inferiori. Infatti il primo codice può ridurre la probabilità che il primo bit sia errato a valori piccoli a piacere. Quindi nella decodifica del secondo codice è ragionevole assumere che il primo bit sia sempre corretto, e calcolare la capacità del secondo bit con tale ipotesi (e analogamente per il terzo bit si possono assumere dati i primi due).

Il primo bit ha una rappresentazione multipla: degli otto punti della costellazione, quattro corrispondono allo zero e quattro all'uno; il punto effettivamente trasmesso dipende dai bit x_2 e x_3, che il primo decodificatore assume sconosciuti. Anche il secondo bit ha una rappresentazione multipla: due punti corrispondono allo zero e due all'uno[2]; il punto trasmesso dipende da x_3. Infine il terzo bit corrisponde ad un solo punto per lo zero e uno solo per l'uno, essendo noti x_1 e x_2.

La teoria suggerisce quindi una decodifica *multistadio*: si decodificano i tre codici successivamente, ritenendo corretti i bit dei livelli inferiori. E' evidente che in pratica errori nel decodificare un livello possono produrre decisioni errate anche nei livelli superiori.

Finché si rispettano i limiti di capacità dei livelli, la tecnica di codifica e decodifica multistadio consente di avvicinarsi quanto si vuole alla capacità del canale, utilizzando per ogni livello codici binari (ad esempio turbo codici).

E' interessante osservare che la capacità complessiva è comunque quella della costellazione, e non dipende dal *mapping*. Invece le capacità dei singoli livelli, e quindi i codici da scegliere, dipendono dal *mapping*.

Per quanto riguarda il calcolo della informazione mutua (8.3) si può notare che i singoli termini sono la differenza di due entropie. Ad esempio $I(X_1, Y) = H(Y) - H(Y/X_1)$. Basta dunque calcolare l'entropia dell'uscita y considerando possibili tutti gli otto livelli e l'entropia di y dato x_1, cioè considerando possibili solo quattro livelli[3]. Analogamente si ha $I(X_2, Y/X_1) = H(Y/X_1) - H(Y/X_1X_2)$ e $I(X_3, Y/X_1X_2) = H(Y/X_1X_2) - H(Y/X_1X_2X_3)$. Si vede dunque che basta calcolare quattro entropie, e poi le loro differenze. Si noterà anche che l'ultima entropia, nel caso di rumore additivo indipendente dal segnale, non è altro che l'entropia del rumore.

Prima di mostrare alcuni esempi conviene chiedersi se sia possibile una strategia di decodifica più semplice, non multistadio ma *parallela*: decodificare indipendentemente i tre livelli, senza l'aiuto dei bit decodificati dai livelli inferiori. Per il primo livello non cambierebbe nulla. Il secondo livello, se si ignora il bit x_1, ha una rappresentazione multipla in cui quattro punti, anziché due, corrispondono a $x_2 = 0$ e quattro a $x_2 = 1$. Analogamente anche il terzo livello ha una rappresentazione multipla: quattro punti, anziché uno, corrispondono a $x_3 = 0$ e quattro a $x_3 = 1$. L'incertezza per i due livelli superiori è maggiore, e quindi le prestazioni sono peggiori.

Per calcolare le capacità dei tre bit nel caso parallelo basta procedere esattamente come per il primo bit nel caso multistadio. Infatti i bit di tutti i livelli hanno lo stesso tipo di

[2]non quattro e quattro, perché il primo bit è già stato decodificato e quindi è noto

[3]se $H(Y/x_1 = 0)$ e $H(Y/x_1 = 1)$ non coincidono occorre calcolare la media delle due entropie condizionate

rappresentazione multipla.

