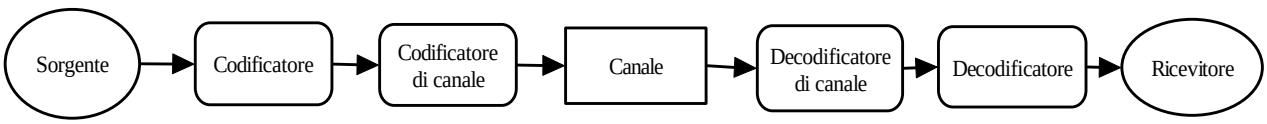
Appunti di Comunicazioni Elettriche
Capitolo 9 - Teoria dell'informazione

INTRODUZIONE ALLA TEORIA DELL'INFORMAZIONE

Consideriamo lo schema di un generico sistema di trasmissione numerico:



C'è una "*sorgente*" che genera un segnale analogico (televisivo, telefonico, musicale e così via) che deve arrivare al ricevitore (cioè all'utente); tale segnale viene per prima cosa convertito in forma numerica, da un dispositivo detto **codificatore**. L'uscita di questo dispositivo è dunque la sequenza di bit contenente l'informazione che intendiamo far arrivare al ricevitore. Questa sequenza di bit non viene però trasmessa così com'è, ma viene preventivamente elaborata dal **codificatore di canale**, il quale la manipola essenzialmente allo scopo di proteggerla dagli inevitabili errori dovuti alla trasmissione.

E' opportuno sottolineare le differenze di comportamento tra il "codificatore" ed il "codificatore di canale": il primo riceve in ingresso il segnale emesso dalla sorgente e genera in uscita la sequenza di bit corrispondente; il secondo, invece, riceve in ingresso la sequenza di bit emessa dal codificatore e opera su di essa quelle manipolazioni che permetteranno al decodificatore di canale di recuperare eventuali errori dovuti alla trasmissione tramite il canale.

Il segnale emesso dal codificatore di canale viene inviato al mezzo trasmissivo, il quale rappresenta tutti quei dispositivi necessari per la trasmissione del segnale stesso.

L'uscita del canale fa da ingresso per il **decodificatore di canale**, che rappresenta la classica cascata tra il filtro di ricezione, il campionatore ed il decisore: il compito di questo dispositivo è quello di ricostruire la sequenza di bit che è stata trasmessa, cercando, nel contempo, di rilevare ed eventualmente correggere gli inevitabili errori introdotti dal canale.

L'uscita del decodificatore di canale è dunque la stessa sequenza di ingresso presumibilmente emessa dal codificatore. Segue allora un ulteriore dispositivo, detto **decodificatore**, che si comporta in modo inverso al codificatore: tale dispositivo riceve in ingresso la sequenza di bit "ripulita", tramite il decodificatore di canale, degli errori dovuti alla trasmissione e, da questa sequenza, ricostruisce il segnale analogico emesso dalla sorgente inviandolo al ricevitore.

SORGENTE DISCRETA SENZA MEMORIA

*Lo scopo della **teoria dell'informazione** è valutare i limiti teorici dell'informazione che si può trasmettere su un canale preassegnato. Questo serve per poter stabilire dei parametri comuni utili per confrontare le prestazioni dei sistemi di trasmissione reali.*

Mettiamoci nelle seguenti ipotesi:

- in primo luogo, supponiamo che la codifica del segnale analogico emesso dalla sorgente sia **binaria**: ciò significa che al segnale emesso dalla sorgente verrà sempre associata, da parte del *codificatore*, una sequenza di bit;
- supponiamo che la sorgente non emetta in modo continuo nel tempo il proprio segnale, ma solo in istanti successivi discreti;
- supponiamo infine che il segnale emesso dalla sorgente sia costituito da una successione di **simboli** discreti appartenenti al cosiddetto **alfabeto** della sorgente stessa. Per capirci meglio, se supponiamo che l'alfabeto della sorgente sia genericamente $X = \{x_1, x_2, \ldots, x_N\}$, l'ipotesi che stiamo facendo, e che definisce la cosiddetta "**sorgente discreta**", è che la sorgente emetta, in istanti di tempo discreti e uno alla volta, i simboli appartenenti ad X (in un ordine che dipende dal tipo di informazioni che si intende trasmettere).

Un'altra ipotesi semplificativa con la quale cominciamo a fare i nostri ragionamenti, è che i simboli emessi dalla sorgente siano indipendenti tra loro: ciò significa che ogni simbolo emesso è indipendente dai simboli emessi in precedenza. In termini quantitativi, possiamo esprimerci dicendo che la probabilità di emettere un simbolo ad un certo istante è sempre la stessa, a prescindere da

quali simboli siano stati emessi in precedenza. Una sorgente che gode di questa proprietà prende il nome di **sorgente (discreta) senza memoria**.

Indichiamo allora con $P(x_i)$ la probabilità che la sorgente emetta il generico simbolo x_i: se supponiamo che l'alfabeto della sorgente comprenda N simboli (con N finito o infinito numerabile), avremo allora N probabilità di trasmissione

$$P(x_1), P(x_2), .., P(x_N)$$

dove ovviamente sussiste la relazione

$$\sum_{i=1}^{N} P(x_i) = 1$$

E' chiaro infatti che la sorgente può emettere o il simbolo x_1 o il simbolo x_2 e così via, per cui la probabilità di emettere almeno uno dei simboli dell'alfabeto (ossia il termine a primo membro di quella relazione) è pari ad 1.

Se supponiamo che gli N simboli siano equiprobabili, abbiamo evidentemente che

$$P(x_1) = P(x_2) = .. = P(x_N) \quad \longrightarrow \quad P(x_i) = \frac{1}{N}$$

Sotto questa ipotesi, dovendo codificare in modo binario ciascuno di questi simboli, non possiamo far altro che usare, per ognuno di essi, configurazione binarie formate da uno stesso numero di bit: per N simboli, avremo bisogno quindi di N configurazioni binarie e quindi di un numero di bit pari a $n = \log_2 N$. Ad esempio, se la sorgente ha N=128 simboli, allora servono configurazioni da $n = \log_2 128 = 7$ bit/simbolo.

Quindi, la quantità **n bit/simbolo** rappresenta, di fatto, la quantità di informazione emessa dalla sorgente. Se è nota la velocità (espressa in simboli/sec) con cui la sorgente emette i propri simboli, avremo anche una misura del numero di bit trasmessi nell'unità di tempo, cioè della frequenza di cifra:

$$f_S\left(\frac{\text{bit}}{\text{sec}}\right) = n\left(\frac{\text{bit}}{\text{simbolo}}\right) \bullet v\left(\frac{\text{simboli}}{\text{sec}}\right)$$

Le cose cambiano, invece, se i simboli non sono più equiprobabili. In questo caso, infatti, è sicuramente molto più ragionevole usare una **codifica a lunghezza variabile**: ai simboli meno probabili faremo corrispondere parole di bit più lunghe, mentre ai simboli più probabili faremo corrispondere parole di bit più corte. Con questo sistema, potremo ancora definire un *numero medio di bit per unità di tempo*, che sarà sicuramente più basso del numero di bit per unità di tempo che otterremmo associando a tutti i simboli configurazioni binarie di uguale lunghezza.

Possiamo dunque definire, nel caso di simboli non equiprobabili, una quantità di informazione media, emessa dalla sorgente, pari a

$$\boxed{I(x) = \sum_{i=1}^{N} P(x_i) n_i}$$

dove $P(x_i)$ è la probabilità di emissione del generico simbolo x_i e n_i il numero di bit associati allo stesso x_i.

Autore: Sandro Petrizzelli

ENTROPIA (DEL PRIMO ORDINE) DELLA SORGENTE

Per sapere quanto "vale" l'informazione emessa dalla sorgente, si usa il cosiddetto **teorema dell'equipartizione**, che andiamo a enunciare.

Intanto, supponiamo che la sorgente sia **stazionaria**, ossia che le sue proprietà statistiche non varino nel tempo. In secondo luogo, supponiamo che la sorgente sia anche **ergodica**: in base alla nota definizione di ergodicità di un processo stocastico, questo significa che è possibile ricavare tutte le proprietà statistiche della sorgente a partire da una sola realizzazione[1]. In concreto, *l'ergodicità della sorgente significa che, osservando una sola realizzazione per un tempo via via crescente, si può essere sicuri che la sorgente passerà attraverso tutti i suoi possibili stati.*

Questo ci consente di fare il seguente ragionamento: se consideriamo un messaggio composto da M simboli, con M molto grande, con probabilità 1 tale messaggio conterrà tutti i simboli della sorgente, che compariranno ognuno un certo numero di volte: se M_i è il numero di volte in cui compare il generico simbolo x_i, risulterà cioè

$$M = \sum_{i=1}^{N} M_i$$

dove ricordiamo ancora che N sono i simboli della sorgente[2].

Osserviamo che il generico $\mathbf{M_i}$ non è altro, per definizione, che $\mathbf{Mp_i}$, cioè il prodotto del numero totale di simboli da cui è composto il messaggio per la probabilità pi del simbolo considerato.

Considerato adesso il generico messaggio in esame, possiamo anche valutare la probabilità $P_{mess,k}$ che esso venga effettivamente emesso: infatti, tale probabilità è pari alla probabilità che il simbolo x_1 venga emesso M_1 volte, che il simbolo x_2 venga emesso M_2 volte e così via fino all'N simbolo; dato, però, che l'emissione di un simbolo è del tutto indipendente dall'emissione degli altri, allora gli N eventi "simbolo x_1 emesso M_1 volte", "simbolo x_2 emesso M_2 volte" e così via sono indipendenti tra loro, il che significa che possiamo scrivere che

$$P_{mess,k} = \prod_{i=1}^{M} p_i^{M_i} = p_1^{M_1} p_2^{M_2} .... p_N^{M_N}$$

dove abbiamo indicato sinteticamente con p_i la probabilità di emissione del generico x_i.

Poter valutare $P_{mess,k}$, per ciascun messaggio, è importante in quanto non tutti i messaggi sono possibili. Alcuni di essi avranno probabilità 0 di verificarsi: ad esempio, un messaggio formato da tutti simboli uguali non si verificherà mai, in quanto non porta con sé alcuna informazione.

Dobbiamo dunque considerare solo i messaggi che hanno una probabilità maggiore di 0 di essere emessi: questi messaggi, sempre per la supposta ergodicità della sorgente, non possono che essere equiprobabili: se, allora, indichiamo con m il numero di messaggi con probabilità non nulla di essere emessi, possiamo scrivere che

$$1 = \sum_{k=1}^{m} P_{mess,k} = P_{mess,k} \cdot m = P_{mess} \cdot m \quad \longrightarrow \quad \boxed{m = \frac{1}{P_{mess}}}$$

[1] Con questo si intende dire che è ergodico il processo stocastico corrispondente all'emissione di simboli da parte della sorgente: ogni messaggio emesso dalla sorgente è una realizzazione di questo processo stocastico, per cui l'ergodicità significa che, osservando un singolo messaggio, di durata sufficiente, si può risalire alle caratteristiche statistiche del processo di emissione dei simboli.

[2] Questo accade perché, per definizione di ergodicità, per M tendente all'infinito, le cosiddette *frequenze relative* M_i/M dei vari simboli devono tendere alle rispettive probabilità.

Abbiamo cioè concluso che *il numero di messaggi possibili è pari al reciproco della probabilità di emissione del generico di essi.*

Così come possiamo pensare di associare un certo numero di bit ad ogni simbolo della sorgente, lo stesso possiamo fare per un intero messaggio: dato che i messaggi sono in numero finito e sono equiprobabili, ognuno di essi sarà codificato da uno stesso numero di bit, che vale in questo caso

$$n_{mess} = \log_2 m = \log_2 \frac{1}{P_{mess}} = -\log_2 P_{mess} \qquad \left[\frac{\text{bit}}{\text{messaggio}}\right]$$

D'altra parte, abbiamo detto che il messaggio è composto da M simboli, per cui il numero medio di bit necessari per descrivere il singolo simbolo sarà

$$H(x) = \frac{n_{mess}}{M} = -\frac{\log_2 P_{mess}}{M} \qquad \left[\frac{\text{bit}}{\text{simbolo}}\right]$$

Questa quantità prende il nome di **entropia del primo ordine** della sorgente considerata: essa rappresenta la quantità di informazione media per simbolo. Possiamo anche esplicitarla meglio, sostituendo l'espressione di P_{mess} ricavata prima:

$$H(x) = -\frac{1}{M}\log_2 \prod_{i=1}^{M} p_i^{M_i} = -\frac{1}{M}\sum_{i=1}^{M} \log_2 p_i^{M_i} = -\frac{1}{M}\sum_{i=1}^{M} M_i \log_2 p_i$$

Portando dentro la sommatoria il fattore M, otteniamo il rapporto M_i/M, che è la cosiddetta **frequenza relativa** del generico x_i, ossia il numero di volte in cui esso compare nel messaggio considerato. D'altra parte, in base sempre all'ergodicità, la frequenza relativa del generico simbolo, per M molto grande, tende alla probabilità del simbolo stesso, per cui possiamo concludere che l'entropia del primo ordine della sorgente è

$$\boxed{H(x) = \sum_{i=1}^{M} p_i \log_2 p_i}$$

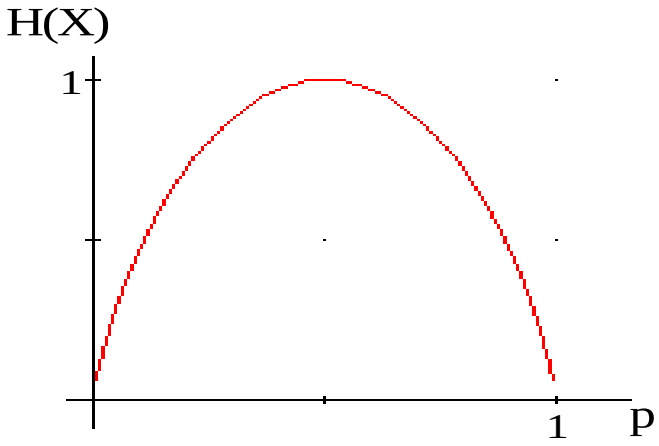
Confrontiamo adesso questa relazione con la relazione $I(x) = \sum_{i=1}^{N} p_i n_i$ trovata prima per sorgenti con simboli non equiprobabili: si deduce che la quantità $-\log_2 p_i$ rappresenta il numero minimo teorico di bit necessario a descrivere un simbolo. Di conseguenza, possiamo affermare che *l'entropia rappresenta il minimo numero medio di bit per simbolo necessari per trasmettere.*

Questo è un concetto importante, in quanto ci dice che, *per avvicinarci al limite teorico indicato appunto dalla H(x), dobbiamo ricorrere a codifiche a lunghezza variabile.*

Tornando adesso all'espressione dell'entropia, si nota che il suo valore numerico dipende dalle probabilità di emissione della sorgente stessa e dal numero di simboli di cui la sorgente dispone. Per esempio, supponiamo che la sorgente abbia solo due simboli (N=2); dato che la somma delle probabilità di emissione deve essere pari ad 1, è chiaro che, se p è la probabilità di emettere uno dei due simboli, 1-p sarà la probabilità di emettere l'altro simbolo. Allora, il valore dell'entropia risulta essere

$$H(X) = \sum_{i=1}^{2} p_i \log_2 \frac{1}{p_i} = p \log_2 \frac{1}{p} + (1-p)\log_2 \frac{1}{1-p}$$

Al variare di p avremo dunque valori diversi di entropia. Possiamo diagrammare H(X) in funzione di p (dove p varia ovviamente tra 0 ed 1):



Si nota quindi che l'entropia assume il suo massimo valore quando p=0.5, ossia quando i due simboli di cui dispone la sorgente sono equiprobabili, mentre assume decrescenti, in modo simmetrico, quando p aumenta o diminuisce rispetto a 0.5.

