17 Teoria dell'Informazione e Codifica

Qui sono raccolti alcuni argomenti relativi alla codifica di sorgente e di canale, assieme alle basi teoriche che individuano le prestazioni-limite che possono essere conseguite da un sistema di trasmissione dell'informazione. In questa prima stesura, viene affrontato il tema della codifica di sorgente, ossia delle tecniche utilizzate per trasmettere un messaggio informativo sfruttando al meglio la *capacità di canale* offerta dal sistema di trasmissione disponibile.

17.1 Codifica di sorgente

Una sorgente di informazione può essere per sua natura di tipo discreto, come nel caso di un documento di tipo testo, o continua, come nel caso di un segnale tempo continuo, ad esempio di natura multimediale come audio e video. In base a considerazioni di tipo statistico, la sorgente può essere caratterizzata da un parametro, l'*Entropia*, che indica il tasso di informazione (in bit/secondo) intrinseco per i messaggi prodotti dalla sorgente; d'altra parte, la rappresentazione nativa dei messaggi effettivamente prodotti dalla sorgente può determinare una velocità di tramissione ben superiore all'Entropia.

Lo scopo della *Codifica di Sorgente* è quello di individuare rappresentazioni alternative per i messaggi della sorgente, in modo da ridurre la quantità di bit/secondo necessari alla trasmissione, a valori il più possibile prossimi a quelli indicati dall'Entropia, sfruttando le particolarità della sorgente, del processo di codifica, e del destinatario dei messaggi, come

- la particolare distribuzione statistica dei simboli o dei valori emessi dalla sorgente, tale da permettere l'uso di meno bit per rappresentare i simboli *più frequenti* di altri;
- la dipendenza statistica presente tra simboli successivi, ovvero la presenza di un fenomeno di memoria intrinseco della sorgente, tale da rendere possibile entro certi limiti *la predizione* dei valori futuri;
- l'introduzione di un *ritardo di codifica* che permette di analizzare un intero intervallo temporale del messaggio;
- nel caso di segnali multimediali, l'esistenza di *fenomeni percettivi* legati alla fisiologia dell'apparato sensoriale, tali da guidare il codificatore nella scelta delle componenti di segnale da sopprimere, in quanto percettivamente non rilevanti.

Nel caso di sorgenti nativamente discrete, come ad esempio per documenti in formato elettronico, lo scopo della codifica di sorgente è quello di permettere la ricostruzione *integrale* di quanto trasmesso, e dunque in questo caso viene detta *senza perdita di informazione*. Nel caso di sorgenti continue invece, si ottiene una sequenza numerica a

seguito di un processo di campionamento e quantizzazione, che determina l'insorgenza di una prima causa di *distorsione* nel messaggio ricostruito, a cui spesso si aggiungono altre cause legate allo sfruttamento dei fenomeni percettivi: in tal caso la codifica di sorgente viene quindi detta *con perdita di informazione*.

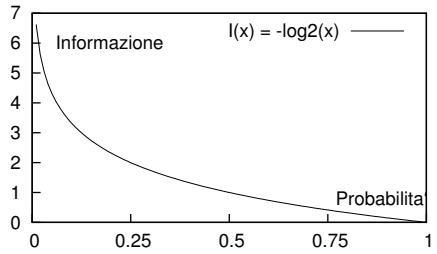
17.1.1 Codifica di sorgente discreta

Sorgente senza memoria Prendiamo in considerazione una sorgente discreta e stazionaria, che emetta una sequenza $x(n)$ composta di simboli x_k appartenenti ad un alfabeto di cardinalità L (ossia con $k = \{1, 2, \cdots, L\}$), ognuno contraddistinto dalla probabilità di emissione $p_k = Pr(x_k)$. Il termine *senza memoria* si riferisce al fatto che, se indichiamo con x_h, x_k una coppia di simboli emessi uno dopo l'altro (prima x_h, e poi x_k), la probabilità del simbolo emesso per secondo non dipende dall'identità di quello(i) emesso precedentemente, ossia $p(x_k/x_h) = p(x_k) = p_k$.

Misura dell'informazione Definiamo informazione associata all'osservazione del simbolo x_k il valore[1]

$$I_k = I(x_k) = \log_2 \frac{1}{p_k} = -\log_2 p_k \text{ bit}$$

che rappresenta il grado di incertezza a riguardo del verificarsi un evento, prima che questo si verifichi, ovvero di quanto possiamo ritenerci sorpresi nel venire a conoscenza di evento, di cui ritenevamo di conoscere la probabilità. Osserviamo infatti che per come è fatta la funzione logaritmo, a bassi valori di probabilità è associata una informazione elevata. La scelta di usare il logaritmo in base 2 conduce ai seguenti risultati:



Prob. p_k	Informazione $-\log_2 p_k$	Commento
1	0	L'evento certo non fornisce informazione
0	∞	L'evento impossibile dà informazione infinita
$\frac{1}{2}$	1	In caso di scelta binaria (es. testa o croce) occorre una cifra binaria (*bit = binary digit*) per indicare il risultato

17.1.1.1 Entropia

Come in termodinamica al concetto di entropia si associa il grado di *disordine* in un sistema, così per una sorgente l'entropia misura il livello di *casualità* dei simboli emessi. Definiamo quindi *Entropia* (indicata con H) di una sorgente discreta S, il valore atteso della quantità di informazione apportata dalla conoscenza dei simboli da essa generati

$$H_S = E\{I_k\} = \sum_{k=1}^{L} p_k I_k = \sum_{k=1}^{L} p_k \log_2 \frac{1}{p_k} \text{ bit/simbolo} \tag{17.1}$$

[1] Per calcolare il logaritmo in base 2, si ricordi che $\log_2 \alpha = \frac{\log_{10} \alpha}{\log_{10} 2} \simeq 3.32 \log_{10} \alpha$.

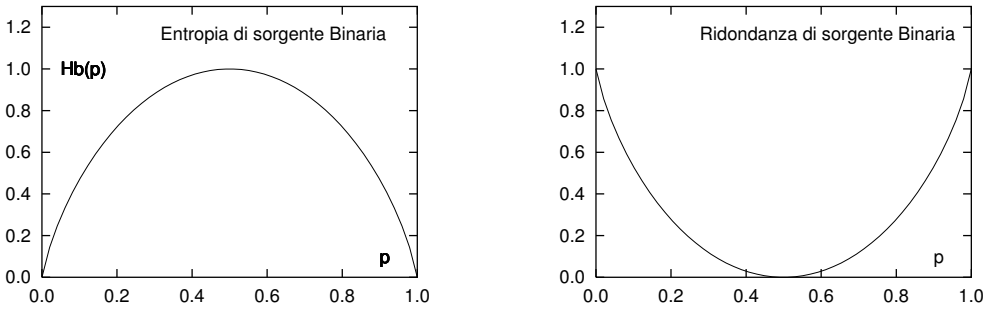


Figura 17.1: Entropia di sorgente binaria, e ridondanza associata

che pesando in probabilità il valore di informazione associato ai diversi simboli, rappresenta il tasso medio di informazione per simbolo delle sequenze osservabili. Osserviamo ora che:

- Se i simboli sono *equiprobabili* ($p_k = \frac{1}{L}$ con $\forall k$), la sorgente è *massimamente informativa*, e la sua entropia è la massima possibile per un alfabeto ad L simboli, e pari a $H_{S_{Max}} = \frac{1}{L}\sum_{k=1}^{L} \log_2 L = \log_2 L$ bit/simbolo.
- Se i simboli non sono equiprobabili, allora $H_S < \log_2 L$.
- Se la sorgente emette sempre e solo lo stesso simbolo, allora $H_S = 0$. Questa circostanza, assieme alle precedenti, consente di scrivere che $0 \leq H_S \leq \log_2 L$.

Entropia di sorgente binaria Un caso particolare è quello delle sorgenti binarie, che producono uno tra due simboli $\{x_0, x_1\}$ con probabilità rispettivamente $p_0 = p$, $p_1 = q = 1 - p$, che inserite nella formula dell'Entropia, forniscono l'espressione

$$H_b(p) = -\left(p \log_2 p + (1-p)\log_2(1-p)\right) \text{ bit/simbolo} \tag{17.2}$$

il cui andamento è mostrato nella figura 17.1, in funzione di p.

I due simboli $\{x_0, x_1\}$ possono essere rappresentati dalle 2 cifre binarie $\{0, 1\}$, che in questo caso chiamiamo *binit* (binary digit), per non confonderli con la misura dell'informazione (il bit). Osserviamo quindi che se se $p \neq .5$, risulta che $H_b(p) < 1$, ossia la sorgente emette informazione con un tasso inferore a un bit/simbolo, mentre a prima vista non potremmo usare meno di un bit per rappresentare ogni simbolo, introducendo una ridondanza pari a $1 - H_b(p)$ (graficata)[2].