8.1.1 Esempi di capacità di costellazioni multilivello

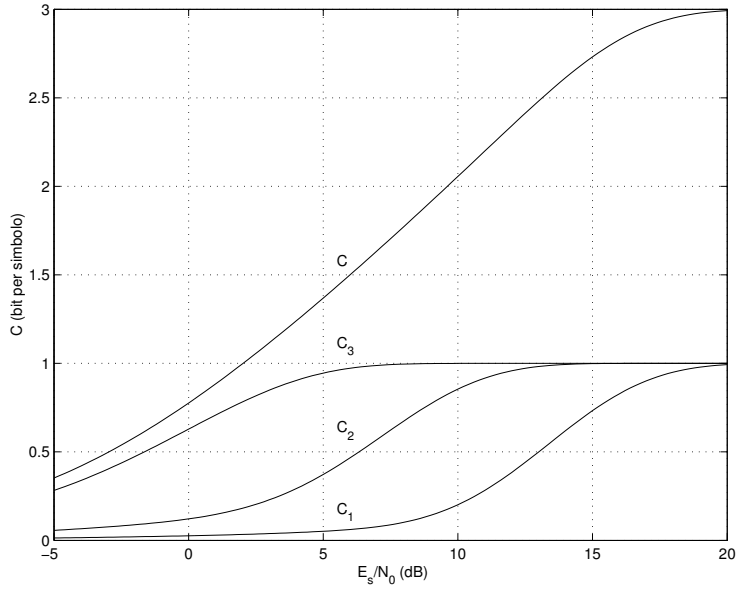

Figura 8.1: Capacità dei singoli bit e capacità complessiva della costellazione 8PAM con *mapping* binario naturale, con decodifica multistadio

La Fig. 8.1 mostra la capacità dei singoli bit e quella complessiva della costellazione 8PAM con *mapping* naturale e decodifica multistadio, supponendo per semplicità i bit x_1, x_2 e x_3, e quindi anche gli otto punti della costellazione, equiprobabili. La decodifica parallela comporterebbe una fortissima degradazione: basti pensare che se i bit x_1 e x_2 sono noti i segnali usati per trasmettere il terzo bit sono molto distanti, mentre se i primi due bit non sono noti possono essere persino adiacenti (011 e 100).

La Fig. 8.2 mostra le capacità con *mapping* di *Gray*, con decodifica sia multistadio sia parallela. Con questo *mapping* la degradazione con decodifica parallela è molto piccola.

Dalle due figure è anche evidente che il *mapping* naturale richiede tre codici con *rate* molto diversi, mentre quello di *Gray* dà *rate* più vicini tra loro. In compenso il *mapping* naturale richiede per il terzo livello *rate* più elevati, quindi codici più semplici o addirittura nessun codice.

Risultati analoghi per la costellazione 8PSK sono mostrati nelle Fig. 8.3 e 8.4. Con il *mapping* di *Gray* le capacità del secondo e terzo bit sono pressoché uguali, e si può anche pensare di usare per i due livelli la decodifica parallela e un unico codice con dimensione del blocco raddoppiata.

Infine la Fig. 8.5 mostra le capacità della costellazione 8PSK con il seguente *mapping* misto: 000, 010, 011, 001, 100, 110, 111, 101. I primi due livelli possono essere decodificati

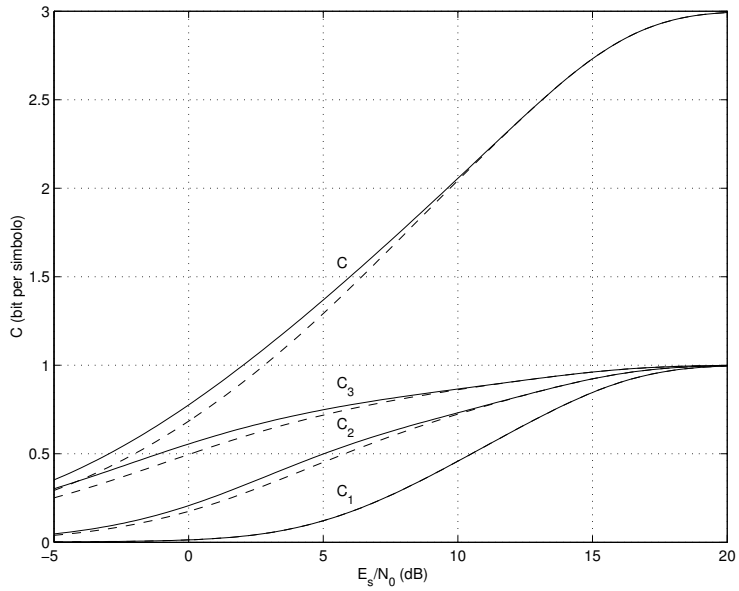

Figura 8.2: Capacità dei singoli bit e capacità complessiva della costellazione 8PAM con *mapping* di *Gray*, con decodifica multistadio (curve continue) e parallela (curve tratteggiate)

in parallelo. Inoltre hanno quasi la stessa capacità e quindi si può usare un unico codice di lunghezza doppia. Per il terzo bit è opportuna la decodifica multistadio.