RICHIAMI DI TEORIA DEI SEGNALI

Concetto di "informazione"

Ogni simbolo emesso dalla sorgente costituisce chiaramente una "*informazione*" che la sorgente emette perché venga trasmessa al ricevitore. Il tipo di informazione è assolutamente generico, nel senso che la sorgente può trasmettere informazioni di qualsiasi tipo. Dovendo però studiare il funzionamento della sorgente, è utile rappresentare in qualche modo tale informazione dal punto di vista matematico. Dato allora il generico simbolo x_i, possiamo definire, in termini matematici, l'informazione ad essa associata nel modo seguente:

$$x_i \longrightarrow I(x_i) = \log_2 \frac{1}{P(x_i)}$$

Si tratta adesso di giustificare questa formula. Le giustificazioni che possiamo dare sono tutte di natura essenzialmente intuitiva.
Per esempio, supponiamo che la sorgente considerata abbia un solo simbolo e quindi trasmette solo quello: è evidente, allora, che questa sorgente, di fatto, non trasmette alcuna informazione, proprio perché emette sempre la stessa cosa. In termini matematici, questo significa che l'informazione associata all'unico simbolo deve valere 0 e questo effettivamente accade in base alla definizione di $I(x_i)$: infatti, se la sorgente possiede un solo simbolo, è ovvio che $P(x_i) = 1$ (in quanto c'è la certezza che emetta quell'unico simbolo); andando allora a sostituire nell'espressione di $I(x_i)$, si trova che quest'ultima è proprio pari a 0.
Ora supponiamo di considerare due generici simboli x_i e x_j tra quelli appartenenti all'alfabeto della sorgente; supponiamo anche che il primo abbia meno probabilità del secondo di essere trasmesso, ossia supponiamo che $P(x_i) < P(x_j)$. Questo significa che, dato un qualsiasi messaggio emesso dalla

sorgente, il simbolo x_i compare in media MENO volte rispetto al simbolo x_j; se compare meno volte, è lecito aspettarsi che l'informazione associata a tale simbolo sia PIU' importante di quella associata al simbolo x_j, ossia è lecito aspettarsi che sia $I(x_i) > I(x_j)$. Effettivamente questo accade: infatti, poiché $P(x_i) < P(x_j)$, andando a sostituire nella definizione di $I(x_i)$, noi troviamo che

$$I(x_i) = \log_2 \frac{1}{P(x_i)} > I(x_j) = \log_2 \frac{1}{P(x_j)}$$

Un'ultima giustificazione della definizione di $I(x_i)$ potrebbe essere la seguente: supponiamo che la sorgente emetta i due simboli x_i e x_j in modo del tutto indipendente dall'altro (come noi stiamo supponendo). Allora, noi siamo portati a credere che l'informazione complessiva associata alla coppia (x_i, x_j) corrisponda alla somma delle informazioni. Anche in questo caso, questo si verifica: infatti, indicata con

$$I(x_i, x_j) = \log_2 \frac{1}{P(x_i \cap x_j)}$$

l'informazione associata alla coppia, si ha che

$$I(x_i, x_j) = \log_2 \frac{1}{P(x_i)P(x_j)} = \log_2 \frac{1}{P(x_i)} \frac{1}{P(x_j)} = \log_2 \frac{1}{P(x_i)} + \log_2 \frac{1}{P(x_j)} = I(x_i) + I(x_j)$$

Entropia della sorgente

Consideriamo adesso la nostra sorgente discreta e adottiamo sempre l'ipotesi per cui ciascun simbolo emesso sia indipendente dai simboli emessi precedentemente. Si definisce allora "**entropia del primo ordine**" della sorgente la quantità

$$H(X) = \sum_{i=1}^{N} P(x_i) I(x_i)$$

dove N è il numero di simboli che la sorgente è in grado di emettere, $P(x_i)$ la probabilità di emettere il generico simbolo x_i e $I(x_i)$ l'informazione associata a tale simbolo.

Avendo detto che $I(x_i) = \log_2 \frac{1}{P(x_i)}$ e ponendo per semplicità $p_i = P(x_i)$, possiamo anche scrivere che

$$H(X) = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i}$$

L'entropia è una grandezza importante per una serie di motivi. Il primo di questi è che essa è legata al numero N di simboli di cui dispone la sorgente dalla seguente relazione (che si può dimostrare facilmente):

$$\boxed{H(X) \leq \log_2 N}$$

In particolare, il simbolo di uguaglianza vale SOLO quando tutti i simboli sono equiprobabili.

Autore: Sandro Petrizzelli

RICHIAMI SUI CODICI BINARI

Abbiamo detto che il **codificatore** ha il compito di associare, alla sequenza di simboli emessi dalla sorgente, una opportuna sequenza di bit. E' ovvio che questa sequenza di bit deve essere tale che, una volta arrivata al decodificatore, quest'ultimo sia in grado di ricostruire esattamente (a meno chiaramente degli errori), a partire da essa, il segnale emesso dalla sorgente. Questo è possibile solo se il codice binario utilizzato gode della proprietà di essere "**univocamente decodificabile**": deve cioè essere tale che la decodifica possa essere una ed una sola, in modo da avere la garanzia che il ricevitore riceva ciò che la sorgente ha emesso.

I codici ai quali si può pensare sono infiniti. In linea di massima, i requisiti che un codice deve avere, oltre appunto alla "univoca decodificabilità", sono i seguenti:

- deve prevedere il più basso numero di bit possibile, in modo da ridurre i tempi di trasmissione;
- deve inoltre consentire di "risparmiare in banda".

A requisiti di questo genere rispondono sia i **codici a lunghezza fissa** (nei quali ad ogni simbolo viene associato sempre lo stesso numero di bit[3]) sia i **codici a lunghezza variabile** (nei quali invece il numero di bit associati a ciascun simbolo è maggiore nei simboli meno probabili).

Metodo ad albero per la ricerca di codici univocamente decodificabili

Sorge subito una domanda: come si fa ad ottenere un codice a lunghezza variabile che sia univocamente decodificabile?

Un risultato importante che si può dimostrare a questo proposito è il seguente: *condizione sufficiente affinché un codice a lunghezza variabile sia univocamente decodificabile è che ogni parola del codice NON sia il prefisso di alcuna altra parola.*

A prescindere dalla dimostrazione matematica, è intuitivo comprendere perché sussiste questo risultato. Consideriamo, per esempio, una sorgente con 4 soli simboli (A,B,C,D), codificata con il seguente codice:

$$A \longrightarrow 0$$
$$B \longrightarrow 01$$
$$C \longrightarrow 10$$
$$D \longrightarrow 100$$

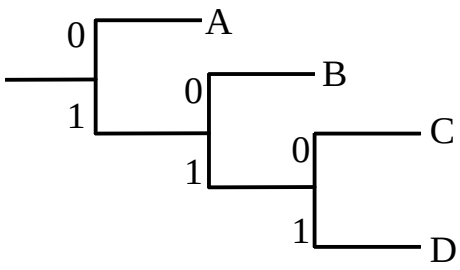
In questo codice, un bit 0 può essere preso come il prefisso della parola associata ad A oppure di quella associata a B, oppure la sequenza 10 poteva essere presa come prefisso per C o come prefisso per D. Di conseguenza, il codice non è univocamente decodificabile.

Naturalmente, trattandosi di una condizione sufficiente, è possibile che siano univocamente decodificabili codici che non abbiano quel requisito.

Vediamo allora come si può ideare un codice binario che soddisfi a quella condizione. Il metodo migliore consiste nell'usare la **tecnica ad albero**.

Supponiamo che l'alfabeto della nostra sorgente sia ancora una volta $\{A, B, C, D\}$. Tracciamo allora uno "**schema ad albero**" del tipo seguente:

[3] Un esempio tipico è la codifica ASCII dei caratteri



Leggendo quest'albero da sinistra verso destra, otteniamo il seguente codice:

$$A \longrightarrow 0$$
$$B \longrightarrow 10$$
$$C \longrightarrow 110$$
$$D \longrightarrow 111$$

Questo codice, che è evidentemente a lunghezza variabile, gode della proprietà per cui nessuna è parola è prefisso di qualche altra, per cui noi siamo certi che si tratti di un codice univocamente decodificabile.

Vediamo allora come si comporta il decodificatore: supponiamo che il messaggio sia

A B D A C

La codifica binaria di questo messaggio è

0 10 111 0 110

Il primo bit ricevuto dal decodificatore è lo 0: lo 0 compare solo nella parola corrispondente ad A ed è anche l'unico termine di tale parola, per cui il decodificatore genera subito il simbolo A. Il bit successivo è 1: questo può essere il primo bit delle parole corrispondenti a B,C e D, per cui è necessario considerare il bit successivo; questo è uguale a 0: la sequenza 10 compare solo nella parola corrispondente a B, per cui il decodificatore genera immediatamente B.

Il bit ancora successivo è 1: anche qui vale lo stesso discorso di prima, per cui bisogna considerare il bit successivo, che vale ancora 1; la sequenza 11 compare come prefisso delle parole associate a C e a D, per cui bisogna considerare il bit successivo, che vale sempre 1: a questo punto, la possibilità è una sola e cioè che tali tre bit corrispondano al simbolo D che quindi viene emesso. Continuando in questo modo, viene ricostruito con esattezza il messaggio emesso dalla sorgente.

Numero medio di bit

Il metodo ad albero appena esposto è uno dei metodi con i quali si può realizzare un codice, a lunghezza fissa o variabile, che sia certamente univocamente decodificabile. Viene allora da chiedersi, dati due codici a lunghezza variabile univocamente decodificabili, quale convenga scegliere. Sicuramente, un parametro fondamentale di giudizio è il numero di bit che il codice associa ai simboli. In particolare, dato che si tratta di codici a lunghezza variabile, il parametro da considerare è il "**numero medio di bit**" associati ai simboli del codice, che possiamo definire come

$$\overline{N} = \sum_{i=1}^{N} n_i p_i$$

dove n_i è il numero di bit che il codice associa al simbolo i-simo, mentre p_i è la probabilità che la sorgente emetta tale simbolo (e ricordiamo a questo proposito che siamo sempre nella ipotesi che tale probabilità sia indipendente dai simboli emessi in precedenza, ossia nella ipotesi di sorgente senza memoria). E' evidente,

dunque, che $\overline{N}$ coincide con la quantità che abbiamo inizialmente definito come I(x), cioè come una quantità di informazione media emessa dalla sorgente.

Il criterio di scelta è allora quello di rendere minimo il valore di $\overline{N}$, in modo da rendere anche minimo il tempo richiesto per la trasmissione del messaggio. Per esempio, se supponiamo che la nostra sorgente emetta un messaggio composto da k simboli, il numero medio di bit da trasmettere sarà $k\overline{N}$.

Facciamo osservare, come sarà chiaro anche dagli esempi, che anche piccole variazioni di $\overline{N}$ sono comunque importanti: ad esempio, supponiamo che il messaggio da trasmettere sia composto da k=10000 simboli; allora il numero medio di simboli da trasmettere è $10000\overline{N}$; se $\overline{N} = 2.5$, tale numero medio vale 25000, mentre, se $\overline{N} = 2.8$, esso vale 28000. Abbiamo perciò una differenza di 3000 simboli che non è irrilevante, specialmente poi se la velocità del canale binario non è elevatissima.

LA CODIFICA DI HUFFMAN

Il problema della ideazione di un codice a lunghezza variabile che abbia il minimo valore possibile di $\overline{N}$ è stato studiato e ottimizzato mediante una tecnica di codifica che prende il nome di "**codifica di Huffman**". Questa tecnica usa uno schema ad albero del tipo visto prima, ma in modo più particolareggiato e questo al fine proprio di ottenere il minimo valore possibile per $\overline{N}$.