Esempio Consideriamo il caso di una sorgente con $p_0 = 0.8$ e $p_1 = 0.2$. L'applicazione della (17.2) fornisce un valore $H_b(0.8) = .8\log_2 \frac{1}{.8} + .2\log_2 \frac{1}{.2} = 0.72$ bit/simbolo, minore del valore di 1 bit/simbolo che si sarebbe ottenuto nel caso di equiprobabilità.

[2]Si presti attenzione sulla differenza: la ridondanza della codifica di sorgente, indica i binit/simbolo sprecati, mentre la ridondanza della codifica di canale, indica il rapporto tra binit di protezione e quelli di informazione.

Entropia di sorgente L-aria Lo stesso concetto ora esposto si applica ad una sorgente che emette simboli appartenenti ad un alfabeto di L simboli, non equiprobabili, a cui compete una Entropia $H_L < \log_2 L$ bit/simbolo: se ne codifichiamo i simboli utilizzando $\lceil \log_2 L \rceil$ binit/simbolo[3] otteniamo una ridondanza pari a $\lceil \log_2 L \rceil - H_L$.

Esempio Consideriamo il caso di una sorgente quaternaria con $p_0 = 0.5$, $p_1 = 0.25$, $p_2 = 0.125$, $p_3 = 0.125$. L'applicazione della (17.1) fornisce $H_4 = 1.75$ bit/simbolo, inferiore ai 2 bit/simbolo che si sarebbero ottenuti nel caso di simboli equiprobabili.

Entropia di sorgente con memoria Rimuoviamo ora l'ipotesi di indipendenza statistica tra i simboli emessi. In questo caso indichiamo con $\mathbf{x} = \{x(1), x(2), \ldots, x(N)\}$ una sequenza di N di simboli, la cui probabilità congiunta risulta essere

$$p(\mathbf{x}) = p(x_1)\, p(x_2/x_1)\, p(x_3/x_1, x_2) \ldots p(x_N/x_1, x_2, \ldots x_{N-1}) \neq \prod_{k=1}^{N} p(x_k)$$

dato che appunto la dipendenza statistica comporta l'uso delle probabilità condizionali. In questo caso, l'espressione dell'entropia si modifica in

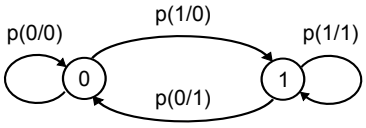
$$H_N = E_{\mathbf{x}}\{I(\mathbf{x})\} = -\frac{1}{N} \sum \sum \sum_{tutti\ gli\ \mathbf{x}} \ldots \sum p(\mathbf{x}) \log_2 p(\mathbf{x}) \text{ bit/simbolo}$$

in modo da eseguire la media statistica su tutte le possibili sequenze $\mathbf{x}$ di lunghezza N. H_N è indicata come *entropia a blocchi*, e si dimostra che al crescere di N il suo valore è non crescente, ossia $H_{N+1} \leq H_N \leq H_{N-1}$,mentre per $N \to \infty$, H_Ne tende ad un valore $H_\infty \leq H_s$, in cui l'uguaglianza è valida solo per sorgenti senza memoria.

Sorgente Markoviana Se oltre ad un certo valore $N = \bar{N}$ la sequenza H_N non decresce più, allora la sorgente è detta a *memoria finita* o *di Markov*, caratterizzata dal fatto che le probabilità condizionate dipendono solo dagli ultimi $\bar{N}$ simboli emessi.

Esempio Analizziamo il caso di una sorgente binaria di Markov del primo ordine, per la quale sono definite le probabilità

$$\begin{aligned} p(0/0) &= 0.9 \qquad p(1/0) = 0.1 \\ p(0/1) &= 0.4 \qquad p(1/1) = 0.6 \end{aligned}$$



ed a cui corrisponde il diagramma di transizione mostrato. In questo caso, l'ultimo simbolo emesso determina *lo stato* della sorgente, condizionando così i valori delle probabilità di emissione di un nuovo simbolo: con i valori dell'esempio, si osserva come la sorgente *preferisca* continuare ad emettere l'ultimo simbolo prodotto, piuttosto che l'altro.

Sotto un certo punto di vista, è come se la sorgente binaria si fosse *sdoppiata*, esibendo due diverse statistiche in base allo stato in cui si trova. Perciò, in questo caso l'entropia di sorgente può essere calcolata applicando la (17.2) ad ognuno dei due stati, ottenendo dei valori di *Entropia condizionata*, che sono poi mediati statisticamente, pesandoli con le probabilità di trovarsi in ognuno degli stati del modello Markoviano. Tornando all'esempio, i valori di entropia condizionata risultano pari a

$$\begin{aligned} H(x/0) &= -0.9 \log_2 0.9 - 0.1 \log_2 0.1 = 0.47 \\ H(x/1) &= -0.4 \log_2 0.4 - 0.6 \log_2 0.6 = 0.97 \end{aligned}$$

[3] La notazione $\lceil \alpha \rceil$ indica l'intero superiore ad α: ad esempio, se $\alpha = 3.7538$, si ha $\lceil \alpha \rceil = 4$.

bit/simbolo, mentre il valore della probabiltà di trovarsi in uno dei due stati si ottiene risolvendo il sistema

$$\begin{cases} p(0) & = p(0/0)\,p(0) + p(0/1)\,p(1) \\ 1 & = p(0) + p(1) \end{cases}$$

in cui la prima equazione asserisce che la probabilità di trovarsi in S_0 è pari alla somma di quella di esserci già, per quella di emettere ancora zero, più la probabilità di aver emesso uno, ed ora emettere zero. Sostituendo i valori, si ottiene $p(0) = 0.8$ e $p(0) = 0.2$, ossia gli stessi valori dell'esempio binario senza memoria. Ma mentre in quel caso il valore dell'entropia risultava pari a 0.72 bit/simbolo, ora si ottiene

$$H = p(0)\,H(x/0) + p(1)\,H(x/1) = 0.58 \text{ bit/simbolo}$$

mostrando come la presenza di memoria aumenti la predicibilità delle sequenze emesse dalla sorgente.

Esercizio Si ripeta il calcolo dell'entropia per un modello di Markov del primo ordine, caratterizzato dalle probabilità $p(x_1) = p(x_2) = 0.5$ e $p(x_1/x_2) = p(x_2/x_1) = 0.01$, mostrando che in questo caso si ottiene una entropia di 0.08 bit/simbolo.

Ci chiediamo ora: è possibile trasmettere tanti binit quanti ne servono a rappresentare il flusso informativo della sorgente, e non di più? La risposta è sí, ricorrendo alle tecniche denominate di CODIFICA DI SORGENTE, di cui appresso forniamo degli specifici casi applicativi.

17.1.1.2 Codifica entropica, a lunghezza di parola variabile

Un *trucco* molto efficiente, è quello di usare CODEWORDS (le *parole di codice* con cui rappresentiamo i simboli di sorgente) caratterizzate da un numero *variabile* di bit, ed usare le più lunghe per descrivere i simboli meno probabili. Consideriamo ad esempio una sorgente con alfabeto di cardinalità $L = 4$, ai cui simboli compete la probabilità riportata alla seconda colonna della tabella che segue.

In questo caso l'Entropia vale

$$\begin{aligned} H = \sum_k p_k \log_2 \frac{1}{p_k} &= \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + \frac{2}{8}\log_2 8 \\ &= \frac{1}{2} + \frac{1}{2} + \frac{2}{8}3 = 1.75 \text{ bit/simbolo} \end{aligned}$$

Simbolo	Prob.	Codeword	L
x_1	.5	0	1
x_2	.25	10	2
x_3	.125	110	3
x_4	.125	111	3

Se adottiamo un codificatore di sorgente che rappresenta i simboli di sorgente con un numero di bit variabile, scelti nel modo indicato dalla terza colonna della tabella, e la cui lunghezza L in binit è indica alla quarta colonna, il risultato è quello di utilizzare un numero *medio* di binit/simbolo pari a

$$\overline{N} = E\{L\} = \sum_k L(k)\,p_k = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{2}{8} = 1.75 \text{ binit/simbolo}$$

La prima osservazione da fare, è che i due risultati (entropia e lunghezza media) coincidono, in virtú del fatto che i valori di probabilità sono potenze negative di 2 (ossia del tipo $p = 2^{-n}$): viceversa, nel caso in cui ciò non si verifichi, torna a risultare $\overline{N} > H$. La seconda osservazione, riguarda la

Regola del prefisso Per poter usare delle codewords a lunghezza variabile, queste devono poter essere separate le une dalle altre presso il ricevitore, e questo è possibile a patto che nessuna codeword sia uguale all'inizio di una codeword più lunga. Osservando l'esempio su riportato, ci rendiamo conto come questo sia effettivamente il caso. Poco più avanti è descritto il metodo (di Huffman) di individuare questo codebook in modo ottimo.