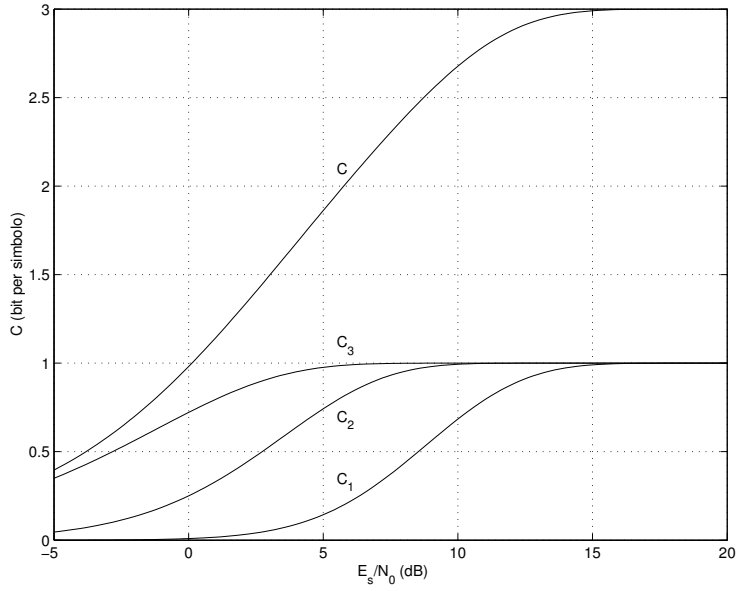

Figura 8.3: Capacità dei singoli bit e capacità complessiva della costellazione 8PSK con *mapping* naturale e decodifica multistadio

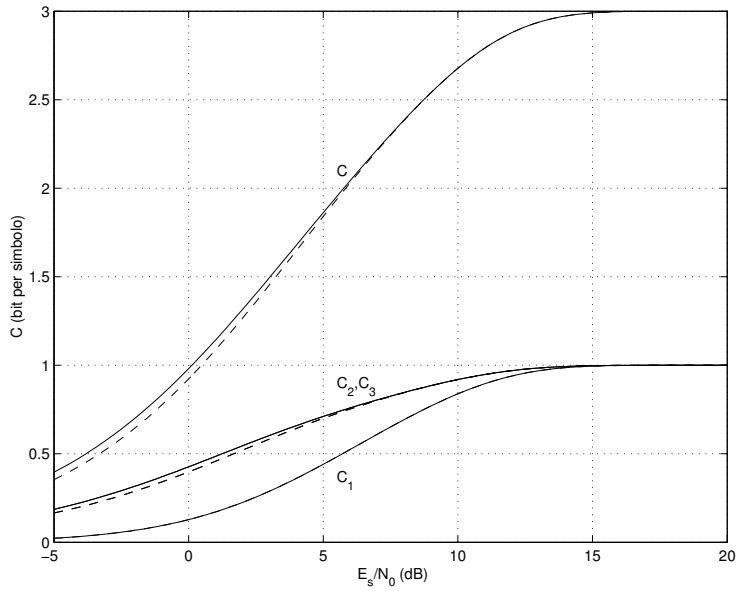

Figura 8.4: Capacità dei singoli bit e capacità complessiva della costellazione 8PSK con *mapping* di *Gray*, con decodifica multistadio (curve continue) e parallela (curve tratteggiate)

8.1.2 Demodulazione bit per bit

Si consideri il codice associato al bit x_1 di livello più basso di una costellazione di M punti. Il decodificatore deve ricevere in ingresso, per ciascun simbolo, la probabilità che x_1 valga

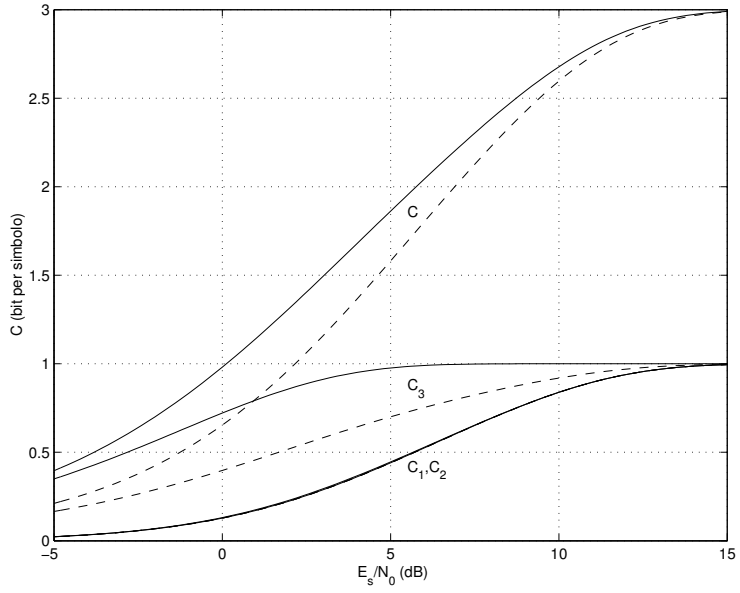

Figura 8.5: Capacità dei singoli bit e capacità complessiva della costellazione 8PSK con un *mapping* misto, con decodifica multistadio (curve continue) e parallela (curve tratteggiate)