Supponiamo ancora una volta che l'alfabeto della nostra sorgente sia composto da N=4 simboli e precisamente $\{A, B, C, D\}$. Supponiamo anche di conoscere le probabilità p_i di trasmissione di tali simboli: ad esempio, prendiamo

$$p_A = 0.1$$
$$p_B = 0.3$$
$$p_C = 0.4$$
$$p_D = 0.2$$

Per costruire l'albero, cominciamo a disporre i 4 simboli dell'alfabeto, con le rispettive probabilità, in ordine di probabilità decrescente, ossia partendo dal più probabile e andando verso il meno probabile: nel nostro caso, la scala risulta essere

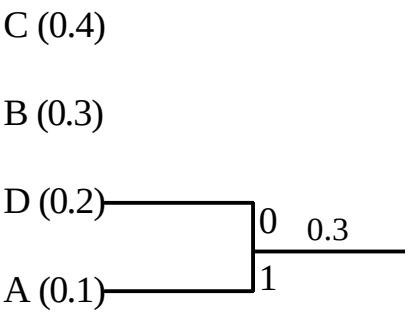
C (0.4)

B (0.3)

D (0.2)

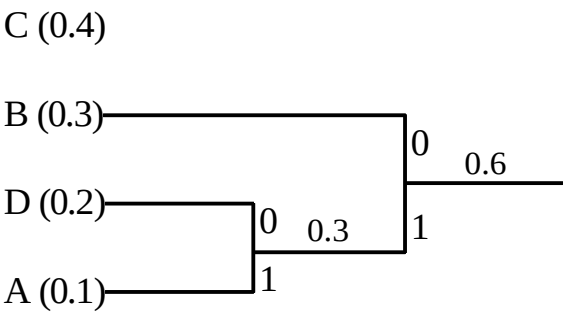
A (0.1)

Adesso, consideriamo i due simboli con probabilità minore, che si troveranno evidentemente al fondo della scala, e cominciamo a costruire l'albero associando il bit 0 a quello più probabile ed il bit 1 a quello meno probabile: abbiamo dunque

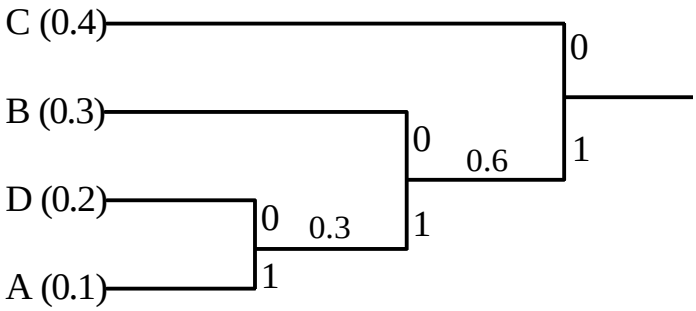


A questo punto, consideriamo questi due simboli come un unico simbolo avente probabilità di trasmissione pari alla somma delle probabilità, che in questo caso 0.3.

Il passo successivo consiste nel ripetere il ragionamento considerando i simboli non ancora esaminati e il simbolo corrispondente agli altri due (con relativa probabilità). Considerando sempre i due simboli con probabilità minore e associando 0 a quello più probabile ed 1 all'altro, abbiamo che



A questo punto rimangono solo due simboli e quindi l'albero può essere completato:



A partire da quest'albero, noi siamo adesso in grado di ottenere il "**codice di Huffman**" della sorgente considerata, leggendo l'albero stesso a partire dalla *radice* (ossia da destra verso sinistra). Il codice che si ottiene è il seguente:

$$C \longrightarrow 0$$
$$B \longrightarrow 10$$
$$D \longrightarrow 110$$
$$A \longrightarrow 111$$

Una osservazione importante che possiamo fare è che, data una coppia di simboli, non è assolutamente obbligatorio associare 0 a quello più probabile e 1 all'altro. E' possibile anche invertire, a patto però di farlo sempre. Il codice che risulta alla fine è chiaramente diverso, ma gode delle stesse caratteristiche.

Andiamo a calcolare il valore di $\overline{N}$ mediante la semplice definizione:

$$\overline{N} = \sum_{i=1}^{N} N_i p_i = \underbrace{3*0.1}_{A} + \underbrace{2*0.3}_{B} + \underbrace{1*0.4}_{C} + \underbrace{3*0.2}_{D} = 1.9$$

Confrontiamo adesso questo valore con quello che si ottiene se, mantenendo le stesse probabilità di emissione, venisse usato il codice ricavato in precedenza: si trattava del codice

$$A \longrightarrow 0$$
$$B \longrightarrow 10$$
$$C \longrightarrow 110$$
$$D \longrightarrow 111$$

(anch'esso univocamente decodificabile) e, facendo i calcoli, si trova un valore del numero medio di bit pari a 2.5.

Abbiamo dunque trovato una conferma (ma non una dimostrazione) del seguente principio fondamentale: *codificando una sorgente discreta senza memoria con il codice di Huffman, si ottiene il minimo valore possibile del numero medio di bit associati a ciascun simbolo.*

Questo comporta un altro importante risultato: supponiamo di avere un certo alfabeto e di ideare, con metodo qualsiasi, un codice per tale alfabeto che risulti univocamente decodificabile; ci andiamo poi a calcolare il valore di $\overline{N}$ relativo a tale codice: se troviamo, secondo criteri opportuni, che questo valore è quello minimo possibile, allora potremo star certi che il codice ideato corrisponde a quello di Huffman, ossia a quello ottenibile con il metodo prima descritto.

NUMERO MEDIO DI BIT ED ENTROPIA DELLA SORGENTE

Abbiamo detto, senza dimostrarlo, che dato un certo alfabeto, la codifica secondo Huffman è quella che presenta sempre il minimo valore di $\overline{N}$. Sorge allora questa domanda: supponiamo di avere un certo alfabeto, di idearne un codice e di calcolarne il valore di $\overline{N}$; come facciamo a stabilire se questo valore sia o meno il più piccolo possibile? Ossia, in altre parole, dato un generico codice, come si fa a calcolare il valore minimo di $\overline{N}$?

Avendo detto che la codifica di Huffman presenta sempre il valore minimo di $\overline{N}$, un modo potrebbe anche quello di trovare in ogni caso tale codifica e di andarsi a calcolare $\overline{N}$. Tuttavia, è chiaro che si tratta di un metodo tutt'altro che agevole, specialmente considerando che il numero N di simboli di cui è costituito l'alfabeto di una sorgente è generalmente molto alto.

Allora, per risolvere il problema, viene in aiuto il seguente risultato (che si può dimostrare facilmente):

$$\boxed{H(X) \le \overline{N} \le H(X) + 1}$$

Esso dice quindi che il numero medio di bit che il codice associa a ciascun simbolo dell'alfabeto non può essere mai inferiore all'entropia della sorgente. In altre parole, ***H(x) è l'estremo inferiore dell'insieme dei possibili valori di*** $\overline{N}$. Da sottolineare il concetto di "estremo inferiore": solo in alcuni casi molto particolari, $\overline{N}$ può raggiungere il valore di H(x), mentre in generale è sempre leggermente maggiore.

Di conseguenza, quando noi dobbiamo codificare una certa sorgente, della quale siano note le probabilità di emissione (sempre nell'ipotesi di indipendenza tra i simboli), ci possiamo calcolare

l'entropia della sorgente, in modo da sapere il valore minimo di $\overline{N}$ al quale noi dobbiamo tendere nell'ideare il codice. Quando poi definiamo effettivamente il codice, magari usando la codifica di Huffman, giungeremo ad un valore di $\overline{N}$ molto prossimo, ma quasi coincidente, con H(x).

Esempio: trasmissione mediante fax-simile

E' il tipico caso in cui la sorgente emette due soli simboli (bianco o nero, B o N), di cui uno (il bianco) normalmente molto più probabile del secondo (il nero). Supponiamo allora che la probabilità di emettere B (bianco) sia p=0.9, per cui quella di emettere N (nero) sarà 1-p=0.1.

Adottiamo la seguente codifica: B=0 ed N=1.

L'informazione media è evidentemente pari ad 1 bit/simbolo:

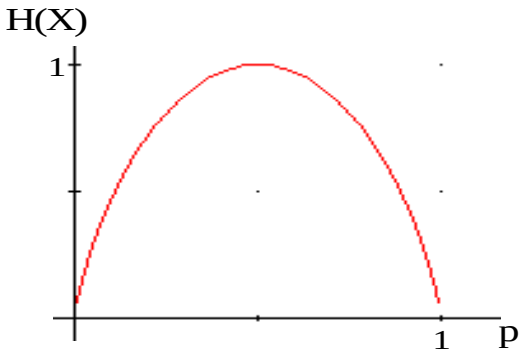
$$I(x) = \overline{N} = \sum_{i=1}^{2} n_i p_i = n_{bianco} p_{bianco} + n_{nero} p_{nero} = 1 \cdot p + 1 \cdot (1-p) = 1$$

L'entropia è invece la seguente:

$$H(X) = \sum_{i=1}^{2} p_i \log_2 \frac{1}{p_i} = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p} = 0.47 \qquad \left[\frac{\text{bit}}{\text{simbolo}}\right]$$

Quindi, nel caso del fax, dove non abbiamo alcuna libertà di scelta sulla codifica (a parte quella di scegliere se B=0 o B=2), l'informazione media è molto maggiore dell'entropia: sprechiamo, in pratica, il 53% dell'informazione trasmessa.

Ricordiamo anche che l'andamento di H(x) per una sorgente con 2 soli simboli è il seguente:



L'entropia viene bassa, come già detto in precedenza, perché siamo lontani dal vertice, che corrisponde a p=0.5, ossia a simboli equiprobabili. Nel caso del fax, invece, un simbolo è molto più probabile dell'altro, per cui H(x)<1.

CODIFICA A BLOCCHI

Abbiamo dunque visto che, nell'ideare un codice con cui codificare l'alfabeto di una certa sorgente, l'obbiettivo primario da raggiungere, oltre la univoca decodificabilità del codice, è quello di ottenere il valore più piccolo possibile per I(x). Abbiamo anche visto che il valore minimo cui si può aspirare corrisponde al valore dell'entropia H(x) della sorgente, valore cui ci si avvicina molto tramite la codifica di Huffman.

Ci poniamo allora il problema seguente: vogliamo ideare un codice, che non sia quello di Huffman, il cui valore di I(x), pur essendo maggiore del valore minimo H(x), sia comunque il più vicino possibile ad H(x) stessa.

Supponiamo, per comodità di ragionamento, che l’alfabeto sorgente sia $\{A, B, C\}$. Di conseguenza, ogni messaggio emesso dalla sorgente sarà una successione di tali simboli. Un esempio potrebbe essere il seguente:

B A C C B A A A C B

Una sequenza di questo tipo può essere pensata in due modi differenti:

- il primo è appunto quello di vederla come una sequenza di simboli singoli emessi dalla sorgente X;
- un altro modo è invece quello di vederla come una sequenza di COPPIE di simboli emessi da una certa sorgente Y.

Possiamo cioè immaginare quel messaggio come prodotto dalla sorgente Y il cui alfabeto sia il seguente:

$$\{(A,A),(A,B),(A,C),(B,A),(B,B),(B,C),(C,A),(C,B),(C,C)\}$$

Si tratta cioè di un alfabeto in cui ogni simbolo corrisponde ad una delle possibili coppie di simboli che si possono formare con l’alfabeto della sorgente X. Chiaramente, dato che X ha 3 simboli, la sorgente Y avrà 3*3=9 simboli.

Naturalmente, dato che noi conosciamo i simboli della sorgente X, conosciamo anche i simboli della sorgente Y, per cui possiamo fare su di essa tutti i discorsi relativi ad una generica sorgente discreta. In particolare, possiamo pensare di effettuare una codifica di Huffman di tale sorgente: indicato con I(y) il numero medio di bit che tale codifica associa a ciascun simbolo di Y e indicata con H(Y) l’entropia della sorgente Y, varrà allora la relazione

$$H(Y) \le I(y) \le H(Y) + 1$$

Vediamo allora quanto vale H(Y): si tratta dell’entropia della sorgente Y , per cui è definita come

$$H(Y) = \sum_{i=1}^{3} \sum_{j=1}^{3} P(x_i, x_j) \log_2 \frac{1}{P(x_i, x_j)}$$

dove $P(x_i, x_j)$ è la probabilità che venga emessa la coppia (x_i, x_j), ossia che la sorgente X emetta prima il simbolo x_i e poi il simbolo x_j.

Poiché siamo nelle ipotesi di indipendenza dei simboli emessi dalla sorgente Y, possiamo scrivere che

$$P(x_i, x_j) = P(x_i) P(x_j)$$

per cui

$$H(Y)=\sum_{i=1}^{3}\sum_{j=1}^{3}P(x_i)P(x_j)\log_2\frac{1}{P(x_i)P(x_j)}=\sum_{i=1}^{3}\sum_{j=1}^{3}P(x_i)P(x_j)\left[\log_2\frac{1}{P(x_i)}+\log_2\frac{1}{P(x_j)}\right]=$$

$$=\sum_{i=1}^{3}\sum_{j=1}^{3}P(x_i)P(x_j)\log_2\frac{1}{P(x_i)}+\sum_{i=1}^{3}\sum_{j=1}^{3}P(x_i)P(x_j)\log_2\frac{1}{P(x_j)}=$$

$$=\sum_{i=1}^{3}P(x_i)\log_2\frac{1}{P(x_i)}\sum_{j=1}^{3}P(x_j)+\sum_{i=1}^{3}P(x_i)\sum_{j=1}^{3}P(x_j)\log_2\frac{1}{P(x_j)}=$$

$$=\sum_{i=1}^{3}P(x_i)\log_2\frac{1}{P(x_i)}+\sum_{j=1}^{3}P(x_j)\log_2\frac{1}{P(x_j)}=H(X)+H(X)=2H(X)$$

Abbiamo dunque trovato che

$$\boxed{H(Y)=2H(X)}$$

per cui possiamo scrivere che

$$2H(X)\le I(y)\le 2H(X)+1$$

o anche che

$$H(X)\le\frac{I(y)}{2}\le H(X)+\frac{1}{2}$$

Adesso, se I(y) è il numero medio di bit che il codice di Huffman associa a ciascun simbolo di Y, I(y)/2 non è altro che I(x), in quanto ogni simbolo di Y corrisponde a 2 simboli di X: abbiamo perciò ottenuto che

$$\boxed{H(X)\le I(x)\le H(X)+\frac{1}{2}}$$

il che significa che abbiamo senz'altro ottenuto un avvicinamento di I(x) al suo valore minimo rispetto a quello che avremmo ottenuto se avessimo codificato secondo Huffman direttamente la sorgente X.

In definitiva, quindi, abbiamo ottenuto che il valore di I(x) migliora (ossia si avvicina a H(X)) se andiamo a codificare secondo Huffman la sorgente Y: questo metodo prende il nome di "**codifica a 2 blocchi**" e la sorgente Y prende il nome di "**sorgente a 2 blocchi**".

E' intuitivo allora comprendere come ulteriori miglioramenti si otterrebbero considerando sorgenti a 3, 4 o più blocchi: si trova infatti in generale che

$$\boxed{H(X)\le I(x)\le H(X)+\frac{1}{n}}$$

Il problema, però, viene dal fatto che, con questo metodo, gli apparati di codifica e decodifica diventano sempre più complessi e quindi sempre più costosi: di conseguenza, ***conviene aumentare la dimensione dei blocchi fino a quando l'incremento di efficienza che si ottiene diventa piccolo rispetto all'incremento della complessità circuitale.***

Esempio: trasmissione mediante fax-simile

La trasmissione mediante fax-simile è un tipico caso in cui la codifica a blocchi risulta particolarmente conveniente.