Codifica a blocchi Un secondo approccio tendente al conseguimento di una velocità trasmissiva più prossima all'entropia di sorgente, prevede di non codificare ogni singolo simbolo in modo indipendente, ma invece di raggrupparli assieme (realizzando così una *codifica a blocchi*) in modo da simulare la condizione di codificare una nuova sorgente virtuale, provvista di un alfabeto di dimensionalità aumentata.

Torniamo ad esaminare il caso dell'esempio di sorgente binaria senza memoria, e raggruppiamo le coppie di simboli, contraddistinte da un probabilità congiunta indicata nella seconda colonna della tabella, per poi rappresentarli con la codifica a lunghezza di parola variabile già analizzata.

Simbolo	Prob.	Codeword
x_1	.8	1
x_2	.2	0
x_1x_1	.64	0
x_1x_2	.16	10
x_2x_1	.16	110
x_2x_2	.04	111

Essendo la sorgente senza memoria, il valore della probabilità congiunta per coppie di simboli si ottiene come prodotto delle probabilità dei due simboli, e così per esempio $p(x_1x_2) = p(x_1)\,p(x_2) = 0.8 \cdot 0.2 = 0.16$.

Mentre il valore dell'entropia della sorgente binaria è sempre quello già calcolato di $H_b = 0.72$ bit a simbolo, per quanto riguarda la lunghezza media, ora questa risulta

$$\overline{N} = 1 \cdot 0.64 + 2 \cdot 0.16 + 3 \cdot 0.16 + 3 \cdot 0.04 = 1.58$$

binit ogni 2 simboli, ossia pari ad una media di 0.79 binit/simbolo, effettivamente più vicina al valore di H_b.

In generale, prendendo blocchi via via più lunghi, è possibile ridurre la velocità media di codifica $\overline{N}$ (in binit/simbolo) rendendola sempre più vicina all'Entropia, ovvero

$$\min\left[\overline{N}\right] = H_S + \varepsilon \quad con\ \varepsilon \to 0$$

se la lunghezza del blocco tende ad infinito (*Teorema di Shannon* sulla codifica di sorgente). D'altra parte, all'aumentare della dimensione del blocco, aumenta di egual misura il ritardo che intercorre tra l'emissione di un simbolo e la sua codifica, e di questo va tenuto conto, nel caso sussistano dei vincoli temporali particolarmente stringenti sulla consegna del messaggio.

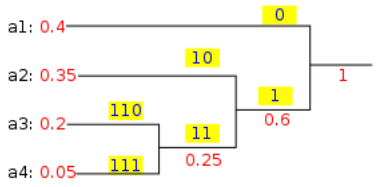
Esercizio Si ripeta il calcolo del numero medio di binit/simbolo, adottando lo stesso codice a lunghezza variabile usato finora, per codifcare i simboli emessi dalla sorgente binaria Markoviana analizzata all'esempio precedente, e mostrare come in questo caso si riesca ad ottenere una velocità media pari a 0.72 bit/simbolo. Sperimentare quindi la costruzione di un codice di Huffman (vedi appresso) basato sul raggruppamento di tre simboli di sorgente, per verificare se ci si riesce ad avvicinare di più al valore limite indicato dalla entropia, già calcolato pari a 0.58 bit/simbolo.

Codice di Huffman Formalizziamo ora il metodo di costruzione di un codice a lunghezza variabile, in modo che soddisfi automaticamente la regola del prefisso. Il metodo si basa[4]

[4]Vedi Wikipedia `http://en.wikipedia.org/wiki/Huffman_coding`

sulla costruzione di un albero binario, i cui rami sono etichettati con 1 e 0, e può essere descritto come segue:

- crea una lista contenente i simboli della sorgente, ordinati in base alle rispettive probabilità, ed associa ad ognuno di essi un nodo foglia dell'albero;
- finché c'è più di un nodo nella lista:
  - rimuovi dalla lista i due nodi con la probabilità più bassa;
  - crea un nuovo nodo interno all'albero con questi due nodi come figli, e con probabilità pari alla somma delle loro probabilità;
  - aggiungi il nuovo nodo alla lista;
- il nodo rimanente è la radice, e l'albero è completo.



Si può dimostrare che il codice di Huffman generato in questo modo è il migliore possibile nel caso in cui la statistica dei simboli di sorgente sia nota a priori, nel senso che produce una codifica con il minor numero possibile di bit/simbolo medi. La codifica di Huffman è ampiamente utilizzata nel contesto di altri metodi di compressione (metodo DEFLATE di PKZIP) e di codec multimediali (JPEG e MP3), in virtù della sua semplicità, velocità, ed assenza di brevetti.

Ovviamente ci deve essere un accordo a priori tra sorgente e destinatario a riguardo delle corrispondenze tra parole di codice e simboli (o blocchi di simboli) della sorgente. Nel caso in cui ciò non sia vero, oppure nel caso in cui la statistica dei simboli della sorgente sia stimata a partire dal materiale da codificare, occorre inviare all'inizio della comunicazione anche la tabella di corrispondenza, eventualmente codificata a sua volta.

Dynamic Huffman coding Questa variante permette di costruire e modificare l'albero di codifica[5] man mano che i simboli sono trasmessi. In questo modo, se un carattere è già presente nel codebook, viene trasmessa la codeword corrispondente, mentre se non lo è, viene trasmesso il codice del carattere, ed aggiornato il codebook. Lo stesso processo si svolge anche dal lato ricevente, permettendo una codifica in tempo reale, e l'adattamento a condizioni di variabilità nei dati. Ovviamente, il metodo inizia ad essere efficiente solo dopo aver accumulato sufficienti informazioni statistiche.

17.1.1.3 Codifica per sorgenti con memoria

Abbiamo già evidenziato come nel caso di sorgenti con memoria i valori di entropia si riducano, e come si possa ridurre anche la velocità di codifica a patto di accettare il ritardo legato all'uso di codici a blocchi. A volte però la dimensione dei blocchi da prendere in considerazione può risultare eccessiva, producendo spropositate tabelle di codeword. Inoltre si può ritenere di *non* conoscere la statistica della sorgente, e non si desidera effettuarne una stima e quindi trasmetterla. In questi casi, può essere opportuno adottare tecniche diverse dalle precedenti, come le due riportate appresso.

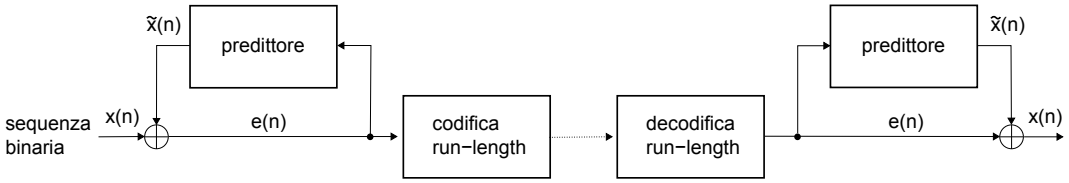
[5] Presso Wikipedia si trova una descrizione dell'Algoritmo di Vitter http://en.wikipedia.org/wiki/Adaptive_Huffman_coding

Codifica run-length Prendendo come esempio tipico il caso della trasmissione fax, si ha a che fare con un segnale di immagine in bianco e nero, scansionato per righe, che è assimilabile ad una sorgente binaria che emetta uno zero per il bianco, ed un uno per il nero: per la natura delle immagini scansionate, tipicamente ci saranno lunghe sequenze di uni o di zeri, e dunque si può assumere valido un modello di sorgente Markoviano di primo ordine, con elevate probabilità condizionate di restare nello stesso stato.