zero o uno condizionata al vettore $\mathbf{r}$ ricevuto in corrispondenza del simbolo, o qualche altra grandezza equivalente.

Si noti la differenza tra valutare la probabilità a posteriori del *punto* della costellazione, data da (ignorando alcune costanti moltiplicative)

$$P(\mathbf{s}/\mathbf{r}) \equiv \exp\left(-\frac{|\mathbf{r}-\mathbf{s}|^2}{2\sigma^2}\right) \tag{8.4}$$

e quella del solo bit x_1, ottenuta sommando le probabilità (8.4) di tutti i punti della costellazione corrispondenti rispettivamente a $x_1 = 0$ e $x_1 = 1$. Il modo più comodo di liberarsi delle costanti moltiplicative è valutare il rapporto $P(x_1 = 0/\mathbf{r})/P(x_1 = 1/\mathbf{r})$, oppure il logaritmo di tale rapporto

$$\lambda_1 = \log \frac{\sum_{\mathbf{s}:x_1=0} \exp\left(-\frac{|\mathbf{r}-\mathbf{s}|^2}{2\sigma^2}\right)}{\sum_{\mathbf{s}:x_1=1} \exp\left(-\frac{|\mathbf{r}-\mathbf{s}|^2}{2\sigma^2}\right)} \tag{8.5}$$

detto *valore algebrico* del bit x_1 o talvolta anche *Log-Likelihood Ratio* (LLR). Si vede facilmente che il legame tra valori algebrici e probabilità è

$$P(x_1 = 0/\mathbf{r}) = \frac{\exp(\lambda_1)}{1+\exp(\lambda_1)} \qquad P(x_1 = 1/\mathbf{r}) = \frac{1}{1+\exp(\lambda_1)} \tag{8.6}$$

ma solitamente si organizza il calcolo ignorando i denominatori (perché comuni alle due ipotesi zero e uno). Dall'insieme dei valori algebrici il decodificatore è in grado di calcolare le probabilità di tutte le parole del codice.

Se i punti della costellazione hanno la stessa energia, come ad esempio nel caso 8PSK, si ha anche

$$\lambda_1 = \log \frac{\sum_{\mathbf{s}:x_1=0} \exp\left(\frac{\mathbf{r}\cdot\mathbf{s}}{\sigma^2}\right)}{\sum_{\mathbf{s}:x_1=1} \exp\left(\frac{\mathbf{r}\cdot\mathbf{s}}{\sigma^2}\right)} \tag{8.7}$$

Il valore algebrico λ_2 del secondo bit della costellazione è calcolato in modo analogo. Naturalmente si deve utilizzare nel calcolo la giusta costellazione: nel caso di decodifica parallela, l'intera costellazione; nel caso di decodifica multistadio, solo la metà dei punti che corrisponde al valore già deciso di x_1. Le stesse considerazioni valgono per i bit di livello superiore.

E' opportuno ricordare, come già visto nella Sez. 6.4, che nel caso delle costellazioni binarie in cui si ha un unico bit trasmesso con ampiezza ± 1 e il vettore ricevuto $\mathbf{r}$ ha una sola

componente r, il valore algebrico (supponendo che il livello positivo corrisponda allo zero) è proporzionale al campione ricevuto:

$$\lambda_1 = \log \frac{\exp\left(\frac{r}{\sigma^2}\right)}{\exp\left(-\frac{r}{\sigma^2}\right)} = \frac{2}{\sigma^2} r \tag{8.8}$$

In pratica per costellazioni generiche i valori algebrici svolgono il ruolo che nella trasmissione binaria hanno i campioni ricevuti.