Abbiamo in precedenza trovato che, prendendo $p_{bianco}=0.1$ e $p_{nero}=0.9$, l'entropia della sorgente risulta essere H(x)=0.47. Codificando B=0 e N=1, come è naturale fare, abbiamo visto che si ottiene I(x)=1: anche se si tratta del minimo valore che noi possiamo ottenere con i "normali" metodi di codifica, è evidente che si tratta di un valore molto distante dal valore dell'entropia 0.47, per cui questo è un tipico caso in è opportuna una codifica a blocchi.

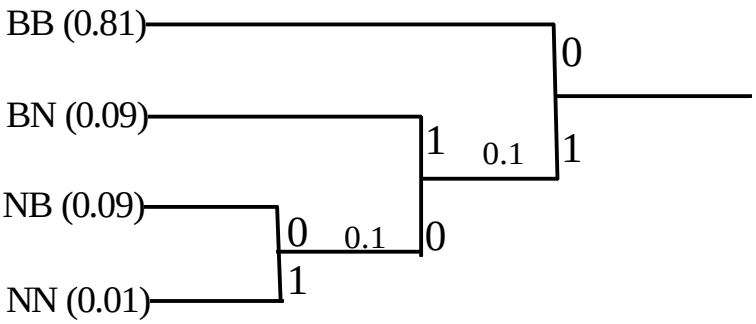
Vediamo perciò se e come migliorano le cose usando una codifica a 2 blocchi: intanto, la sorgente a 2 blocchi sarà evidentemente

$$Y = \{(B,B),(B,N),(N,B),(N,N)\}$$

Le probabilità di emissione sono inoltre le seguenti:

$$p_{BB} = 0.9 * 0.9 = 0.81$$
$$p_{BN} = 0.9 * 0.1 = 0.09$$
$$p_{NB} = 0.09$$
$$p_{NN} = 0.01$$

Andiamo allora a codificare secondo Huffman questa sorgente Y: l'albero risulta essere



per cui il codice è

$$BB \longrightarrow 0$$
$$BN \longrightarrow 11$$
$$NB \longrightarrow 100$$
$$NN \longrightarrow 101$$

Andando allora a calcolare il numero medio di bit che questo codice associa a ciascun simbolo della sorgente Y si trova che I(y)=1.29, da cui

$$I(x) = \frac{I(y)}{2} = 0.645$$

Questo è dunque il numero medio di bit necessari per codificare ogni simbolo (mentre 1.29 è il numero medio di bit necessario per codificare una coppia di simboli). Evidentemente, questo valore è già molto minore del valore 1 che avevamo all'inizio, per cui effettivamente la situazione è migliorata.

PROBLEMI DELLE CODIFICHE A LUNGHEZZA VARIABILE

Le codifiche a lunghezza variabile presentano una serie di problemi. Il primo è stato già citato e riguarda la **decodificabilità**: mentre, nel caso delle codifiche a lunghezza fissa, il ricevitore è sempre in grado di distinguere una parola dall'altra, nel caso di codifiche a lunghezza variabile questo è possibile solo se nessuna parola di codice costituisce prefisso di una parola di codice diversa.

Un altro problema è il fatto che un errore su un bit comporta un errore non solo sulla parola di cui esso fa parte, ma anche sulle parole successive.

Ancora, nel caso delle codifiche a lunghezza variabile il tasso di informazione non è costante nel tempo, mentre il mezzo trasmissivo può accettare informazione solo a tasso costante. Si rende allora necessario un processo di *equalizzazione del tasso di generazione dei bit*: si usa perciò un dispositivo che riceve bit a tasso variabile, ma li emette a tasso fisso, in modo che possano transitare sul mezzo trasmissivo senza problemi.

Ci sono però una serie di problematiche relative al funzionamento di questo dispositivo. Infatti, la perfetta equalizzazione si avrebbe solo se tale dispositivo potesse accumulare tutti i bit da trasmettere, per poi ritrasmetterli alla velocità desiderata. Così facendo, però, avremmo un aumento indefinito del ritardo medio con cui l'informazione giunge al destinatario. E' necessario, allora, fissare un ritardo massimo tollerabile nella trasmissione e dimensionare di conseguenza la memoria del dispositivo. Tale memoria sarà una **memoria a doppia porta**, dove appunto si possa scrivere a velocità variabile e leggere a velocità costante. Il ritardo inevitabile è dovuto chiaramente al fatto che la capacità di questa memoria non può essere infinita: una volta fissato il ritardo massimo tollerabile, viene di conseguenza fissata la capacità di memoria.

Non è ancora finita, perché bisogna fare in modo che il dispositivo funzioni anche in due situazioni critiche, che sono quelle di memoria vuota (non ci sono più bit da trasmettere) e di memoria piena (o overflow, per cui ulteriori bit vengono persi):

- quando la memoria risulta vuota, il problema è quello di tener occupato il canale, inviando informazioni sempre con la stessa velocità; in questo caso, si ricorre al cosiddetto **bit stuffing**: si procede alla trasmissione di bit a caso, privi di informazione, facendo ovviamente in modo da avvisare il ricevitore che si tratta di *bit fittizi*;
- quando, invece, la memoria è piena (**overflow**), gli ulteriori bit ad essa inviati vengono necessariamente persi; allora, per perdere meno bit possibile, si cerca di abbassare, temporaneamente, il tasso di informazione all'ingresso della memoria (il che equivale ovviamente a degradare, sempre momentaneamente, la qualità con cui si descrive l'informazione); è evidente che, anche in questo caso, il ricevitore va informato, altrimenti non sarebbe più in grado di ricostruire le informazioni: infatti, ridurre il tasso di informazione in ingresso alla memoria significa, in pratica, ridurre il numero di bit per ogni campione, ed il ricevitore deve perciò adeguare la ricostruzione del segnale ad questa riduzione (temporanea).

SORGENTI DISCRETE CON MEMORIA

Fino ad ora, abbiamo supposto che la sorgente (discreta) considerata emetta simboli, appartenenti al proprio alfabeto, in modo indipendente uno dall'altro: in altre parole, l'emissione di un simbolo, in un certo istante, ha una probabilità indipendente da quali siano stati i simboli emessi in precedenza. Questa ipotesi non è molto realistica, in quanto, nella realtà, la maggior parte delle sorgenti sono **sorgenti con memoria**, ossia sorgenti in cui l'emissione di ciascun simbolo è comunque condizionata dai simboli emessi in precedenza. Vogliamo studiare questo tipo di sorgenti: vedremo che valgono grossomodo gli stessi concetti visti fino ad ora, con la differenza

principale di una maggiore complicazione matematica, derivante essenzialmente dalla presenza delle probabilità condizionate in luogo di quelle assolute viste fino ad ora.

Possiamo anticipare subito un concetto: nel caso delle sorgenti senza memoria, abbiamo visto che l'entropia H(x) della sorgente rappresenta in pratica il contenuto informativo della sorgente. Nel caso delle sorgenti con memoria, questo non è più vero, fondamentalmente perché *la statistica dipendenza tra i simboli costituisce una informazione aggiuntiva, di cui non teniamo invece conto nel caso dell'entropia come è stata fino ad ora definita.*

Entropia condizionale

Supponiamo dunque che l'alfabeto della nostra sorgente con memoria sia $\{s_1, s_2, \ldots, s_M\}$: questa sorgente, usando i simboli di questo alfabeto, emette dei messaggi che vanno codificati in binario. Ciascuno di questi messaggi, come abbiamo già detto, può essere sicuramente visto come una possibile realizzazione del processo stocastico che rappresenta l'emissione da parte della sorgente. Supponiamo allora di avere i seguenti messaggi generici:

messaggio 1 $\longrightarrow s_3\ s_4\ s_5\ s_1\ s_2\ s_M\ \ldots.$

messaggio 2 $\longrightarrow s_1\ s_2\ s_5\ s_2\ s_M\ s_{M-1}\ \ldots.$

$\ldots\ldots$

messaggio k $\longrightarrow s_M\ s_1\ s_4\ s_2\ s_2\ s_{M-2}\ \ldots.$

Queste realizzazioni sono evidentemente delle funzioni discrete nel tempo (in quanto l'emissione dei simboli avviene non in modo continuo bensì in istanti di tempo discreti) a valori discreti (in quanto l'alfabeto sorgente possiede un numero finito di simboli). Siamo cioè in presenza di un processo tempo-discreto a valori discreti. Per le stesse ipotesi fatte in precedenza, il processo è ergodico, per cui resta ancora valido l'approccio, che è alla base della **teoria di Shannon**, di considerare una sequenza di bit molto lunga per stabilire le prestazioni limite di un sistema.

Possiamo pensare di estrarre dal processo una o più variabili aleatorie: la variabile aleatoria X_n estratta al generico istante t=nT indica perciò il valore assunto dal processo all'istante nT o, ciò che è lo stesso, il valore assunto dal processo al passo n, che corrisponde all'intervallo di tempo $[(n-1)T, nT]$. Indichiamo in particolare con X_0 la variabile aleatoria estratta al passo 0, cioè all'istante t=0, e con X_1 quella estratta al passo 1, ossia all'istante t=T.

Mentre, nel caso di simboli indipendenti, abbiamo considerato le probabilità di emissione dei singoli simboli, adesso la cosa è più complicata, in quanto dobbiamo considerare le probabilità condizionate: in un generico istante di osservazione, dovremo valutare la probabilità di emissione di un determinato simbolo s_i nell'ipotesi che il simbolo precedente sia stato s_j. Indicheremo questa probabilità con il simbolo $P(s_i|s_j)$. Estendendo il discorso ad un intero messaggio x, composto da un numero M di simboli, avremo allora lo stesso risultato trovato nel caso di simboli indipendenti, con la differenza di introdurre in questo caso le probabilità condizionate al posto delle probabilità assolute:

$$H(x \mid s_j) = -\sum_{i=1}^{M} P(s_i|s_j) \log_2 P(s_i|s_j)$$

Questa è dunque l'**entropia del primo ordine** della sorgente con memoria in esame.

Se estendiamo quella formula a tutte le possibili situazioni condizionanti, ossia a tutti i possibili simboli s_j, otteniamo

$$H(x \mid s) = -\sum_{j} P(s_j) \sum_{i=1}^{M} P(s_i \mid s_j) \log_2 P(s_i \mid s_j)$$

Possiamo portare il termine $P(s_j)$ all'interno della seconda sommatoria: così facendo, otteniamo il prodotto $P(s_j)P(s_i \mid s_j)$, che rappresenta notoriamente la probabilità di emettere prima s_j e poi s_i:

$$P(s_j)P(s_i \mid s_j) = P(s_i, s_j) \longrightarrow \boxed{H(x \mid s) = -\sum_{j=1}^{M} \sum_{i=1}^{M} P(s_i, s_j) \log_2 P(s_i \mid s_j)}$$

Quella ottenuta è la cosiddetta **entropia condizionata** della sorgente con memoria.

Facciamo osservare l'analogia esistente tra questa definizione e quella di "entropia" data per le sorgenti senza memoria, che era

$$H(X) = -\sum_{i=1}^{N} P(x_i) \log_2 P(x_i)$$

CODIFICA RUN-LENGTH

E' abbastanza intuitivo comprendere che, tenendo conto della statistica dipendenza tra i simboli emessi dalla sorgente, possiamo ottenere prestazioni notevolmente migliori di quelle che invece otterremmo trascurando tale dipendenza. Un tipico caso è quello della **codifica run-length** (o *codifica a lunghezza di run*), che per esempio trova un'ottima applicazione nella trasmissione mediante fax.

Nel caso del fax, infatti, non è corretto supporre la statistica indipendenza tra i simboli, come abbiamo invece fatto in precedenza: una volta che si sia verificato l'evento NERO, è abbastanza probabile che se ne verifichi un altro, dato che la traccia della scrittura ha un certo spessore. Questo discorso vale ancora maggiormente nel caso del BIANCO: una volta che si sia verificato l'evento BIANCO, è estremamente probabile che se ne verifichi un altro, dato che un foglio di fax ha generalmente più spazio lasciato in BIANCO rispetto allo spazio ricoperto dal NERO, cioè appunto dalla traccia.

Il fatto che il BIANCO sia predominante induce ad adottare uno schema del tipo seguente: quando ci si trova in una zona bianca del fax, anziché trasmettere una lunga sequenza di 0, si effettua prima un conto di quanti 0 andrebbero trasmessi, dopo di che si invia al ricevitore un messaggio (detto appunto **run length**) che lo informi di quanti sono questi simboli 0. E' evidente che in questo modo si ottiene una notevole riduzione del *numero medio di bit per simbolo*, il quale numero scenderà sicuramente al di sotto dell'entropia calcolata nell'ipotesi di simboli equiprobabili ed indipendenti: in pratica, cioè, *si sfrutta la statistica dipendenza tra i simboli in modo da portare il numero medio di bit per simbolo al di sotto dell'entropia del primo ordine* $H(X) = -\sum_{i=1}^{N} P(x_i) \log_2 P(x_i)$, *che invece sembrava essere il limite minimo raggiungibile.* Tale limite è consistente solo per simboli indipendenti.

ENTROPIA DI ORDINE SUPERIORE

Osservando l'emissione della sorgente per un periodo di tempo sufficientemente lungo, siamo dunque in grado di calcolarne l'entropia condizionata, che rappresenta in pratica l'ulteriore contenuto informativo trasmesso dalla sorgente. E' però necessario calcolare anche l'entropia vera della sorgente, ossia le *entropie di ordine superiore*.

La differenza tra l'entropia del primo ordine e una entropia di ordine superiore sta semplicemente nel fatto che, mentre la prima considera le probabilità $P(x_i)$ di emissione del singolo simbolo, la seconda considera le probabilità di una successione di simboli, ossia le probabilità congiunte $P(x_i, x_{i+1}, ...)$. L'entropia del secondo ordine considera perciò le probabilità del tipo $P(x_i, x_{i+1})$, quella di terno ordine le probabilità del tipo $P(x_i, x_{i+1}, x_{i+2})$ e così via per entropie di ordine via via crescente.