Le lunghe sequenze di bit tutti uguali vengono dette *run*, e la codifica *run-length* consiste effettivamente nel trasmettere una parola di codice che indica il numero (*length*) di questi bit uguali. In questo caso quindi, la codeword è di lunghezza fissa (ad esempio $k+1$ binit, il primo dei quali indica se il run è tutto di uni o di zeri), e rappresenta un numero variabile (da 0 a 2^k-1) di bit di sorgente. Se ad esempio $k=6$ binit, questi 6+1 = 7 binit possono codificare fino a 64 bit di immagine: un bel risparmio![6]

Codifica predittiva Questa ulteriore tecnica si basa sul fatto che un elevato grado di dipendenza statistica dei messaggi comporta la possibilità di *predire* in qualche modo i simboli a venire, in base all'identità di quelli già emessi. La differenza tra la sequenza predetta $\tilde{x}(n)$ e quella effettiva $x(n)$ è una nuova sequenza indicata come *errore di predizione* $e(n)=\tilde{x}(n)-x(n)$ che, se il predittore *ci azzecca* per la maggior parte del tempo, è quasi tutta nulla.

Nella figura seguente mostriamo l'applicazione della tecnica al caso di sequenze binarie, per le quali l'operazione di differenza è realizzata tramite una somma modulo due, in modo che nel caso di predizione corretta, l'errore sia nullo.



Il predittore conserva uno stato interno che rappresenta gli ultimi bit di ingresso, in base ai quali determina[7] la stima $\tilde{x}(n)$; l'errore $e(n)$ che è frequentemente nullo, e che viene sottoposto a codifica run-length, è re-inserito anche nel predittore, in modo che questo possa ri-determinare l'ultimo simbolo di ingresso $x(n)=e(n)\oplus\tilde{x}(n)$, ed aggiornare il proprio stato interno. La medesima formula di riscostruzione viene applicata anche in uscita del predittore di ricezione, che condivide con quello di trasmissione la conoscenza dello stato iniziale del segnale trasmesso, in modo da evolvere allo stesso modo.

Notiamo come la codifica run-length non preveda l'esistenza di un accordo a priori tra trasmettitore e ricevitore, a parte il comune stato di partenza ad inizio messaggio, e la

[6] In realtà, nel caso specifico del fax le cose non stanno esattamente in questi termini: infatti, anziché usare una parola di lunghezza fissa di k binit, l'ITU-T ha definito un apposito codebook `http://www.itu.int/rec/T-REC-T.4-199904-S/en` contenente un codice di Huffman a lunghezza variabile, in modo da codificare le run lenght più frequenti con un numero ridotto di bit.

[7] Il lettore più curioso si chiederà a questo punto, come è fatto il predittore. Molto semplicemente, *scommette* sul prossimo simbolo più probabile, in base alla conoscenza di quelli osservati per ultimi, ed ai parametri del modello markoviano: se questo simbolo possiede una probabilità a priori > 0.5, allora *la maggioranza* delle volte la predizione sarà corretta, ed il metodo inizia a consentire una riduzione di velocità. Nel caso di sorgenti continue, troveremo invece alcune particolarità agguntive.

medesima struttura del predittore. Per contro, in presenza di errori di trasmissione i due predittori restano disallineati, finchè non si inizia a co-decodificare un nuovo messaggio. Ma lo stesso problema, è comune anche al caso di codifica a lunghezza di parola variabile, ed a quello di Huffman dinamico.

17.1.1.4 Compressione basata su dizionario

Nella comune accezione del termine, un dizionario è costituito da un *array* di stringhe, popolato con le parole esistenti per un determinato linguaggio. Anzichè operare carattere per carattere, un codificatore di sorgente testuale può ricercare la posizione nel dizionario delle parole del messaggio, e quindi trasmettere l'indice della parola: per un dizionario di 25.000 termini bastano 15 bit di indirizzamento, ottenendo un rapporto di compressione variabile, in funzione della lunghezza della parola codificata.

Metodo di Lempel-Ziv-Welsh Per evitare di dover condividere la conoscenza dell'intero dizionario tra sorgente e destinatario, che tra l'altro potrebbe essere assolutamente sovradimensionato rispetto alle caratteristiche dei messaggi da trattare, il metodo LZW prevede che il codificatore generi il dizionario in modo graduale, man mano che analizza il testo, e che il decodificatore sia in grado di replicare questa modalità di generazione. Inoltre, il dizionario non è limitato a contenere parole del messaggio, ma ospita le sequenze di caratteri effetivamente osservati, di lunghezza due, tre, quattro...

Operando su di un alfabeto ad L simboli, rappresentabili con $n = \lceil \log_2 L \rceil$ bit, il dizionario conterrà i simboli di sorgente alle prime L posizioni, e posti liberi nelle restanti $2^n - L - 1$ posizioni[8].

Ogni carattere letto in ingresso viene accodato in una FIFO che viene confrontata con le stringhe già presenti nel dizionario e che, in caso negativo, viene aggiunta come nuova voce di vocabolario; quindi, viene trasmesso il codice associato alla sua parte iniziale, escludendo cioè il simbolo concatenato per ultimo, e che ha determinato l'occorrenza della nuova stringa mai vista. La prima parte del testo, ovviamente, ha una alta probabilità di contenere tutte copie di caratteri mai viste prima, e quindi in questa fase vengono semplicemente emessi i codici associati ai simboli osservati.

```
w = NIL;
while (read a char c) do
  if (wc exists in dictionary) then
    w = wc;
  else
    add wc to the dictionary;
    output the code for w;
    w = c;
  endif
done
output the code for w;
```

Nel caso invece in cui la stringa sia già presente (e questo in particolare è vero per la stringa di lunghezza uno corrispondente al primo simbolo analizzato) non si emette nulla, ma si continuano a concatenare simboli fino ad incontrare una stringa mai vista. Presso wikipedia `http://en.wikipedia.org/wiki/Lempel-Ziv-Welch` è presente un esempio di risultato della codifica.

Ogni volta che viene esaurito lo spazio residuo per i nuovi simboli, viene aggiunto un bit alla lunghezza della codeword, ovvero viene raddoppiata la dimensione del vocabolario. Man mano che viene analizzato nuovo materiale, aumenta la lunghezza delle stringhe memorizzate nel dizionario, che riflette l'effettiva composizione statistica del

[8] Ad esempio con L=96 simboli si ha n=7 bit, ed un dizionario con 128 posizioni, di cui 96 occupate e 32 libere.

documento in esame, ivi compresa la presenza di memoria; allo stesso tempo, la dimensione del dizionario (e la lunghezza delle codeword) resta sempre la minima indispensabile per descrivere il lessico effettivamente in uso. L'algoritmo LZW è usato nel programma di compressione Unix *Compress*, per la realizzazione di immagini GIF e TIFF, ed incorporato in altri software, come ad esempio *Adobe Acrobat*.

Algoritmo Deflate L'ultimo metodo di compressione senza perdite che esaminiamo è quello che è stato introdotto dal programma PKZIP, e quindi formalizzato nella rfc 1951 `http://tools.ietf.org/html/rfc1951`, e tuttora ampiamente utilizzato per le sue ottime prestazioni e l'assenza di brevetti. Usa una variante dell'algoritmo LZW, al cui risultato applica poi una codifica di Huffman. Deflate opera su blocchi di dati con dimensione massima 64Kbyte, ognuno dei quali può essere replicato intatto (come nel caso in cui i bit siano già sufficientemente impredicibili), oppure essere compresso con un codice di Huffman statico, oppure ancora dinamico.

Per quanto riguarda la variante di LZW, essa consiste nel *non costruire* esplicitamente il dizionario, ma nell'usare invece *dei puntatori all'indietro* per specificare che una determinata sotto-stringa di ingresso, è in realtà la ripetizione di un'altra già osservata in precedenza. In questo caso, anziché emettere il codice (di Huffman) associato al byte corrente, si emette (il codice di Huffman del) la lunghezza della stringa da copiare, e la distanza (nel passato) della stessa. Quindi in pratica, anzichè usare una codeword di lunghezza fissa per indicizzare gli elementi del dizionario come per LWZ, viene usato un puntatore di lunghezza variabile, privilegiando le copie della sottostriga corrente più prossime nel tempo, oppure quelle con un maggior numero di caratteri uguali.

17.1.2 Codifica con perdite di sorgente continua

Senza scendere nei dettagli analitici che stanno alla base di questo risultato, abbiamo più volte discusso come la ricezione di un segnale $r(t)$ all'uscita di un canale con banda W, ed alla cui uscita sia presente un rumore $n(t)$ gaussiano bianco di potenza N, non possa corrispondere ad un tasso di informazione superiore alla *capacità di canale* $C = W \log_2\left(1 + \frac{S}{N}\right)$ bit per secondo. Notiamo ora che

- il rumore al ricevitore può essere visto come una distorsione $d(t) = n(t) = x(t) - r(t)$ di potenza $D = N$, in cui $x(t)$ rappresenta il segnale trasmesso
- variando l'entità della distorsione D, ossia variando $\frac{S}{N}$, varia anche C, e dunque la massima velocità possibile.