8.1.3 *Bit interleaved coded modulation*

Con il *mapping* di *Gray*, che come si è visto consente la decodifica parallela con perdita di capacità trascurabile, si può addirittura utilizzare un *unico* codice binario per tutti i livelli. Ciò può sorprendere, perché i diversi livelli hanno capacità diverse e non si vede come un unico codice possa essere adatto a tutti i livelli. Evidentemente si può usare un solo codice solo se si fa in modo che i diversi bit della costellazione contribuiscano con *pesi diversi* alla decisione sull'intero blocco codificato.

Occorre anzitutto disperdere in modo sufficiente i bit di uno stesso blocco codificato in simboli diversi, in modo da ottenere l'indipendenza statistica dei valori algebrici[4].

Inoltre occorre fornire in ingresso al decodificatore i valori algebrici, che intrinsecamente tengono conto della diversa qualità dei bit a seconda del livello che occupano nella costellazione. Un modo intuitivo per capire la tecnica è pensare che i diversi bit abbiano un diverso rapporto segnale-rumore: quelli in posizione migliore hanno (mediamente) valori algebrici di modulo più elevato, mentre i bit in posizione peggiore hanno (mediamente) valori algebrici più piccoli. Anche questi sono utilizzati dal decodificatore, ma pesano meno nella decisione, come è giusto[5].

E' anche opportuno che una parola di codice utilizzi (circa) lo stesso numero di bit per ciascuna posizione nella costellazione, cioè evitare che una parola di codice possa essere trasmessa prevalentemente con bit *deboli*. Ciò si ottiene automaticamente se tutti i bit di ciascun simbolo sono usati (opportunamente dispersi) in uno stesso blocco codificato.

8.2 *Trellis coded modulation*

Un'altra tecnica di codifica per costellazioni multilivello che adotta un unico codice per più livelli è stata proposta alla fine degli anni '70. Utilizza il *mapping* binario naturale e

[4]come al solito si assume che il canale non abbia memoria e quindi simboli diversi siano trasmessi indipendentemente; idealmente i bit trasmessi da uno stesso simbolo dovrebbero appartenere a blocchi codificati diversi; in pratica non occorre un *interleaving* così drastico e si possono utilizzare tutti i bit della costellazione per uno stesso blocco

[5]qualcosa di simile si ha con i codici convoluzionali perforati: i bit non trasmessi contribuiscono alla decisione con peso nullo

un unico codice convoluzionale appositamente progettato tenendo conto delle proprietà di distanza dei diversi livelli.

Si consideri la costellazione 4PAM con *mapping* binario naturale: 00, 01, 10, 11. Detta 2 la distanza tra punti adiacenti, è immediato verificare che una differenza nel bit meno significativo (primo bit) garantisce $d^2 \geq 4$; se il primo bit coincide, una differenza nel secondo dà $d^2 = 16$.

Analogamente per la costellazione 8PAM una differenza nel primo bit garantisce $d^2 \geq 4$; nel caso il primo bit coincida, una differenza nel secondo dà $d^2 \geq 16$; se infine coincidono sia il primo sia il secondo bit, il terzo fornisce una distanza $d^2 = 64$.

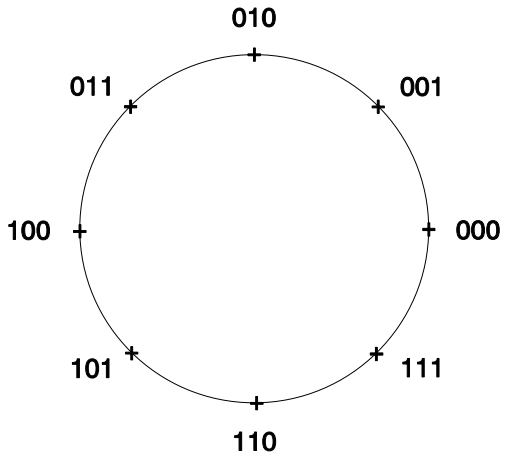

Figura 8.6: Costellazione 8PSK con *mapping* naturale

La costellazione 8PSK di Fig. 8.6 ha proprietà analoghe: detta 1 l'energia per simbolo, una differenza nel primo bit garantisce $d^2 \geq 0.586$; se il primo bit coincide, una differenza nel secondo dà $d^2 = 2$; se coincidono sia il primo sia il secondo bit, il terzo fornisce una distanza $d^2 = 4$.

Per le costellazioni QAM si consideri il *mapping* di Fig. 8.7. Normalizzando a 2 la distanza tra punti adiacenti, il primo bit differente garantisce $d^2 \geq 4$; nel caso coincida, una differenza nel secondo bit garantisce $d^2 \geq 8$; se coincidono primo e secondo bit, il terzo bit produce una distanza $d^2 \geq 16$; e se ancora occorresse (ma non avviene, di norma) il quarto bit potrebbe garantire $d^2 \geq 32$.