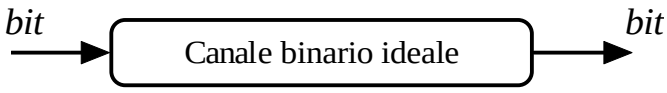
Quello che si osserva è che, ***all'aumentare dell'ordine, cioè del numero di simboli considerati, l'entropia tende ad un valore minimo.*** Senza scendere nei dettagli analitici, è facile spiegarsi questo fatto: oltre una certa distanza temporale, non c'è più dipendenza statistica tra il primo e l'ultimo simbolo considerato. Questo valore minimo dell'entropia di ordine superiore è assunto essere l'**entropia della sorgente**.

Ovviamente, se i simboli emessi dalla sorgente sono indipendenti, allora non ha senso considerare probabilità condizionate, che sarebbero uguali alle probabilità assolute, per cui l'entropia di ordine superiore viene comunque a coincidere con l'entropia del primo ordine. In questo caso, quindi, l'entropia della sorgente si riduce a sua volta all'entropia del primo ordine.

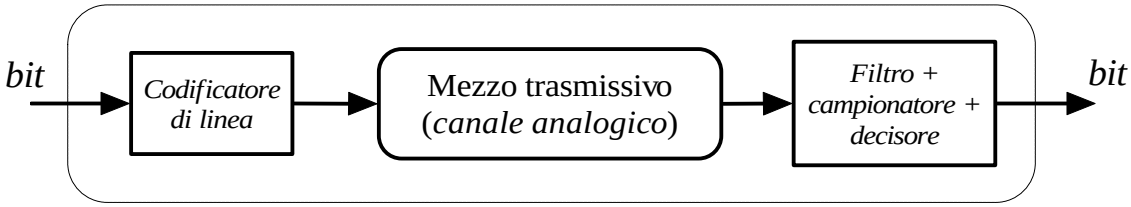
CANALE BINARIO IDEALE E REALE

Lo studio di una sorgente si riduce dunque al calcolo della sua entropia, la quale caratterizza la sorgente dal punto di vista del contenuto informativo emesso. Il passo successivo è quello di utilizzare un **canale** che possa trasmettere questo contenuto informativo al destinatario.

Cominciamo i nostri discorsi da un **canale binario ideale**, cioè privo completamente di rumore e quindi della probabilità di sbagliare:



E' bene osservare da cosa sia costituito, in realtà, questo canale binario ideale: come detto in precedenza, esso rappresenta tutto quell'insieme di dispositivi necessari alla trasmissione dei bit dal trasmettitore al ricevitore; di conseguenza, con riferimento a quanto visto a proposito della trasmissione numerica, esso comprenderà, al minimo, i seguenti dispositivi:



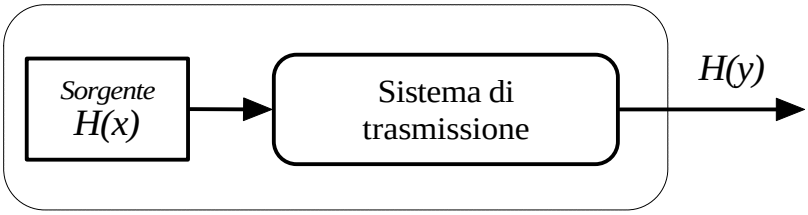
Se il canale analogico ci mette a disposizione una banda B, sappiamo che possiamo raggiungere diverse velocità di **trasmissione a seconda** che usiamo un sistema a 2 livelli o a più livelli:

- per un normale sistema binario (a 2 soli livelli), la frequenza di cifra (o velocità di trasmissione) è semplicemente il doppio della banda disponibile (nell'ipotesi di un progetto alla Nyquist): $f_S = 2B$ bit/sec;
- se il sistema è a più livelli, allora dobbiamo considerare più periodi di cifra successivi, considerare il messaggio composito e contare quanti messaggi di questo tipo si possono ottenere.

In altri termini, si compie una osservazione su un intervallo di tempo molto lungo e si conta quanti possibili messaggi (configurazioni di simboli) si possono verificare in questo intervallo di tempo. Se indichiamo con N(t) il numero di messaggi riscontrati, il numero di bit necessario a descriverli sarà evidentemente $\log_2 N(t)$. Per quantificare il numero medio di bit/sec da trasmettere, non dobbiamo far altro che rendere il tempo di osservazione t quanto più grande possibile e mediare la quantità $\log_2 N(t)$: si definisce perciò **capacità del canale (ideale)** la quantità

$$C_{ideale} = \lim_{t\to\infty} \frac{\log_2 N(t)}{t} \qquad \left[\frac{bit}{sec}\right]$$

Le cose cambiano se il canale è reale, per cui i bit in uscita, a causa del rumore, possono essere in parte sbagliati. Per analizzare questa situazione, consideriamo semplicemente una sorgente, con entropia H(x), che trasmette bit su un **sistema di trasmissione** (che include il codificatore di linea, il mezzo trasmissivo ed il decodificatore di linea):



La sorgente emette un certo messaggio x, il quale deve essere recapitato all'utente tramite il sistema di trasmissione. Quest'ultimo, essendo affetto da rumore, fornisce in uscita un messaggio y che, in generale, sarà diverso da quello trasmesso, a causa appunto degli errori. Ai fini pratici, è come se noi avessimo una sorgente, costituita dall'insieme sorgente vera+sistema di trasmissione, avente una certa entropia H(y). Se il sistema di trasmissione fosse ideale, risulterebbe H(y)=X(x) e non ci sarebbero problemi.

Al contrario, data la non idealità, risulta H(y)≠H(x): questa diversità deriva, quindi, dal fatto che H(y) contiene in parte informazione corretta e in parte informazione non corretta. Questo ha una conseguenza fondamentale: il contenuto informativo di H(y) che a noi interessa non è tutta H(y), ma solo quella parte di H(y) che contiene le informazioni corrette. In altre parole, se vogliamo valutare l'*informazione vera* a disposizione dell'utente, dobbiamo considerare H(y) privata però della cosiddetta **equivocazione**, ossia di quella parte di informazione sbagliata che il canale introduce, di suo, a causa del rumore.

L'informazione vera che emerge dal canale è dunque valutabile nel modo seguente:

$$I(x,y) = H(y) - H(y\,|\,x)$$

Come detto, abbiamo cioè la differenza tra tutto ciò che viene fuori dal canale, H(y), e tutto ciò che c'è di sbagliato, ossia l'equivocazione H(y|x). Questa equivocazione, in base alla simbologia H(y|x), sembra dipendere sia dal canale sia dall'entropia della sorgente. In effetti, come verificheremo poi negli esempi, *è intuitivo aspettarsi che l'equivocazione non*

dipenda dall'entropia della sorgente, cioè dalla statistica della sorgente, in quanto sappiamo che il rumore è un processo indipendente da ciò che transita sul canale.

Esiste anche un altro modo di valutare l'informazione vera I(x,y): infatti, quando noi trasmettiamo un certo messaggio x sul canale, proprio perché sappiamo che il canale non è ideale ci aspettiamo che x non arrivi così com'è all'utente, ma affetto da un certo numero di errori; abbiamo cioè una *incertezza* su ciò che l'utente riceve a fronte di ciò che gli abbiamo trasmesso. In termini matematici, possiamo allora scrivere che

$$I(x,y) = H(x) - H(x \mid y)$$

dove $H(x|y) \neq H(y|x)$.

Questa relazione è del tutto equivalente alla precedente, ma possiamo fare due osservazioni: la prima è che essa mette meglio in evidenza il fatto che I(x,y) dipenda sia dal canale sia dalla sorgente; la seconda è che, invece, questa espressione non è comoda ai fini dei conti. Lo capiremo meglio con l'esempio che seguirà.

A questo punto, per un canale reale, la **capacità** si definisce nel modo seguente:

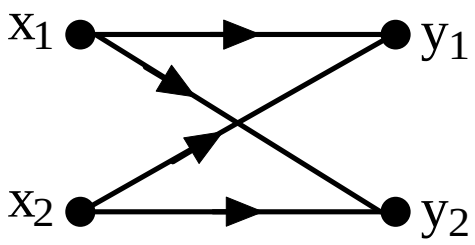
$$C = \max_{x} I(x,y)$$

Questa definizione dice quanto segue: considerando tutte le possibili sorgenti di informazione (cioè considerando tutte le possibili statistiche di emissione dell'informazione), *la capacità del canale considerato è pari alla massima quantità di informazione vera che il canale stesso riesce a trasmettere.* In altre parole, misurando, per tutte le infinite sorgenti, la quantità di informazione vera che il canale trasmette, la capacità è pari al massimo valore riscontrato.

Questa definizione, come sarà chiaro più avanti, mostra in pratica che, dato un canale reale con una data capacità C, il massimo delle prestazioni del canale si otterrà solo con quella sorgente tale che C=I(x|y), ossia solo con quella che sorgente che consente al canale di trasmettere una informazione vera pari proprio alla sua capacità.

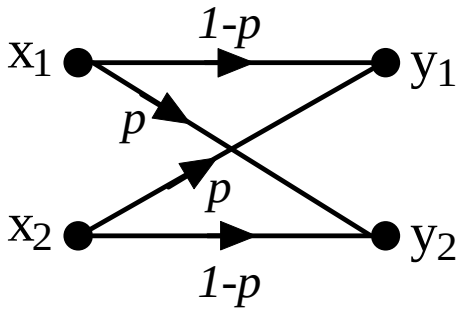
Esempio: canale di trasmissione binario

Facciamo subito un esempio pratico di applicazione dei concetti appena esposti. Consideriamo un canale di trasmissione binario e una sorgente che invia su di esso i simboli x_1 e x_2 con probabilità rispettivamente α e $1-\alpha$. A fronte di questi simboli, il canale fornisce in uscita i simboli y_1 ed y_2, secondo uno schema del tipo seguente:



Ci mettiamo subito sotto due ipotesi semplificative: il rumore sovrapposto al segnale utile è di tipo gaussiano ed il decisore ha la soglia a metà. Sotto queste ipotesi, la probabilità di errore è la stessa quale che sia il simbolo trasmesso: $p = P(\varepsilon \mid 1T) = P(\varepsilon \mid 0T)$. Abbiamo cioè la stessa probabilità

p di confondere x_1 con x_2 oppure x_2 con x_1. Possiamo allora indicare le probabilità di errore sullo stesso schema di prima:



La probabilità di non commettere errori è ovviamente 1-p in entrambi i casi.

Vogliamo dunque calcolare la capacità di un canale con queste caratteristiche: applicando la definizione, abbiamo che

$$C = \max_x I(x,y) = \max_x \big(H(y) - H(y \mid x)\big) = \max_x \big(H(x) - H(x \mid y)\big)$$

Abbiamo la possibilità di scegliere se usare, per calcolare I(x,y), l'espressione $\big(H(x) - H(x \mid y)\big)$ oppure l'espressione $\big(H(y) - H(y \mid x)\big)$. Per motivi che saranno chiari tra poco, scegliamo quest'ultima espressione, per cui dobbiamo calcolare H(y) e H(y|x).

Applichiamo anche qui le rispettive definizioni:

$$H(y) = -\sum_{i=1}^{2} P(y_i) \log_2 P(y_i)$$

$$H(y \mid x) = -\sum_{j=1}^{2} \sum_{i=1}^{2} P(y_i, x_j) \log_2 P(y_i \mid x_j)$$

Cominciamo da H(y): esplicitando i due termini della sommatoria, abbiamo che

$$H(y) = -P(y_1) \log_2 P(y_1) - P(y_2) \log_2 P(y_2)$$

Il simbolo y_1 si ottiene in uscita sia quando è stato trasmesso x_1 e non c'è stato errore sia quando è stato trasmesso x_2 ma c'è stato un errore; analogo discorso per la ricezione di y_2. Possiamo perciò scrivere che

$$P(y_1) = P(x_1)\big(1 - P(\varepsilon \mid x_1)\big) + P(x_2) P(\varepsilon \mid x_2) = \alpha(1-p) + (1-\alpha)p = \alpha + p - 2\alpha p$$
$$P(y_2) = P(x_2)\big(1 - P(\varepsilon \mid x_2)\big) + P(x_1) P(\varepsilon \mid x_1) = (1-\alpha)(1-p) + \alpha p = 1 - p - \alpha + 2\alpha p$$

Sostituendo nell'espressione di H(y), otteniamo dunque che

$$H(y) = -(1 - p - \alpha + 2\alpha p)\log_2(1 - p - \alpha + 2\alpha p) - (\alpha + p - 2\alpha p)\log_2(\alpha + p - 2\alpha p)$$

Come potevamo aspettarci, questa quantità dipende sia dalla statistica della sorgente, tramite il parametro α, sia dalle caratteristiche del canale, tramite il parametro p.

Passiamo adesso al calcolo di H(y|x), che si conduce in maniera analoga, ma con la difficoltà di dover considerare probabilità congiunte e probabilità condizionate:

$$H(y\,|\,x) = -\sum_{j=1}^{2}\sum_{i=1}^{2} P(y_i, x_j)\log_2 P(y_i\,|\,x_j) =$$

$$= -P(y_1, x_1)\log_2 P(y_1\,|\,x_1) - P(y_2, x_1)\log_2 P(y_2\,|\,x_1) - P(y_1, x_2)\log_2 P(y_1\,|\,x_2) - P(y_2, x_2)\log_2 P(y_2\,|\,x_2)$$

Possiamo allora servirci di una tabella, nella quale indichiamo le 4 possibili coppie x_1y_1, x_1y_2, x_2y_1, x_2y_2 e le corrispondenti probabilità congiunte e condizionate:

coppie	$P(x_i, y_i)$	$P(x_i\,|\,y_i)$
x_1y_1	$\alpha(1-p)$	$1-p$
x_1y_2	αp	p
x_2y_1	$p(1-\alpha)$	p
x_2y_2	$(1-\alpha)(1-p)$	$1-p$

Abbiamo dunque a disposizione gli 8 termini da sostituire nell'espressione di H(y|x):

$$H(y\,|\,x) = -\alpha(1-p)\log_2(1-p) - \alpha p\log_2 p - p(1-\alpha)\log_2 p - (1-\alpha)(1-p)\log_2(1-p) =$$
$$= -\left[\alpha(1-p) + (1-\alpha)(1-p)\right]\log_2(1-p) - \left[\alpha p + p(1-\alpha)\right]\log_2 p = -p\log_2 p - (1-p)\log_2(1-p)$$

Osservando il risultato ottenuto, deduciamo che, come anticipato in precedenza, ***H(y|x) non dipende dalla statistica della sorgente ma solo dalle caratteristiche del canale.*** Questa proprietà non è generale, ma dipende dal fatto che abbiamo considerato la soglia a metà, per cui le probabilità condizionate $P(y_1|x_2)$ e $P(y_2|x_1)$, cioè le probabilità di errore, sono le stesse.