Curva velocità-distorsione In base a queste considerazioni, si arriva (dopo una serie di sviluppi teorici che vengono omessi) alla conclusione che nel caso in cui la sorgente $x(t)$ sia gaussiana (stazionaria ed ergodica) con potenza σ_x^2, la minima distorsione conseguibile in corrispondenza ad una velocità di trasmissione pari ad R, assume un valore pari a

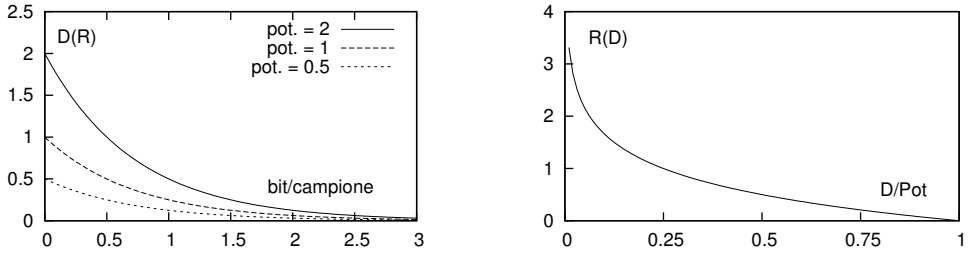
$$D(R)_G = 2^{-2R}\sigma_x^2$$

che risulta essere *il più grande* valore minimo possibile, dato che per sorgenti non gaussiane, oppure gaussiane ma non bianche, si possono ottenere valori inferiori. D'altra parte, è definita anche la funzione inversa, ovvero la curva $R(D)_G$, che descrive la velocità

necessaria per trasmettere i campioni di una sorgente gaussiana

$$R(D)_G = \begin{cases} \frac{1}{2}\log_2 \frac{\sigma_x^2}{D} & \text{se} \quad 0 \leq D \leq \sigma_x^2 \\ 0 & \text{se} \quad D \geq \sigma_x^2 \end{cases}$$

e che può essere interpretata osservando che, per riscontrare una distorsione superiore alla potenza di segnale, non occorre trasmettere proprio nulla, dato che tanto il ricevitore può ri-generare un segnale di errore, a partire da un rumore gaussiano di pari potenza prodotto in loco.

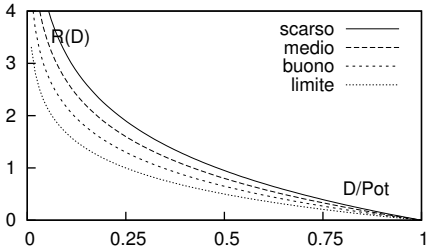


Valori limite Il valore $D(R)_G$ costituisce un *Limite Superiore* per ciò che riguarda la distorsione ottenibile ad una certa velocità R, utile per rapportare le prestazioni del codificatore della nostra sorgente con quelle migliori ottenibili per la sorgente più *difficile*, ossia la gaussiana. Allo stesso tempo, è definito un *Limite inferiore* $D(R)_L$ (ossia, la minima distorsione sotto cui non si può scendere per un dato R) per sorgenti *non* gaussiane e *senza memoria*, in modo da poter scrivere

$$D(R)_L = \frac{2^{-2R} \cdot 2^{2h(x)}}{2\pi e} \leq D(R) \leq D(R)_G = 2^{-2R}\sigma_x^2 \tag{17.3}$$

in cui $h(x)$ è l'entropia di una sorgente continua x.

In definitiva, per una determinata sorgente per la quale sono disponibili diversi codificatori, potremmo ottenere una famiglia di curve del tipo di quelle mostrate in figura.



Entropia di sorgente continua Estendendo formalmente al caso continuo la definizione trovata per le sorgenti discrete, si ottiene un valore indicato come *entropia differenziale* e pari a

$$h(X) = E\{-\log_2 p_x(x)\} = -\int p_x(x)\log_2 p_x(x)\,dx$$

che viene detta *differenziale* perché il suo valore può risultare positivo, negativo o nullo, in funzione della ampiezza dinamica della variabile aleatoria X.

Esempio Se calcoliamo il valore di entropia differenziale per un processo i cui valori sono descritti a una variabile aleatoria a distribuzione uniforme $p_x(x) = \frac{1}{A} rect_A(x)$, otteniamo il risultato $h(X) = -\frac{1}{A}\int_{-\frac{A}{2}}^{\frac{A}{2}} \log_2\left(\frac{1}{A}\right) dx = \log_2 A$ il cui valore effettivo, appunto, dipende dal valore di A.

Sebbene inadatta ad esprimere il contenuto informativo assoluto di una sorgente continua, l'entropia differenziale può comunque essere utile per confrontare tra loro due sorgenti con uguale varianza; in particolare, il suo massimo valore è ottenuto in corrispondenza di un processo gaussiano, e risulta pari a

$$h(X)_G = \frac{1}{2}\log_2\left(2\pi e\sigma_x^2\right) > h(X)$$

ed è per questo motivo che, a parità di velocità R, il processo gaussiano incorre nella massima distorsione. Associata alla definizione di entropia differenziale, sussiste quella di *potenza entropica* Q, scritta come

$$Q = \frac{1}{2\pi e} 2^{2h(X)}$$

che per sorgenti gaussiane fornisce $Q_G = \sigma_x^2$, mentre per altri tipi di statistiche, si ottiene un valore minore. Applicando questa definizione alla (17.3), il limite inferiore di distorsione si ri-scrive quindi come $D(R)_L = 2^{-2R}Q$.

Sorgenti con memoria Come per il caso di sorgenti discrete, l'esistenza di una dipendenza statistica tra i valori emessi da una sorgente continua riduce la quantità di informazione emessa, al punto che a parità di distorsione, queste possono essere codificate a velocità ridotta, ovvero a parità di velocità, può essere conseguita una distorsione inferiore. Anche stavolta, la sorgente più *difficile* è quella gaussiana, per la quale risulta che la distorsione minima può scriversi come

$$D(R)_G = 2^{-2R}\gamma_x^2\sigma_x^2$$

in cui $\gamma_x^2 \leq 1$ rappresenta una misura di *piattezza spettrale*, con il segno di $=$ valido nel caso senza memoria, ovvero di processo *bianco*, mentre nel caso opposto, essendo i campioni correlati tra loro, la densità spettrale non è più bianca.

17.1.3 Codifica di Immagine

Un segnale di immagine può essere sia di tipo vettoriale[9], come ad esempio un disegno generato al computer, che è descritto mediante un linguaggio descrittivo che codifica le operazioni grafiche necessarie alla sua realizzazione, oppure si tratta di un segnale di tipo bitmap, o *raster* (griglia, reticolo), come nel caso di una foto digitale, di un fax, o del risultato di un processo di scansionamento elettronico. Mentre nel caso dell'immagine vettoriale, questa è pienamente scalabile e ridimensionabile senza perdita di definizione, le immagini bitmap sono ottimizzate per essere riprodotte nelle loro dimensioni originali, avendo già operato un processo di distorsione tale da sfruttare al più possibile le caratteristiche di predicibilità e di sensibilità percettiva.

[9] Esempi di formati per la grafica vettoriale sono SVG, EPS, PDF, e VRML.

17.1.3.1 Dimensioni

Per quanto riguarda le immagini bitmap, queste sono definite nei termini di una matrice di elementi di immagine o PIXEL (*picture elements*)[10], che sono l'equivalente dei campioni estratti da un segnale unidimensionale. Per ogni pixel è definito un valore associato alla intensità con la quale deve essere riprodotto: nel caso di immagini a colori, sono necessari tre valori di intensità, per cui una immagine è in realtà descritta da tre matrici, come approfondiamo di seguito.