In tutti i casi il valore del primo bit suddivide l'insieme di punti della costellazione in due sottoinsiemi (*subset*) tali che la distanza tra un elemento qualsiasi del primo ed uno del secondo è $d^2 \geq d_1^2$. La distanza tra punti di uno stesso *subset* è invece $d^2 \geq d_2^2$, con $d_2^2 > d_1^2$. Entrambi i *subset* possono essere a loro volta partizionati secondo il valore del secondo bit. Si ottengono quattro *subset*, individuati dai primi due bit. La distanza tra punti di *subset* che differiscono nel primo bit è $d^2 \geq d_1^2$; tra *subset* con il primo bit uguale ma che differiscono nel secondo è $d^2 \geq d_2^2$; tra punti dello stesso *subset* $d^2 \geq d_3^2$, con $d_3^2 > d_2^2$.

Se la costellazione contiene almeno otto punti, l'operazione può essere ripetuta ottenendo

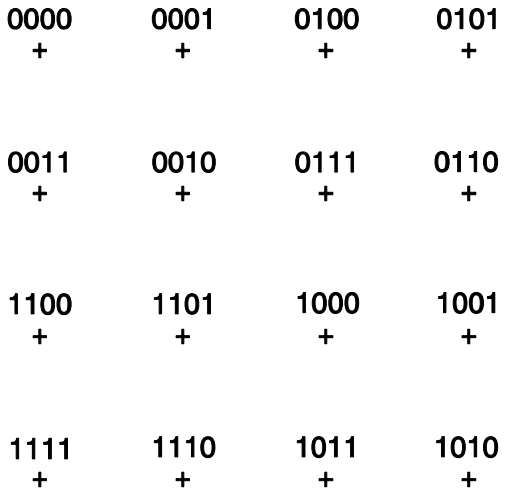

Figura 8.7: Costellazione 16QAM con *set partitioning*

otto *subset*, e di norma non occorre andare oltre. La distanza tra punti generici ...0 e ...1 è $d^2 \geq d_1^2$; tra ..0x e ..1x ($x = 0, 1$) è $d^2 \geq d_2^2$; tra .0xy e .1xy è $d^2 \geq d_3^2$; tra punti di uno stesso *subset* .xyz e .xyz è $d^2 \geq d_4^2$ (con $d_4^2 > d_3^2 > d_2^2 > d_1^2$; se i *subset* hanno un solo punto si suppone $d_4^2 = \infty$).

Per progettare il codice si scelgono i *subset* in modo tale che i punti all'interno di ciascuno siano già sufficientemente distanti da non richiedere ulteriore protezione. Punti appartenenti a *subset* diversi possono invece essere troppo vicini. Si seleziona la sequenza dei *subset* trasmessi mediante un codice convoluzionale, in modo da poter accumulare su più simboli una distanza sufficiente anche in questo caso. Questi codici, introdotti da Ungerboeck alla fine degli anni '70, sono spesso indicati con la sigla TCM (*Trellis Coded Modulation*).

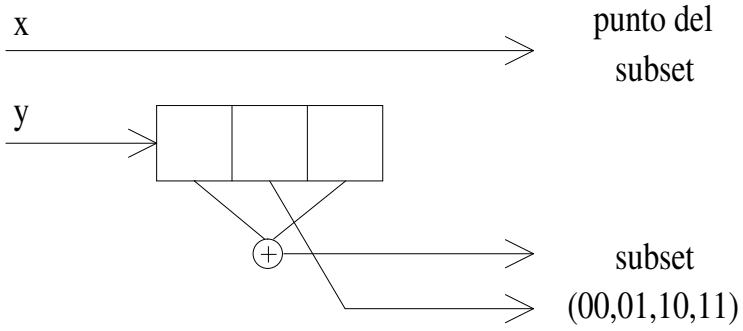

Figura 8.8: Codificatore TCM 8PSK a quattro stati

Un semplicissimo esempio con modulazione 8PSK, uno dei primi proposti, aiuterà a chiarire. I punti dei quattro *subset* sono $x00$, $x01$, $x10$ e $x11$ ($x = 0$ o 1). Ogni *subset* contiene due punti, e la distanza al quadrato tra questi è $4E_s$. Il primo bit diverso garantisce $d^2 \geq 0.586E_s$; se il primo coincide, una differenza nel secondo dà $d^2 = 2E_s$. La sequenza dei *subset* è selezionata dal codice convoluzionale in Fig. 8.8, alimentato dalla sequenza

$\{y\}$. Il codice è diverso dal miglior convoluzionale per modulazione binaria con *rate* 1/2 a quattro stati, perché non deve rendere massima la distanza di Hamming, ma piuttosto quella geometrica nello spazio dei segnali.