Possiamo dunque scrivere l'espressione completa di I(x,y):

$$I(x,y) = \left[-(1-p-\alpha+2\alpha p)\log_2(1-p-\alpha+2\alpha p) - (\alpha+p-2\alpha p)\log_2(\alpha+p-2\alpha p)\right] +$$
$$-\left[-p\log_2 p - (1-p)\log_2(1-p)\right]$$

A questo punto, dato che $C = \max_x I(x,y)$, dobbiamo massimizzare la quantità I(x,y) rispetto alla statistica della sorgente, ossia dobbiamo derivare I(x,y) rispetto ad α: è evidente, allora, che, in questo calcolo, il termine H(y|x), essendo indipendente da α, non interviene:

$$\frac{dI(x,y)}{d\alpha} = \frac{dH(y)}{d\alpha} = \frac{d}{d\alpha}\left[-(1-p-\alpha+2\alpha p)\log_2(1-p-\alpha+2\alpha p) - (\alpha+p-2\alpha p)\log_2(\alpha+p-2\alpha p)\right]$$

Questo è il motivo per cui, all'inizio, abbiamo considerato l'espressione $I(x,y) = H(y) - H(y\,|\,x)$ e non l'altra espressione.

Dovremmo adesso calcolare quella derivata rispetto ad α e uguagliarla a 0, in modo da ricavare il corrispondente valore di α. Possiamo anche procedere in modo più semplice: infatti H(y) è comunque l'entropia di una sorgente binaria ed abbiamo visto che la sua espressione analitica, se β è la probabilità di emissione di uno dei due simboli, è

$$H(y) = \sum_{i=1}^{2} p_i \log_2 \frac{1}{p_i} = \beta\log_2\frac{1}{\beta} + (1-\beta)\log_2\frac{1}{1-\beta}$$

Abbiamo anche diagrammato l'andamento di questa quantità in funzione di β, verificando che il valore massimo si ottiene quando i simboli sono equiprobabili, ossia quando β=1/2. Di conseguenza, possiamo applicare questo risultato al nostro caso: possiamo infatti scrivere che

$$H(y) = -(1-p-\alpha+2\alpha p)\log_2(1-p-\alpha+2\alpha p) - (\alpha+p-2\alpha p)\log_2(\alpha+p-2\alpha p) = \beta\log_2\frac{1}{\beta} + (1-\beta)\log_2\frac{1}{1-\beta}$$

da cui deduciamo che

$$\beta = -(1-p-\alpha+2\alpha p)$$
$$1-\beta = -(\alpha+p-2\alpha p)$$

Imponendo allora β=1-β=0.5, possiamo usare una qualsiasi delle due equazioni per ricavare il valore di α: si trova α=0.5, ossia ancora una volta simboli equiprobabili da parte della sorgente con entropia H(x). Abbiamo cioè ottenuto che *la massima quantità di informazione (vera) che un canale binario simmetrico può far pervenire si ha quando la sorgente che lo alimenta è a simboli equiprobabili.*

Quindi, α=0.5 è il valore che massimizza la quantità I(x,y). A questo punto, fissato α, il valore di C viene a dipendere dal valore di p, ossia dalla probabilità di errore del canale. Si verifica facilmente che, se p=1/2, ossia se il canale sbaglia mediamente 1 bit su 2, risulta C=0. Se invece p tende a 0 (nessun errore) oppure ad 1 (errore sempre), allora risulta C=1.

Dire che p=0 e quindi C=1 equivale a dire che il canale, ricevendo i bit della sorgente a velocità f_S bit/sec, fa arrivare all'utente questi stessi bit senza errori, per cui la capacità, intesa questa volta come numero di bit corretti al secondo, è pari ad f_S. Se, invece C<1, allora il numero di bit corretti al secondo è minore di f_S.

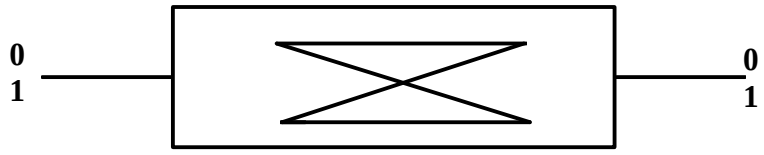
In definitiva, *la* ***capacità del canale espressa in bit/sec*** *rappresenta la massima velocità di trasmissione dei bit sul canale, teoricamente ottenibile senza commettere errori.*

Tanto per fare un esempio concreto, supponiamo che la sorgente emetta bit (equiprobabili) a frequenza di cifra $f_S = 10\ \text{kbit/sec}$; se il canale, supposto binario simmetrico, fosse ideale (p=0), i bit corretti arriverebbero all'utente alla velocità di 10 kbit/sec; se invece il canale è reale (p≠0), allora i bit corretti arrivano a velocità inferiore a 10 kbit/sec. La velocità con cui arrivano i bit corretti è appunto la capacità del canale (espressa in bit/sec): dire che il canale ha una capacità di 9kbit/sec significa dire che questa è la velocità con cui l'utente riceverà i bit corretti.

Codici per rilevazione e correzione d'errore

INTRODUZIONE

Consideriamo un semplice canale binario discreto, che possiamo schematizzare nel modo seguente:



Essendo un canale reale, esso ha una certa probabilità di sbagliare, il che significa che le **transizioni incrociate**, indicate nella figura, hanno una probabilità non nulla di verificarsi. In particolare, se il canale è simmetrico, sappiamo che esse hanno uguale probabilità di verificarsi, in quanto il canale sbaglia un simbolo o l'altro con la stessa probabilità.

Il nostro scopo, data appunto la presenza degli errori, è quello di capire come possiamo impiegare una parte dei bit che inviamo sul canale per proteggere l'informazione che trasmettiamo sul canale stesso.

Per capire questo concetto, ci serviamo di un esempio concreto. Consideriamo un segnale analogico (ad esempio quello televisivo) sottoposto a campionamento e poi a quantizzazione; supponiamo che ad ogni campione vengano associati 8 bit: ciò significa che abbiamo a disposizione, per la codifica di ogni campione, $2^8=256$ configurazioni binarie. Supponiamo infine di usare tutte queste 256 configurazioni binarie. Questo ha una implicazione fondamentale: nel momento in cui si verificano uno o più errori sulla generica parola di 8 bit, otteniamo in ricezione una nuova parola di 8 bit che fa ancora parte delle parole consentite: questo comporta che il ricevitore non abbia modo di capire che la parola ricevuta sia sbagliata, cioè sia diversa da quella trasmessa.

Si capisce, quindi, che, se vogliamo mettere in grado il ricevitore quanto meno di riconoscere la presenza di errori, non possiamo utilizzare tutte le configurazioni binarie lecite, ma solo alcune di esse: sotto questa ipotesi e sotto opportuni altri vincoli che via via esamineremo, noi facciamo in pratica in modo che, a seguito della presenza di uno o più errori, la generica parola binaria ricevuta dal ricevitore non appartenga all'insieme di quelle lecite, per cui il ricevitore capisce che ci sono stati errori.

D'altra parte, se è vero che il ricevitore si accorge della presenza di errori, dato che riceve una parola non lecita, è anche vero che non ha modo di capire su quali bit si sono verificati gli errori.

Nei paragrafi seguenti vedremo i principali metodi di codifica necessari alla rilevazione dell'errore, da parte del ricevitore, ed eventualmente alla correzione dell'errore stesso.

USO DI UN BIT DI PARITÀ

Supponiamo di avere un codice formato da parole di 8 bit, come per esempio accade nella trasmissione numerica del segnale telefonico o di quello televisivo (in entrambi i casi vengono infatti associati 8 bit ad ogni campione in uscita dal campionatore). Si introduce un ulteriore bit, detto **bit di parità**, in modo che ogni parola da 9 bit rispetti un vincolo ben preciso: il numero di bit posti ad 1 nella parola deve essere pari (**parità pari**) oppure dispari (**parità dispari**).

Il ricevitore, ricevendo la generica parola da 9 bit e sapendo che tipo di parità è stata imposta, verifica se tale parità è rispettata o meno: in caso affermativo, conclude che la parola è valida,

mentre in caso negativo rileva la presenza dell'errore nella trasmissione. Ci sono ovviamente due controindicazioni a questo modo di procedere: la prima è che, nel caso di un numero pari di errori, la parità non viene comunque violata, per cui il ricevitore prende per buona una parola che in realtà è sbagliata; in secondo luogo, la parità viene violata sia in presenza di un errore singolo sia in presenza di 3 errori sia di 5 errori e così via, ossia in presenza di un numero dispari di errori.

A fronte di queste controindicazioni, c'è però una considerazione di fondo da fare: si può dimostrare che, se la probabilità p(ε) di sbagliare un bit è abbastanza piccola, la probabilità di sbagliare più di un bit è ancora più piccola, tanto da poter essere generalmente trascurata. Ciò significa che l'errore singolo è l'evento decisamente più probabile, per cui è vero che, in presenza di un numero pari di errori, il ricevitore sbaglia, ma è anche vero che questo (cioè appunto la presenza di un numero pari di errori) è un evento altamente improbabile.

In base a quest'ultima considerazione, deduciamo che, *nell'ideare* ***codici di protezione dagli errori****, ha senso concentrarsi solo sulla protezione dall'errore singolo, che è quello che ha maggiore probabilità di verificarsi.*

Torniamo dunque all'aggiunta di un singolo bit di parità: per quanto visto, si tratta di un codice che consente solo la rilevazione di un numero dispari di errori, e quindi anche dell'errore singolo, ma non consente in alcun modo la correzione, in quanto non indica la posizione degli errori. Il motivo per cui tale codice consente la rilevazione dell'errore singolo può essere spiegato intuitivamente nel modo seguente: con 9 bit, abbiamo 2^9=512 possibili configurazioni binarie, ma, dato che il bit di parità non porta informazioni, le configurazioni utilizzate continuano ad essere 256; queste configurazioni, in base al criterio con cui viene fissato il bit di parità, godono di una proprietà fondamentale: presa una qualsiasi coppia di tale parole, esse differiscono per almeno 2 bit (si tratta cioè di un codice con **distanza di Hamming** pari a 2). Questo è il motivo per cui è rilevabile l'errore singolo: data la generica parola trasmessa, in presenza di 1 solo errore il ricevitore riceve una parola che sicuramente non fa parte di quelle lecite.

Una volta rilevato l'errore, il ricevitore non ha modo di correggerlo, per cui ha due possibilità: la prima, decisamente poco praticabile, è quella di fornire all'utente la parola effettivamente ricevuta, "avvertendolo" però che è sbagliata; la seconda, più realistica, è quella di chiedere la **ritrasmissione** (**procedura ARQ**) esso invia cioè un segnale al trasmettitore chiedendo che ritrasmetta la stessa parola: se sulla nuova parola non si verifica alcun errore, allora esso riesce a fornire all'utente la parola corretta, altrimenti richiede una ulteriore ritrasmissione e così via finche la parola ricevuta non risulta corretta.

E' ovvio che un meccanismo di questo tipo presenta almeno 3 problemi:

- la prima riguarda una maggiore complessità del ricevitore, il quale dovrà possedere l'hardware necessario ad effettuare la richiesta di ritrasmissione;
- in secondo luogo, tale richiesta di ritrasmissione dovrà viaggiare, verso il trasmettitore, attraverso un canale (**canale di servizio**) distinto da quello su cui viaggiano le informazioni; il mezzo trasmissivo dovrà cioè essere di tipo bidirezionale e non è detto che questo si possa realizzare o che sia conveniente farlo;
- infine, lo svantaggio principale è nella diminuzione della velocità di trasmissione (bit/sec): se non usassimo il bit di parità, non avremmo modo di rilevare errori per cui trasmetteremmo alla massima velocità possibile f_S, che è quella con cui la sorgente in trasmissione emette i bit; aggiungendo il bit di parità, nell'ipotesi di non rilevare errori (e quindi di non effettuare ritrasmissioni), la velocità di trasmissione sarebbe 8/9 di quella massima f_S, in quanto ogni 9 bit trasmessi, solo 8 contengono l'informazione; infine, se, rilevando l'errore singolo, chiediamo la ritrasmissione, la velocità di trasmissione scende al di sotto degli 8/9 di f_S e scende tanto più quante più ritrasmissioni siamo costretti a richiedere. Si può anche arrivare al punto che le prestazioni subiscano un tale crollo da portare il sistema al di sotto dei requisiti minimi di funzionamento.

Questi inconvenienti rendono spesso poco praticabile il sistema di mettere in piedi il **canale di ritorno** per la ritrasmissione. Si ricorre, perciò, a strategie diverse, ben più sofisticate, per la rilevazione e la correzione dell’errore singolo: si usano cioè codici particolari.

BIT DI PARITÀ SULLE RIGHE E SULLE COLONNE

Un primo possibile codice per la rilevazione e la correzione dell’errore è una semplice estensione del concetto del bit di parità. Anziché considerare una singola parola da 8 bit e aggiungervi il bit di parità, si considerano più parole e si aggiungono più bit di parità. Per esempio, supponiamo di considerare 3 parole da 8 bit per volta; date queste 3 parole, le disponiamo in uno schema del tipo seguente:

									X
									X
									X
bit di parità ⟶	X	X	X	X	X	X	X	X	\ \

Si impilano le 3 parole una sull’altra, formando 8 colonne da 3 bit ciascuna. Successivamente, si impostano i bit di parità sia sulle colonne sia sulle righe (le caselle riempite con le X sono quelle destinate appunto ai bit di parità). Una volta ottenuto tutti i bit di parità, si trasmette l’insieme dei bit partendo dal primo in alto sinistra.

In ricezione, questo meccanismo consente di rilevare e correggere l’errore singolo: infatti, un eventuale errore singolo viola la parità sia sulla riga sia sulla colonna, per cui il ricevitore è in grado di individuare immediatamente la sua posizione.