Sebbene le dimensioni della matrice di pixel possano essere qualunque, nel corso del tempo si sono affermate una valori di riferimento, associati ad altrettante serie di sigle, legate al tipo di dispositivo che deve poi riprodurre l'immagine, ma anche a quello da cui l'immagine viene acquisita

	banda	*linee*	*fps*	*aspetto*	*colonne*	*righe*	*colore*
PAL	6 MHz	625	25 int	4:3			
NTSC	5 MHz	525	30 int	4:3			
HDTV		1080		4:3	1440	1152	
				16:9	1920	1152	
SVGA				4:3	1024	768	
4:2:2		625/525	50/60 non int	4:3	720	576/480	360 x 576/480
4:2:0		625/525	25/30 int	4:3	720	576/480	360 x 288/240
VGA				4:3	640	480	
SIF		625/525	25/30 non int	4:3	360	288/240	180 x 144/120
CIF			30 non int	4:3	360	288	180 x 144
QCIF			15 - 7.5 non int.	4:3	180	144	90 x 72

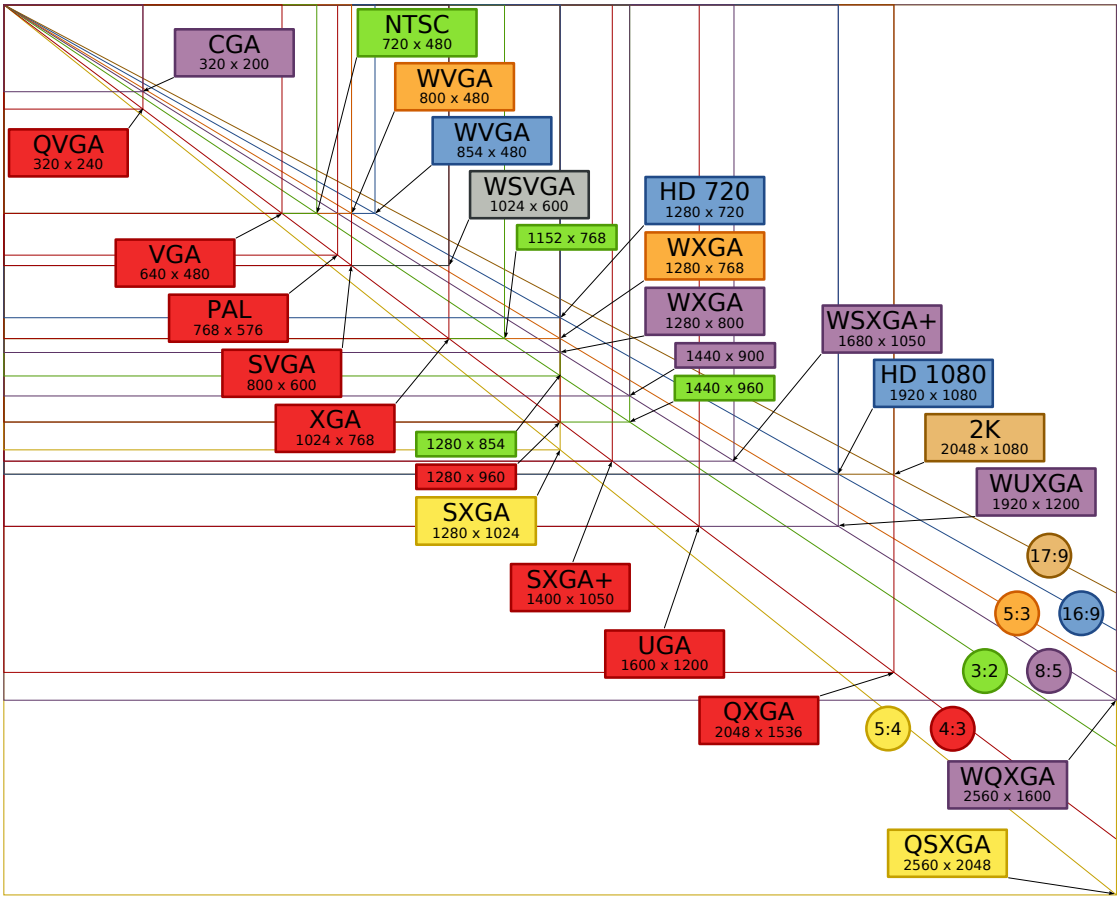
Ad esempio, la risoluzione VGA (640 x 480) trae origine dai parametri dello standard NTSC della televisione analogica, i cui quadri sono composti da una serie di 525 linee, di cui solo 480 visibili: volendo mantenere lo stesso rapporto d'aspetto di 4:3, ogni linea deve essere campionata su 480/3x4 = 640 punti. Prima ancora dell'uso broadcast della TV digitale, la raccomandazione BT 601[11] stabilisce le regole per la conversione tra standard video differenti, mediante l'uso di una comune frequenza di campionamento del segnale video a 13.5 MHz, individuando così nei 52 μsec di una linea 52 x 10^{-6}x 13.5 ^{6} = 702 campioni per linea, a cui si aggiungono 9 campioni neri in testa ed in coda per ottenere 720 campioni per linea; per un segnale a 525 linee si ottiene quindi la matrice 720 x 480 del formato 4:2:2, che approfondiremo tra breve.

Le matrici più grandi di 1024 x 768 sono spesso descritte in termini di *Megapixel* (es 1600 x 1200 = 1,9 Mpixel), spesso usati per confrontare la risoluzione (ma non necessariamente la qualità) delle moderne macchine fotografiche digitali; inoltre, i *grandi formati* traggono origine anche dalla tecnologia delle schede video per computer da un lato, e da quella della televisione ad alta definizione da un altro, come riassunto nella figura seguente[12].

[10] Per alcuni anni, si è usato come sinonimo anche il termine PEL `http://www.foveon.com/files/ABriefHistoryofPixel2.pdf`.

[11] Il sito di ITU-R `http://www.itu.int/ITU-R/index.asp?category=information&link=rec-601&lang=en` non consente l'accesso pubblico alla raccomandazione. Un approfondimento può essere svolto presso Wikipedia `http://it.wikipedia.org/wiki/BT.601`.

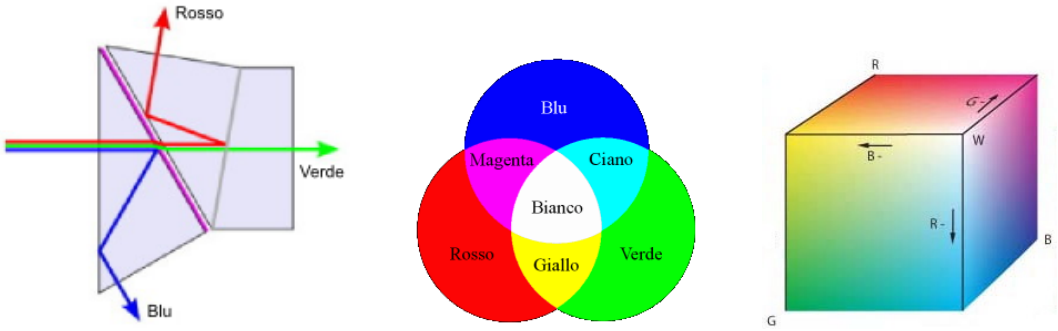
[12] La figura è tratta da Wikipedia, dove possono essere approfonditi gli altri aspetti legati a queste risoluzioni video `http://it.wikipedia.org/wiki/Risoluzioni_standard`.



Il formato SIF (*source intermediate format*) è ottenuto a partire dal 4:2:2, conservando la metà dei pixel sia in verticale che in orizzontale, e trascurando la metà dei quadri di immagine; il suo uso è orientato alla memorizzazione, e quindi usa una scansione non interallacciata. Il formato CIF (*common intermediate format*) è simile al SIF, tranne per aver perso il riferimento al numero di linee analogiche da cui deriva; il suo uso è orientato ai sistemi di videoconferenza, e da questo sono definiti formati a maggior risoluzione, come il 4CIF ed il 16CIF, equivalenti al 4:2:2 ed all'HDTV. Il formato QCIF (*quarter* CIF) è orientato alla videotelefonia, dimezzando ancora sia la risoluzione spaziale che quella temporale. Da questo è a sua volta derivato il formato SUB-QCIF (o S-QCIF) di 128 x 96 pixel, orientato a collegamenti lenti come quelli via modem.

17.1.3.2 Spazio dei colori

I dispositivi di acquisizione e riproduzione di immagini a colori operano su tre diverse matrici di pixel, che rappresentano i tre colori di base della *sintesi additiva*, ossia *Rosso*, *Verde*, e *Blu*, o RGB (dalle iniziali inglesi). Alla figura seguente viene mostrato il principio di funzionamento di un *prisma dicroico*, che devia le tre componenti di colore verso tre diversi dispositivi di acquisizione. Variando quindi la proporzione con cui si sommano gli stimoli dei tre colori, si ottiene, oltre al bianco, anche qualunque altro colore.



Sebbene dalle figure riportate sembra che il bianco risulti dal contributo in parti uguali delle tre componenti RGB, in realtà la scala di grigi della immagine *monocromatica* corrispondente si ottiene calcolando un segnale Y di *luminanza* secondo la formula

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{17.4}$$

che è quella usato per modulare il segnale video analogico. Come già discusso, in tale ambito la componente di colore viene trasmessa utilizzando due altri segnali, C_b o *Crominanza Blu* e C_r o *Crominanza Rossa*, secondo la formula

$$C_b = B - Y \quad \text{e} \quad C_r = R - Y \tag{17.5}$$

Disponendo dei segnali Y, C_be C_r, si possono riottenere i valori RGB inserendo la (17.4) nelle (17.5), e risolvendo il sistema di tre equazioni in tre incognite risultante.