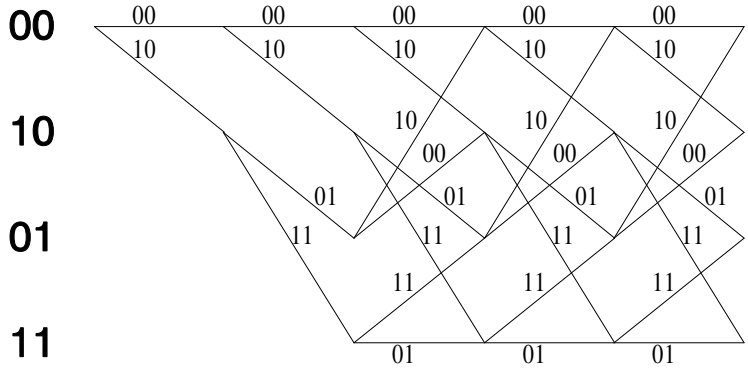

Figura 8.9: Traliccio del codificatore TCM 8PSK a quattro stati

In Fig. 8.9 è mostrato il traliccio, dove esistono (anche se non sono indicate esplicitamente) transizioni di stato *parallele*: se ad esempio il codice convoluzionale decreta che la transizione di stato sia da 00 a 00 e quindi che si debba trasmettere il *subset* 00, resta la possibilità di trasmettere la fase 000 oppure 100 a seconda del valore x del bit non codificato. Ogni ramo del traliccio corrisponde quindi ad una coppia di segnali possibili.

L'ispezione del traliccio, con l'aiuto delle proprietà del *set partitioning* mostra che la minima distanza per transizioni non parallele è $d^2 = 4.586E_s$, e si ha con la sequenza di *subset* 10,01,10.

La distanza corrispondente alle transizioni parallele, eventi errore che si esauriscono in un passo e producono un solo bit errato, è invece $d^2 = 4E_s$, e domina ad alto rapporto segnale-rumore. Sostituendo $E_s = 2E_b$ si vede che il codice ha un guadagno asintotico di 3 dB rispetto al 4PSK non codificato, *a parità di ritmo di trasmissione e banda*.

Le prestazioni del codice sono mostrate in Fig. 8.10: i punti indicati con un asterisco sono ottenuti mediante simulazione[6]; sono mostrate per confronto le prestazioni del 4PSK non codificato, e quelle valutate mediante lo *union bound* tenendo conto dell'unico evento errore a distanza minima, oppure anche degli otto con distanza $d^2 = 4.586E_s$. Questi ultimi danno un contributo non del tutto trascurabile perché sono più numerosi, hanno distanza poco superiore alla minima e producono un maggior numero di bit errati.

Il codice di Fig. 8.8 costellazione è adatto anche alla modulazione 8PAM, e fornisce un guadagno asintotico rispetto al 4PAM non codificato di 3.31 dB. In questo caso la distanza minima non viene dalle transizioni parallele, ma dal codice convoluzionale. Si possono migliorare le prestazioni aumentando il numero degli stati del codice: con 8 stati si può arrivare a 3.77 dB, con 16 stati a 4.18 dB, e così via fino a quasi 6 dB con 256 stati[7].

[6] per probabilità molto basse occorrono tecniche particolari, cosiddette di *importance sampling*

[7] non si dimentichi però che sono guadagni *asintotici*, di norma raggiunti abbastanza rapidamente nei codici