E’ ovvio che, anche in questo caso, non c’è prevenzione dagli errori multipli: ad esempio, se su una stessa riga si verificano 2 errori, la parità sulla riga viene conservata, mentre ci saranno due colonne con parità violata: in questo caso, il ricevitore non saprà su quale delle 3 righe effettuare la correzione. In modo analogo, se si verificano 2 errori sulla stessa colonna, abbiamo due righe con parità violata, me nessuna colonna con parità violata, per cui anche in questo caso non c’è possibilità di effettuare la correzione.

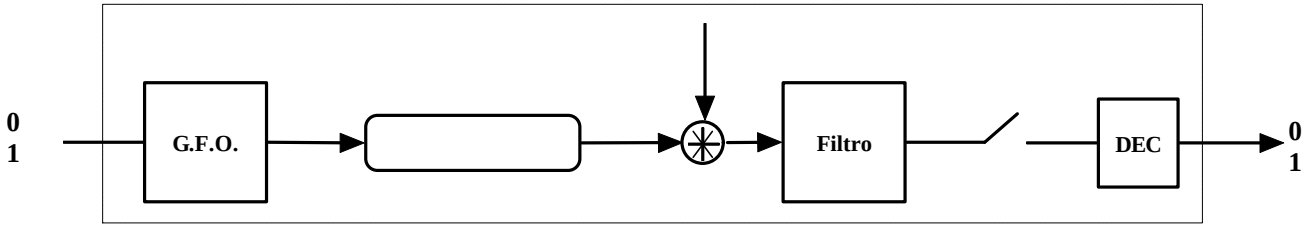
CODICI DI HAMMING

Codici molto potenti per la correzione dell’errore singolo sono i cosiddetti codici di Hamming, i quali sfruttano un preciso risultato matematico, in base al quale se vogliamo rilevare e correggere d errori sulle parole di un determinato codice, è necessario che tali parole abbiano una distanza minima di Hamming pari a 2d+1. Sulla descrizione di tali codici non ci dilunghiamo (vedere appunti di Elettronica Digitale).

SOFT DECISION E HARD DECISION

Abbiamo detto più volte, nei paragrafi precedenti, che l’introduzione di un unico bit di parità sulle parole di codice consente solo la rilevazione di un numero dispari di errori, mentre non consente la correzione. In realtà non è proprio così, in quanto, una volta rilevato l’errore singolo, c’è una accettabile probabilità di effettuare la correzione, a patto però di porsi in un contesto completamente diverso da quello considerato fino ad ora.

Se consideriamo un sistema di trasmissione numerico nel suo complesso, otteniamo una struttura schematica del tipo seguente:



Nei precedenti paragrafi, abbiamo sempre scelto di guardare il sistema ai morsetti esterni, avendo perciò a che fare solo con i bit: in questo caso, l'unico modo sensato di effettuare la correzione e la rilevazione dell'errore singolo è quello di ricorrere ai codici di Hamming, ossia, in generale, di ricorrere al concetto di distanza di Hamming tra la parola binaria ricevuta e quella, tra quelle lecite, che più si avvicina adesso.

Concentrandoci solo sui bit, noi ci disinteressiamo in pratica di quello che avviene in sede di ricezione, vale a dire quello che succede in corrispondenza del campionatore: aspettiamo che il campionatore effettui le sue misure e che il decisore effettui le sue decisioni, per poi valutare se i bit così ottenuti soddisfano alla regola della parità. Possiamo allora pensare di sfruttare prima il fatto che i bit di ogni parola devono soddisfare alla regola della parità; il metodo è il seguente: in corrispondenza di ogni parola binaria, il campionatore ci fornisce 8 campioni, più quello relativo alla parità; se non ci fosse rumore, gli 8 campioni non sono affetti da errori, per cui la successiva decisione darà sicuramente la parola corretta; in presenza di errori, invece, gli 8 campioni presentano valori (reali e non più discreti) diversi da quelli attesi teoricamente (cioè appunto in assenza di rumore); lo scostamento di ciascun campione dal valore teorico è dato proprio dall'entità del rumore sovrapposto. Allora, se è stato rilevato un errore, è logico aspettarsi che il campione che ha dato l'errore sia quello che si trova più vicino alla soglia, in quanto è quello che, verosimilmente, ha subito il maggiore picco di rumore.

Si può fare anche qualcosa in più, ragionando direttamente sull'insieme di 8 campioni: infatti, si può pensare di confrontare tale insieme con tutti gli altri insiemi che si potrebbero ottenere in assenza di rumore (a questo scopo, il ricevitore dovrà possedere una memoria di opportune dimensioni); l'insieme, tra quelli teorici, che più si avvicina (in termini di distanza euclidea e non più di distanza di Hamming) a quello misurato sarà quello che, con maggiore probabilità, è stato realmente trasmesso.

Questo discorso, però, lascerebbe pensare che sia a questo punto indifferente l'aggiunta del bit di parità: si può comunque pensare di effettua una decisione multipla invece di decisioni singole successive. In realtà, il bit di parità dà un aiuto notevole, in quanto esso, non portando con se alcuna informazione, fa in modo che la distanza tra tutte le possibili configurazioni sia raddoppiata rispetto al caso in cui non si introducesse alcun bit di parità. Questo è un sicuro aiuto quando bisogna optare per l'uno o l'altro insieme di campioni.

Quindi, passando dalla distanza di Hamming alla distanza euclidea, si può comunque ridurre la probabilità di errore anche con un solo bit di parità.

Questo meccanismo prende il nome di **SOFT DECISION** e si contrappone alla **HARD DECISION**, nella quale si esegue prima la singola decisione sui bit e poi si considera il gruppo di bit secondo il concetto della distanza di Hamming.

Osservazione

Possiamo visualizzare un aspetto importante relativo all'aggiunta di un bit di parità.

Consideriamo sempre parole di codice da 8 bit: se non usiamo il bit di parità, noi trasmettiamo le nostre informazioni elementari (cioè le singole parole binarie) su 8 periodi di cifra; se invece, aggiungiamo il bit di parità, che non porta informazione, in pratica "diluiamo" le informazioni elementari su 9 periodi di cifra. Questo fatto, anche in assenza di richiesta di ritrasmissione, comporta una diminuzione della frequenza di cifra, che diventa 8/9 di quella ottenibile in assenza di bit di parità. Possiamo allora pensare di aumentare la velocità, in modo da riportarci alla velocità iniziale, pur conservando il bit di parità. Il problema è che l'aumento di velocità ci può essere di danno: infatti, un aumento della velocità (cioè della frequenza di cifra) equivale ad un aumento della banda occupata dal segnale e questo richiede quindi un allargamento del filtro di ricezione, il quale sarà perciò obbligato ad ingurgitare più rumore; la presenza di maggiore di rumore determina la presenza di più errori, per cui andremmo a perdere ciò che volevamo guadagnare con il bit di parità.

Una alternativa possibile per mantenere invariata la banda ma recuperare ugualmente la velocità di trasmissione iniziale è quello di ricorrere ad un sistema multilivello: in un sistema multilivello, infatti, quello che noi facciamo è scegliere un numero maggiore di 2 di forme d'onda elementari da trasmettere, in modo da associare a ciascuna di esse più di 1 bit. Ad esempio, per 4 forme d'onda elementari, potremo associare 2 bit a ciascuna di esse. La velocità con cui trasmettiamo le forme d'onda è il **baud rate**, che diventa quindi un sottomultiplo della frequenza di cifra:

$$f_B = \frac{f_S}{b} = \frac{f_S}{\log_2 M}$$

dove M sono i livelli e b i bit associati a ciascuna forma d'onda.

Con un sistema di questo tipo, noi possiamo trasmettere forme d'onda ad una velocità (f_B) più bassa, pur mantenendo una frequenza di cifra più elevata, semplicemente aumentando M, ossia considerando più forme d'onda elementari. Quello di trasmettere a velocità più bassa è per noi un vantaggio, proprio perché occupiamo meno banda e quindi ingurgitiamo, in ricezione, meno rumore. Volendo vedere la cosa, anziché nel dominio della frequenza, nel dominio del tempo, il discorso è semplice: man mano che allunghiamo la durata della singola forma d'onda, noi diluiamo in pratica la decisione sul singolo simbolo trasmesso (dove per "simbolo" intendiamo in questo caso la configurazione binaria associata alla forma d'onda considerata) e questo ci consente di ridurre maggiormente l'influenza del rumore: quest'ultimo è infatti un processo a media nulla, per cui il risultato della sua media temporale si approssima tanto più a 0 quanto più è lungo l'intervallo di integrazione.

Teoricamente, quindi, un modo di procedere per minimizzare la probabilità di errore è quello di usare forme d'onda elementari in numero M molto grande, ma di durata quanto più estesa possibile: così facendo, il rumore crea il minimo fastidio, mentre il tasso di informazione si mantiene comunque diverso da 0.

In quest'ottica rientra parzialmente anche il caso dell'aggiunta del bit di parità: infatti, col bit di parità, non allunghiamo le forme d'onda elementari, ma sostanzialmente stabiliamo un legame temporale più lungo tra i bit, aggiungendo una forma d'onda che non è rappresentativa di informazione. In quest'ottica, si può comprendere ancora meglio l'utilità della tecnica della soft decision.

Codici convoluzionali

INTRODUZIONE

Facciamo una osservazione a proposito dei codici di Hamming: sappiamo che, al crescere del numero N di bit di informazione che compongono la generica parola di codice, è necessario aumentare il numero k di bit di parità da introdurre per ottenere la correzione dell'errore singolo: deve infatti essere verificata la relazione

$$2^k \geq N - k - 1$$

Se si vogliono delle prestazioni abbastanza accettabili in termini di correzione dell'errore singolo ma si vuole anche una percentuale di incremento dei bit più bassa, bisogna aumentare il rapporto k/N, come in effetti è indicato da quella formula: si verifica, infatti, che, all'aumentare di N, il corrispondente aumento di k è minore, per cui il rapporto k/N diminuisce all'aumentare di N.

In altre parole, *un uso ottimale dei codici di Hamming è quello basato su blocchi di lunghezza N abbastanza lunghi.* C'è, però, un problema: all'aumentare delle dimensioni N del generico blocco, comincia a diventare non più trascurabile la probabilità dell'evento "errore doppio", cioè la probabilità di ottenere 2 errori nello stesso blocco. Allora bisognerebbe passare a codici a consentano la correzione di 2 errori, cosa che i codici di Hamming non possono fare. Tutt'al più, con tali codici si può rilevare l'errore doppio, inserendo un ulteriore bit di parità oltre quelli necessari all'individuazione dell'errore singolo.

In definitiva, tirando le somme di quanto detto fino ad ora, appaiono evidenti due cose:

- in primo luogo, che l'elemento che consente di correggere gli errori è la memoria che inseriamo all'interno del sistema, ossia la dipendenza tra i vari bit;
- in secondo luogo, che questo modo di agire è tanto più efficiente, dal punto di vista della correzione degli errori, quanto più lunga è la suddetta memoria: infatti, quanto più lunga è la memoria, tanto diverso è il comportamento tra il codice, che ha una sua regolarità ben precisa, e gli errori ossia il rumore, che invece ha un comportamento del tutto casuale. Questa stessa considerazione è stata fatta quando abbiamo citato la convenienza di usare, in trasmissione, forme d'onda più lunghe ma in numero elevato: si riesce a media via il rumore in modo sempre più efficace.

L'aumento della lunghezza dei blocchi ha però delle controindicazioni:

- sicuramente, quanto più grande è il blocco, tanto più complicato dovrà essere l'hardware in grado di identificare ed eventualmente correggere l'errore;
- problema ancora maggiore è relativo al **ritardo di trasmissione**, che cresce al crescere della grandezza del blocco: all'aumentare della grandezza N del blocco, il momento in cui è possibile effettuare una decisione su tale blocco viene sempre più allontanato nel tempo. Ci sono delle applicazioni in cui un ritardo di trasmissione elevato può risultare inaccettabile.

A questo punto, ci chiediamo se è possibile realizzare un sistema di codifica che abbia memoria grande, per garantire accettabili prestazioni in termini di correzione, ma al contempo abbia anche un ritardo di trasmissione contenuto.

La risposta a questa domanda risiede nei cosiddetti **codici convoluzionali**, che sono una generalizzazione dei codici a blocchi.

PRINCIPI FONDAMENTALI

Supponiamo di considerare parole di codice di lunghezza N molto lunga, il che significa che è altrettanto lunga la memoria del sistema (intesa come memoria sia del codificatore sia del decodificatore). Per implementare questa memoria con una *codifica a blocchi*, dobbiamo procedere nel modo seguente: prima dobbiamo riempire tutta la memoria, inserendovi gli N bit del generico blocco in esame; successivamente, dobbiamo calcolare i k bit di parità; infine, dobbiamo comporre il blocco di N+k bit da inviare al codificatore di linea per la trasmissione.

In realtà, possiamo anche pensare di fare qualcosa di diverso: anziché calcolare i bit di parità per ogni blocco di N bit, ossia quindi ogni N periodi di cifra, possiamo farlo ogni M periodi di cifra, con M<N; ovviamente, però, i bit di parità calcolati ogni M periodi di cifra non dovranno dipendere solo dagli ultimi M bit arrivati, in quanto, in questo caso, non avremmo altro che un codice a blocchi di lunghezza M; al contrario, i suddetti bit di parità devono dipendere da un certo numero di blocchi da M bit che hanno preceduto gli ultimi M bit arrivati. Il numero M di blocchi che sceglieremo dipenderà dalle specifiche e dovrà evidentemente essere tale da non farci ricadere nei problemi di complessità e ritardo di propagazione.