Segnale video composito Al §**??** abbiamo descritto come nel segnale televisivo analogico, la componente di colore sia trasmessa assieme alla luminanza, su di una diversa portante, con modulazione in fase e quadratura. In realtà, per diversi motivi, le componenti trasmesse non sono direttamente quelle individuate dalle (17.5), ma piuttosto componenti denominate U, V oppure I, Q, e così definite:

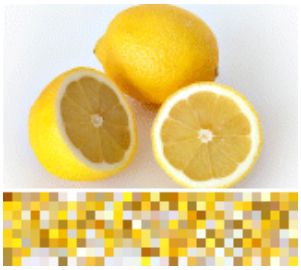
$$\begin{aligned} PAL: \quad U &= 0.493 \cdot C_b \\ V &= 0.877 \cdot C_r \end{aligned}$$

$$\begin{aligned} NTSC: \quad I &= 0.74 \cdot C_r - 0.27 \cdot C_b \\ Q &= 0.48 \cdot C_r + 0.41 \cdot C_b \end{aligned}$$

Pertanto, in funzione della diverse modalità di rappresentazione, un segnale video a colori può essere descritto indifferentemente da una delle seguenti quattro terne di segnali: RGB, YC$_r$C$_b$, YUV, YIQ.

Una descrizione alternativa dello spazio di colore è fornita dai parametri di *Tinta*, *Saturazione* e *Chiarezza*, ovvero HUE, SATURATION e LIGHTNESS, o HSL: si tratta di attributi più legati alla descrizione percettiva che non alle tecnologie della riproduzione dell'immagine. Mentre la tinta descrive una famiglia di colori (es tutti i rossi), la saturazione ne indica il grado di purezza, ossia la presenza congiunta di altre tonalità; la chiarezza, infine, denota la luminosità del colore, rispetto ad un punto bianco. La terna HSL viene a volte usata per descrivere un colore mediante programmi di grafica, mediante i quali è fornito anche l'equivalente RGB.

Profondità di colore Dato che l'occhio umano non distingue più di 250 tinte diverse, e di 100 livelli di saturazione, si ritiene che utilizzare 8 bit per ogni componente dello spazio di colore RGB sia più che sufficiente. Con 8x3=24 bit per pixel (bpp) si possono infatti rappresentare $2^{24} - 1$ diversi colori, ovvero più di 16 milioni, molti dei quali indistinguibili ad occhio umano. Modalità più spinte di quella a 24 bpp (detta *truecolor*), che adottando 10, 12, 16 bpp, o rappresentazioni in virgola mobile, sebbene non migliorino la qualità visiva, possono comunque essere usate in contesti professionali, per non perdere precisione nelle operazioni di editing ripetuto. Al contrario, profondità inferiori sono comunemente usate per risparmiare memoria, come nel caso di 15 bpp, che usa 5 bit per componente, o 16 bpp, che usa 6 bit per il verde, offrendo 65.536 colori diversi.



Nel caso si decida di adottare profondità molto ridotte, come 8 bpp, si preferisce ricorrere ad una modalità detta a *colore indicizzato*: l'insieme dei colori presenti nell'immagine viene *quantizzato*[13] in un insieme ridotto, i cui valori a 24 bpp sono memorizzati in una tavolozza (nota come *palette* od anche *colour look-up table* o CLUT), che viene quindi utilizzata come un dizionario. La figura a lato mostra una immagine di esempio, assieme alla palette dei colori che usa. In questo modo, per ogni pixel dell'immagine è ora sufficiente specificare l'indice della palette dove è memorizzata la rappresentazione a 24 bpp del colore più prossimo: ad es. con una palette di 256 elementi da 24 bit, si può accedere a 256 diversi colori scelti tra 16 milioni.

Sottocampionamento del colore Nella tabella riportata a pag. 313 è presente la colonna colore, che mostra come la dimensione riservata alle matrici di pixel che codificano le informazioni di crominanza sia ridotta di metà rispetto a quella della luminanza. Questo fatto trae origine da due buoni motivi: il primo è che l'acutezza visiva dell'occhio umano per ciò che riguarda le variazioni cromatiche è ridotta rispetto a quella relativa alle variazioni di luminosità; il secondo è che il segnale di crominanza presente nel segnale video composito occupa una banda circa metà di quella del segnale di luminanza. Pertanto, le componenti di luminanza sono generalmente campionate con una risoluzione spaziale inferiore a quella del segnale di luminanza. Il tipo di sotto-campionamento spaziale adottato per le componenti di crominanza è generalmente caratterizzato da quattro numeri, in accordo allo schema seguente.

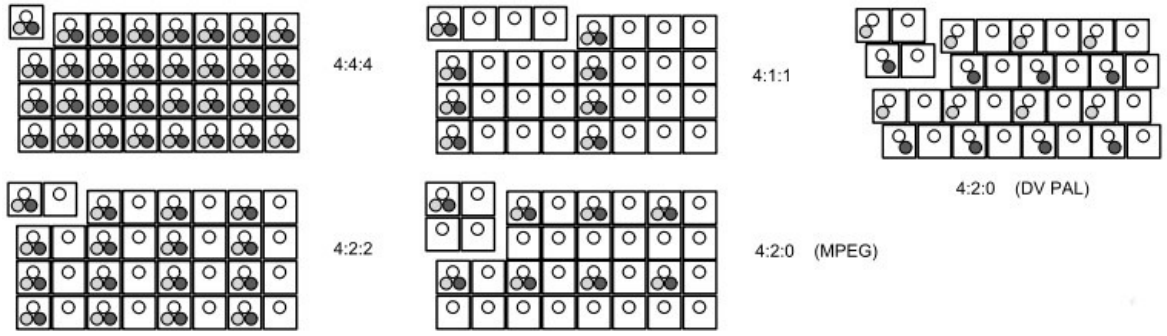
4:4:4 Non si effettua sottocampionamento, e le tre componenti hanno lo stesso numero di campioni. Applicato principalmente a segnali RGB trattati in studio di produzione.

4:2.2 Questo schema si applica tipicamente alle rappresentazioni $\mathrm{YC}_b\mathrm{C}_r$, memorizzando per ogni 4 campioni di luminanza, 2 campioni della componente C_b e 2 della componente C_r, ed è utiizzato in ambito professionale e broadcast.

4:1:1 In questo caso ogni quattro campioni di luminanza su una riga, ne viene preso uno per C_b ed uno per C_r. E' lo schema usato nello standard DV NTSC.

[13] Per una breve introduzione alla *quantizzazione cromatica*, può essere consultata Wikipedia `http://en.wikipedia.org/wiki/Color_quantization`

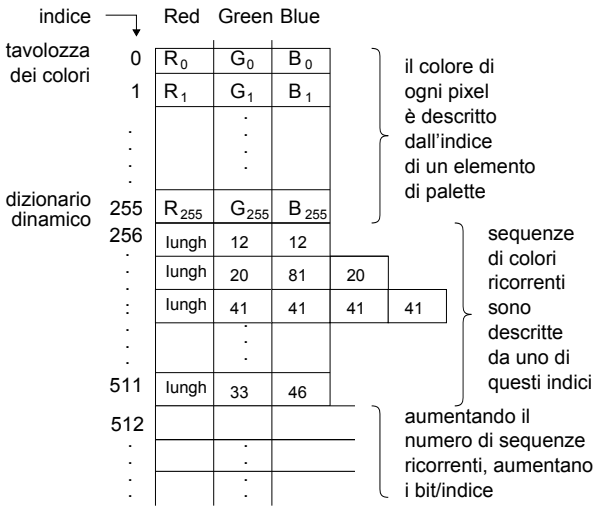
4:2:0 Ogni 4 campioni di luminanza, ne vengono salvati uno per C_b ed uno per C_r come per il caso 4:2:1, ma ora la crominanza è campionata su righe alterne. In particolare, la versione utilizzata per l'MPEG-1 campiona assieme entrambi i segnali di crominanza, una riga si ed una no, mentre quella usata con il DV PAL li campiona a righe alternate, e prevede una riproduzione in modalità interallacciata.