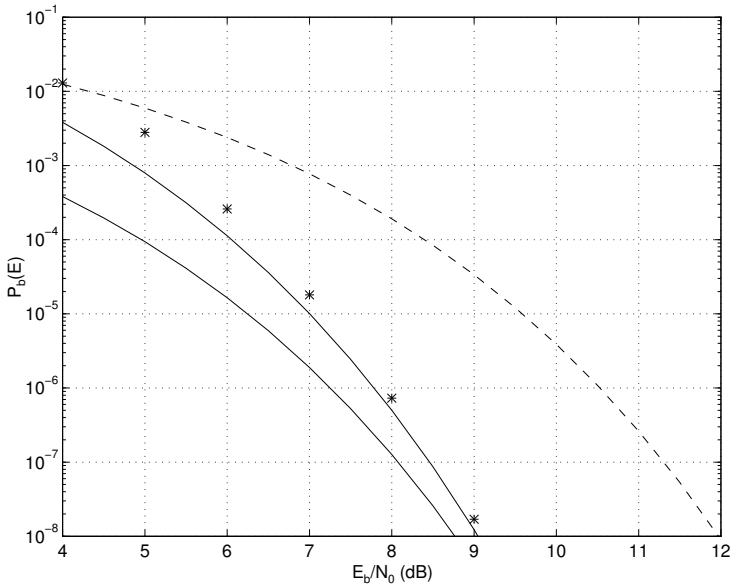

Figura 8.10: Probabilità d'errore per il codice TCM 8PSK a quattro stati; simulazione (asterischi) e *union bound* (curve continue: eventi errore a distanza minima, o fino a $d^2 = 4.586E_s$) a confronto con il 4PSK non codificato (tratteggio)

Con costellazione 8PSK risultano invece limitanti le transizioni parallele. Quindi almeno asintoticamente non serve a nulla migliorare il codice. Occorre piuttosto *eliminare le transizioni parallele*, cioè partizionare in 8 *subset* di un solo punto ciascuno. Non vi sono quindi bit non codificati (o *liberi*, come talvolta si dice) e il codice convoluzionale ha *rate* effettivo 2/3. Con 8 stati si riesce a ottenere un guadagno asintotico di 3.60 dB, e con 256 stati di 5.75 dB.

Per le costellazioni QAM valgono considerazioni analoghe. I codici più semplici, a 4 stati, richiedono 4 *subset* e quindi codificatori con *rate* 1/2, e danno guadagni asintotici intorno a 3 dB. A partire da 8 stati occorre partizionare in 8 *subset*. Se quindi ad esempio si vogliono trasmettere 5 bit per simbolo (2.5 bit per dimensione) si usa una costellazione di $2 \cdot 2^5 = 64$ punti, partizionata in 8 *subset* di 8 punti ciascuno; ci sono quindi 8 transizioni parallele. Due bit d'informazione entrano nel codificatore convoluzionale con *rate* $R = 2/3$. Escono tre bit che selezionano il *subset*. I tre bit non codificati selezionano il punto all'interno del *subset*.

Si noti quanto poco cambierebbe se si volessero trasmettere 7 bit per simbolo (3.5 bit per dimensione). Si partizionerebbe un 256QAM in 8 *subset* di 32 punti, e si avrebbero 32 transizioni parallele. I tre bit uscenti dal codificatore sceglierebbero, come prima, il *subset* e i cinque non codificati il punto nel *subset*. In un certo senso si può dire che per le alte capacità il codice è *universale*, cioè praticamente indipendente dalla dimensione della

semplici e sempre più lentamente nei casi più complessi

costellazione.

Anche per le costellazioni QAM i guadagni asintotici con 256 stati risultano intorno ai 6 dB.

Il ricevitore può essere realizzato come per i codici convoluzionali, con l'unica novità delle transizioni parallele. Risulta conveniente determinare come primo passo per ogni gruppo di transizioni parallele il punto del corrispondente *subset* che risulta più vicino al vettore ricevuto. Dopo aver così sfoltito il grafo, lasciando un solo candidato per ciascuna transizione di stato, si applica l'usuale algoritmo di Viterbi.

Per la valutazione delle prestazioni valgono considerazioni analoghe ai codici convoluzionali, con l'unica avvertenza di contare con opportuna molteplicità nello *union bound* le transizioni parallele. La distanza minima può dipendere dalla sequenza trasmessa e può talvolta risultare maggiore di quella garantita dal *set partitioning*; tuttavia quasi sempre si trovano sequenze per cui vale l'uguaglianza, e quindi si deve fare conto solo sulla distanza minima garantita. Analogamente il numero di eventi errore a distanza minima può dipendere dalla sequenza trasmessa.

Infine per la ricerca di buoni codici Ungerboeck ha proposto alcune regole empiriche che si sono rivelate molto valide. Infatti con l'indagine esaustiva si trova poco di meglio. Per le tabelle dei migliori codici si rimanda ai testi specializzati. Occorre però osservare che la notazione di Ungerboeck è diversa da quella qui utilizzata: i codificatori sono dati in forma recursiva sistematica.