Usiamo un esempio molto semplice: consideriamo nuovamente la relazione

$$2^k \geq N - k - 1$$

che lega il numero k di bit di parità al numero N di bit di informazione. Per N=1, quella relazione è soddisfatta da k=1, il che significa che possiamo usare 1 bit di parità per ogni bit di informazione. Vediamo quali prestazioni ci dà questo codice:

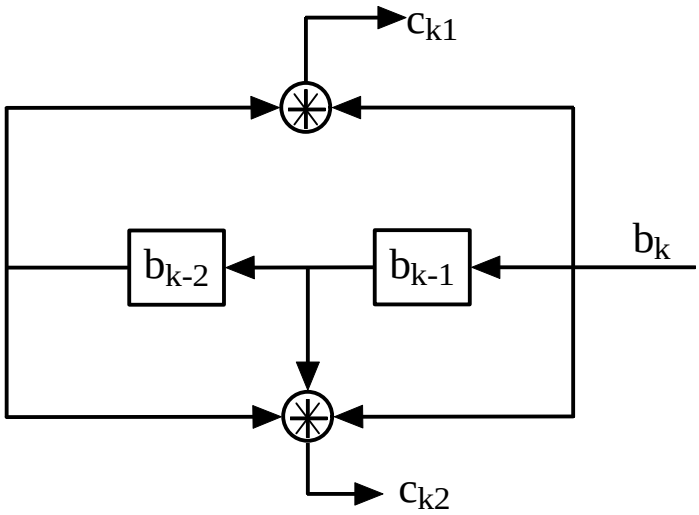
- in primo luogo, esso ha una resa del 50%, in quanto trasmettiamo 2 bit per ogni bit di informazione, per un rapporto quindi pari a 1/2;
- in secondo luogo, imponendo una parità pari, il codice è semplicemente tale che ad 1 associa 11 e a 0 associa 00; allora, in caso di errore singolo, il codice fornisce comunque parole a distanza 1 dalle parole lecite: da 11 si può arrivare a 01 o 10, così come da 00, per cui non c'è alcuna possibilità di correzione.

Quindi, un codice di questo tipo non serve assolutamente a niente. Si può invece pensare di modificarlo trasformandolo in un **codice convoluzionale**: in questo caso, noi continuiamo ad associare 2 bit ad ogni bit di informazione, ma il modo con cui associamo tali 2 bit è radicalmente diverso, in quanto si tiene conto non solo del bit da trasmettere nell'istante considerato, ma anche di un certo numero di bit trasmessi in precedenza.

Si tratta di scegliere su quanti bit passati vogliamo basare la codifica, ossia si tratta di stabilire la memoria del sistema. Un caso semplice è quello in cui usiamo una memoria lunga 2, ossia codifichiamo ciascun bit, nel modo che vedremo adesso, sulla base degli ultimi 2 bit di informazione[4] che abbiamo trasmesso.

[4] E' importante sottolineare, come sarà chiaro dallo schema a blocchi, che, nell'effettuare la codifica, noi teniamo conto degli ultimi 2 bit di informazione che ci sono pervenuti ed abbiamo codificato, mentre non teniamo conto di come abbiamo codificato tali bit. La **memoria** del sistema, quindi, è sempre riferita ai bit di informazione e noi ai bit derivanti dalla codifica.

Lo schema a blocchi del **codificatore convoluzionale con memoria degli ultimi 2 bit trasmessi** è il seguente:

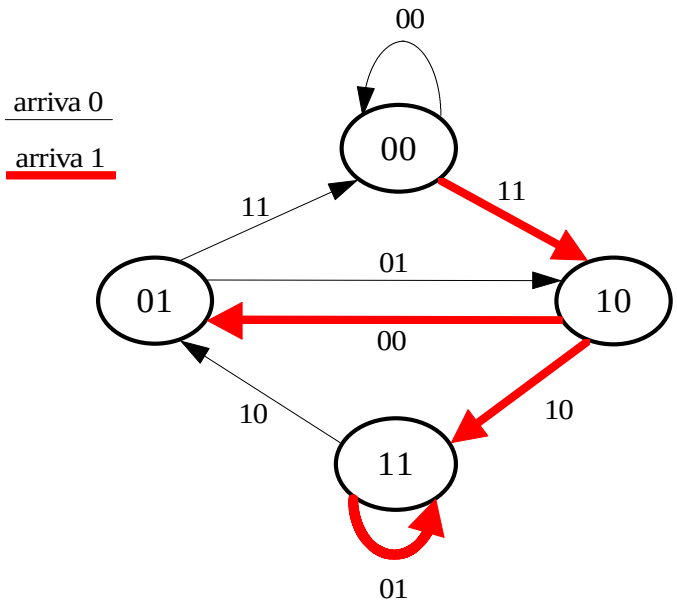


I due "nodi sommatori" che compaiono in questo schema sono, in pratica, due porte logiche EXOR, una a 2 ingressi (quella che fornisce il bit c_{k1}) e l'altra a 3 ingressi (quella che fornisce il bit c_{k2}):

$$c_{k1} = b_{k-2} \oplus b_k$$
$$c_{k2} = b_{k-2} \oplus b_{k-1} \oplus b_k$$

Ricordiamo che la funzione EXOR di un certo numero di ingressi è tale da fornire 1 in uscita se, negli ingressi, compare un numero dispari di 1, altrimenti fornisce 0. In pratica, quindi, la funzione EXOR fa si che la parola formata dai bit di ingresso e dall'uscita della porta stessa contenga sempre un numero pari di 1. In altre parole, essa garantisce la parità pari tra gli ingressi e la sua uscita o, ciò che è lo stesso, effettua una prova di parità sugli ingressi, dando 1 in uscita in caso di esito negativo.

Supponendo di partire da una condizione iniziale in cui i due registri del codificatore sono vuoti ($b_{k-1}=b_{k-2}=0$), è immediato costruire il diagramma degli stati che descrive l'evoluzione dello stato del sistema in funzione dello stato presente e del bit b_k ricevuto in ingresso:

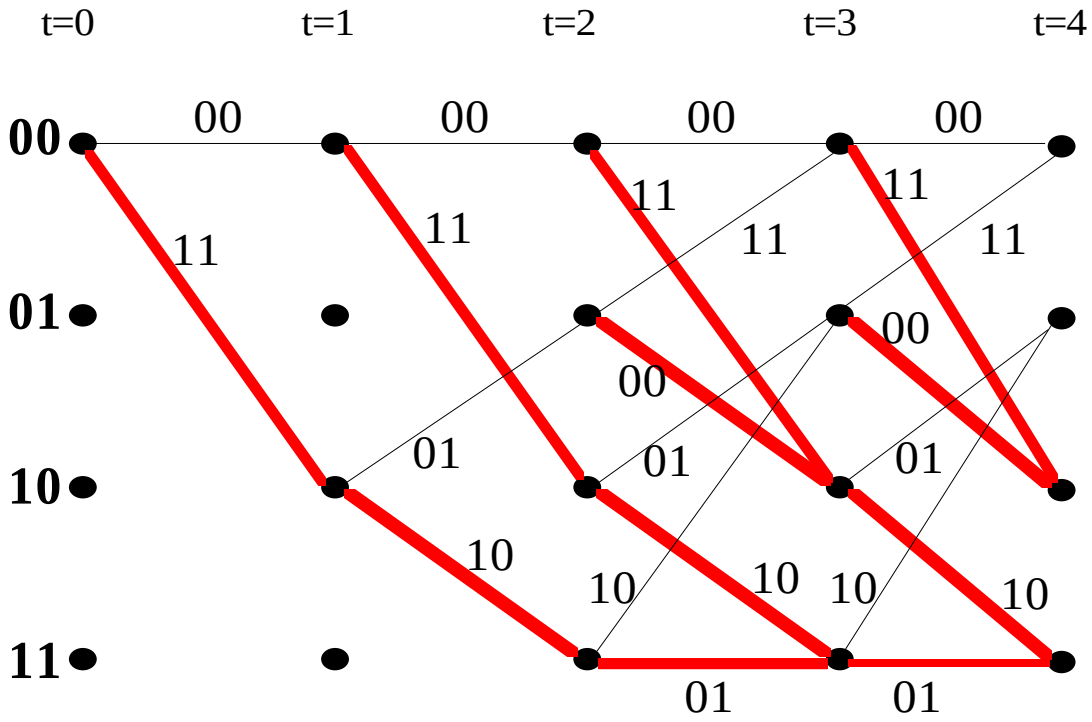


I questo diagramma, lo stato della macchina è rappresentato dal contenuto dei due registri, mentre i bit indicati sugli archi corrispondono ai bit che si deve trasmettere.

Osservando il diagramma, si notano immediatamente due cose:

- la prima è che esistono delle transizioni di stato che non sono possibili: ad esempio, il passaggio dallo stato 00 allo stato 11, oppure il passaggio dallo stato 10 allo stato 01 ed altre. Queste transizioni non permesse garantiscono, in ricezione, che ci si possa rendere conto della presenza di errori[5]; se non ci fossero state queste **transizioni non possibili**, il grafo sarebbe stato connesso, ossia sarebbe stato possibile raggiungere un qualsiasi stato del grafo partendo da un qualsiasi altro: in questo caso, non ci sarebbe stata alcuna possibilità di rilevare gli errori;
- in secondo luogo, confrontando il comportamento di questo codice rispetto a quello considerato prima (2 bit per ogni bit di informazione), si ottiene che la codifica non è più rigida: essendoci 4 archi diversi corrispondenti all’arrivo di un 1 in ingresso, è evidente che possiamo codificare l’1 in 4 modi diversi. Questo fatto dipende proprio dall’aver vincolato la codifica del generico bit di informazione alla memoria degli ultimi 2 bit di informazione che sono arrivati.

Il grafo di prima non consente di evidenziare l’evoluzione temporale della macchina, cosa che invece è evidenziata dal seguente diagramma, dove invece i passaggi di stato sono indicati con riferimento agli istanti t=0 (istante iniziale, in cui lo stato è, per ipotesi, 00), t=1, t=2 e t=3:



Si nota che, essendo prefissato lo stato iniziale, all’istante t=1 ci sono solo due possibilità, cioè due possibili stati di arrivo, in funzione che il bit di informazione fosse 0 (nel qual caso si rimane in 00 e si trasmette 00) oppure 1 (nel qual caso si va in 10 e si trasmette 11). All’istante t=2 le possibilità diventano invece 4, per cui solo da questo istante in poi la macchina è in condizioni di regime, ossia può passare, in ogni istante, per uno qualsiasi dei suoi 4 stati.

[5] Come si vedrà, le transizioni non possibili consentono, in ricezione, la rilevazione dell’errore, mentre la correzione degli errori è una operazione molto più complessa da ottenere.

A questo punto, avendo capito come funziona la fase di codifica, dobbiamo occuparci della decodifica da effettuare in ricezione. Il problema fondamentale è che la decodifica richiede l'uso delle decisioni passate: mentre in trasmissione si parla di un processo di **convoluzione**, in ricezione si parla di un processo di **deconvoluzione**.

In generale, noi potremmo effettuare la decisione in ogni istante, ma abbiamo visto che non ci conviene, in quanto è opportuno effettuare la decisione su blocchi via via più lunghi, in modo da ridurre l'influenza del rumore. Allora, possiamo procedere in modo diverso.

Osserviamo intanto quanto segue: *esaminando le coppie di bit che ci sono giunte durante un certo tempo di osservazione, ci accorgiamo che non abbiamo esaurito tutte le possibilità, ma solo una parte di esse.* Per esempio, consideriamo l'istantet=3 e supponiamo che il codificatore si trovi nello stato 00: affinché il codificatore potesse partire da 00 e giungere, dopo 3 istanti, ancora in 00, le strade possibili sono solo 2: o è rimasto perennemente nello stato 00, oppure da 00 è andato in 10 all'istante t=1, poi in 01 all'istante t=2 ed infine in 00 all'istante t=0. Nel primo caso, la sequenza trasmessa sarebbe stata 00-00-00, mentre nel secondo sarebbe stata 11-01-11. In ricezione, allora, possiamo evidentemente dire questo: se i 6 bit ricevuti sono diversi da 000000 e da 110111, allora sicuramente c'è stato un errore durante la trasmissione.

Questo per quanto riguarda la rilevazione dell'errore. Se vogliamo correggere gli errori, invece, ci basta fare il seguente discorso: data la configurazione binaria ricevuta e sapendo che lo stato di partenza del codificatore era 00, andiamo a valutare qual è la configurazione binaria che differisce, da quella ricevuta, per il minor numero di errori; questa configurazione è quella cosiddetta **a costo minimo**, ossia quella che con maggiore probabilità è stata realmente trasmessa.

Ovviamente, l'ideale sarebbe poter fare questo confronto dopo un tempo di osservazione infinito, ma non ce lo possiamo certo permettere, in quanto ricadremmo nuovamente nel problema del ritardo di trasmissione. Non abbiamo allora altra scelta che effettuare il confronto dopo un certo numero finito di istanti.

Con un criterio di questo tipo, noi basiamo il confronto tra la parola ricevuta e quella teorica nuovamente sul concetto di distanza di Hamming: andiamo cioè a confrontare la parola ricevuta con tutte quelle che, teoricamente, avremmo potuto ricevere in assenza di rumore e poi scegliamo, tra queste, quella che differisce da quella ricevuta per il minor numero di bit. Calcoliamo perciò il **costo** di tale parola sulla base della differenza di bit. In realtà, possiamo anche perfezionare ulteriormente il discorso, ricorrendo nuovamente alla **SOFT DECISION**: anziché memorizzare la configurazione binaria, dobbiamo memorizzare i valori dei campioni analogici campionati in presenza di rumore e poi dobbiamo confrontarli con quelli teorici ottenibili in assenza di rumore; la configurazione di campioni teorici che meno si distanzia, in termini questa volta di **distanza euclidea**, dalla configurazione di campioni realmente ottenuti è quella che, con maggiore probabilità, è stata trasmessa.

Ricordiamo, a tal proposito, il modo con cui calcolare la distanza euclidea tra due distinte configurazioni di campioni: sia v_1, v_2, v_3, v_4 il vettore di campioni realmente misurati dal campionatore in presenza di rumore e sia a_1, a_2, a_3, a_4 uno dei possibili vettori di campioni misurati in assenza di rumore; la distanza euclidea tra i due vettori è

$$d_{v-a} = (v_1 - a_1)^2 + (v_2 - a_2)^2 + (v_3 - a_3)^2 + (v_4 - a_4)^2$$

Ricavando la distanza tra il vettore v_1, v_2, v_3, v_4 e tutti i possibili vettori a_1, a_2, a_3, a_4 che ci aspetteremmo in assenza di rumore, si sceglierà il vettore a_1, a_2, a_3, a_4 cui corrisponde la distanza minima.

Ovviamente, anziché fare un confronto reale tra v_1, v_2, v_3, v_4 e tutti i possibili vettori a_1, a_2, a_3, a_4, si potrebbe anche pensare di minimizzare d_{v-a} rispetto al vettore incognito a_1, a_2, a_3, a_4, del quale quindi si ottiene il valore.

Autore: **SANDRO PETRIZZELLI**
e-mail: sandry@iol.it
sito personale: http://users.iol.it/sandry
succursale: http://digilander.iol.it/sandry1