17.1.3.3 Formato GIF

Il *Graphics Interchange Format* è un formato ad 8 bpp definito da *CompuServe* nel 1987[14] e da allora ha continuato ad essere molto popolare. Usa una *palette* con cui rappresentare 256 colori scelti tra 16 milioni, e quindi li comprime mediante l'algoritmo LWZ, individuando configurazioni ricorrenti dei valori di colore. Un singolo file può contenere più immagini (ognuna con la sua palette) in modo da realizzare brevi animazioni. Il numero ridotto di colori rende il formato poco idoneo alla riproduzione di fotografie, ma più che adatto ad immagini più semplici, come ad es. un logo di pagina web. Per rappresentare i colori assenti dalla palette, il codificatore può ricorrere ad una operazione di *dithering,* alternando colori che, osservati da lontano, ricreano l'effetto della tonalità mancante.

Il metodo di compressione è illustrato con l'ausilio della figura a fianco. Inizialmente, ogni pixel è rappresentato con gli 8 bit che indicizzano la terna RGB a 24 bit nella palette. Quando si incontra una sequenza di codici di colore già osservata, viene aggiunta una riga alla tabella, ed il valore dell'indice corrispondente viene usato per rappresentare tutta la sottosequenza; eventualmente, il numero di bit usati per descrivere questi indici viene aumentato di uno. Per disegnare le sequenze di pixel rappresentate da indici inclusi nella sezione dinamica della tabella, occorre dunque individuare prima le rispettive terne RGB nella tavolozza.



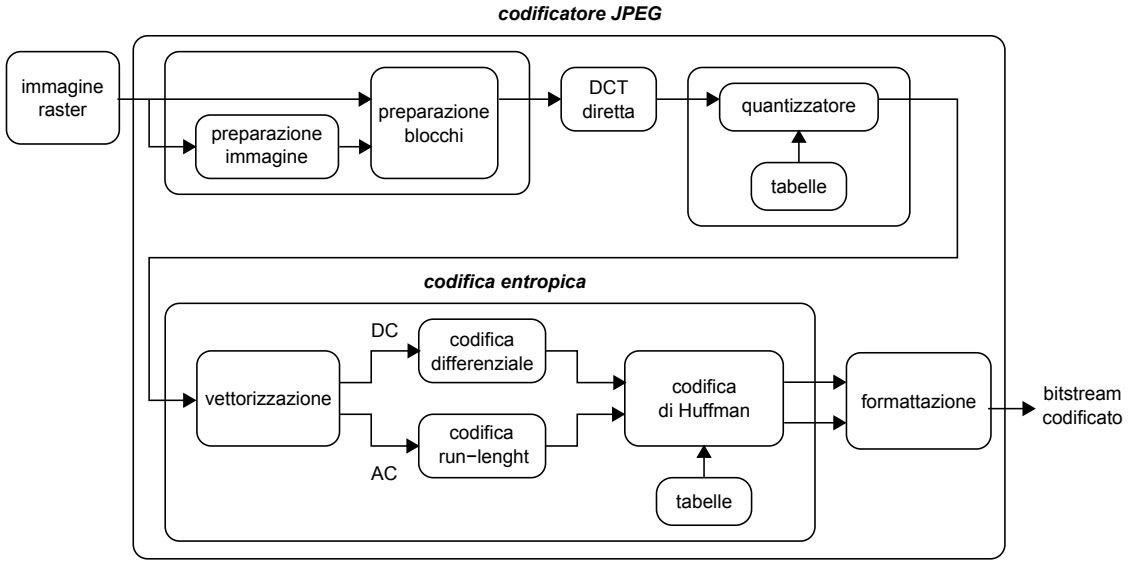
[14] Il documento di specifica `http://www.w3.org/Graphics/GIF/spec-gif89a.txt` può essere trovato presso W3C

PNG Dato che la compressione LZW era stata brevettata, venne sviluppata una codifica alternativa, denominata *Portable Network Graphics*. Al giorno d'oggi i brevetti relativi al formato GIF sono tutti scaduti, ed il formato PNG è stato standardizzato nella RFC 2083[15]. Come per GIF, anche PNG è di tipo *lossless* (senza perdite), ossia individua una compressione invertibile, capace di replicare in modo identico l'immagine di partenza, ovviamente senza considerare il processo di quantizzazione che porta alla generazione della palette. Oltre alla modalità di colore indicizzato, PNG offre anche una modalità *truecolor* a 24 o 32 bpp, e per questo può correttamente rappresentare anche materiale fotografico, al punto da consigliare l'uso di PNG (anzichè JPEG) nel caso di prevedano successive operazioni di editing dell'immagine.

Per quanto riguarda la compressione, PNG fa usa dell'algoritmo *deflate*, preceduto da un passaggio di compressione differenziale, in cui al valore che rappresenta il colore di un pixel viene sottratto il valore predetto a partire dai pixel adiacenti: in tal modo l'algoritmo *deflate* riesce a conseguire rapporti di compressione più elevati, riuscendo quasi sempre a battere le prestazioni di GIF.

17.1.3.4 Codifica JPEG

Il *Joint Photographic Experts Group* è un comitato congiunto ISO/ITU che ha definito lo standard internazionale per la compressione di immagini ISO 10918-1[16], particolarmente adatto alla codifica di immagini fotografiche. Descriviamo di seguito il funzionamento della modalità operativa detta baseline, o lossy sequential mode, che è quella che offre il migliore grado di compressione, e che prevede cinque stadi di elaborazione: preparazione dei blocchi, Discrete Cosine Transform (DCT), quantizzazione, codifica entropica, e formattazione.



Preparazione dell'immagine e dei blocchi L'immagine *raster* di partenza è formata da una o più matrici bidimensionali di valori (scala di grigi, oppure a colori indicizzati, o

[15] Reperibile presso il sito di IETF `http://tools.ietf.org/html/rfc2083`
[16] Scaricabile presso il W3C `http://www.w3.org/Graphics/JPEG/itu-t81.pdf`

RGB, $\mathrm{YC}_r\mathrm{C}_b$, YUV, ...), eventualmente di dimensioni differenti (come nel caso $\mathrm{YC}_r\mathrm{C}_b$). Sebbene sia possibile elaborare direttamente una rappresentazione RGB, le migliori prestazioni si ottengono nello spazio $\mathrm{YC}_r\mathrm{C}_b$ con sotto-campionamento spaziale 4:2:2 o (meglio) 4:2:0, e dunque il primo passo è quello di convertire l'immagine in questa modalità di rappresentazione.

Ogni matrice viene quindi suddivisa in *blocchi* della dimensione di 8x8 pixel[17], ognuno dei quali è elaborato in sequenza in modo indipendente dagli altri.

DCT diretta Prima di procedere, la matrice Y (oppure le tre matrici R, G e B) che contiene valori ad 8 bit tutti positivi, viene normalizzata sottraendo ad ogni pixel il valore 128, in modo da ottenere valori tra -128 e 127. Quindi, per ogni blocco di 8x8 pixel, i cui valori indichiamo con $p(x, y)$, viene calcolata una nuova matrice di 8x8 valori $D(i, j)$ ottenuti come coefficienti di una *trasformata coseno discreta* (DCT) bidimensionale:

$$D(i,j) = \frac{1}{4} c_i c_j \sum_{x=0}^{7} \sum_{y=0}^{7} p(x,y) \cos \frac{(2x+1)\, i\pi}{16} \cos \frac{(2y+1)\, j\pi}{16}$$

in cui c_i e c_j sono ognuno pari a $1/\sqrt{2}$ con i oppure j pari a zero, oppure $c_i = c_j = 1$ negli altri casi, e gli indici i e j variano tra zero e sette. Tralasciando di approfondire le relazioni esistenti tra DCT e DFT, consideriamo invece come i coefficienti $D(i, j)$ così ottenuti permettano la ricostruzione della matrice originaria nei termini di una somma pesata delle superfici rappresentate (per mezzo di una scala di grigi) nel diagramma riportato alla figura 17.2, mediante l'applicazione della DCT *inversa*

$$p(x,y) = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} c_i c_j D(i,j) \cos \frac{(2x+1)\, i\pi}{16} \cos \frac{(2y+1)\, j\pi}{16}$$

Ma se fosse tutto qui, non avremmo realizzato la funzione di compressione!

Per approfondire il significato di questa rappresentazione, osserviamo che ognuna della superfici elementari rappresentate in fig. 17.2 è legata ad una coppia i, j associata ad un coefficiente della DCT calcolata, in modo che tale coefficiente esprime il contenuto di frequenze spaziali descritto da quella particolare funzione della base. Per questo l'elemento $(i, j) = (0, 0)$ in alto a sinistra, ad andamento costante, è indicato come *coefficiente* DC, o componente continua, dato che essendo calcolato come somma di tutti i pixel, riflette un valore che è legato alla intensità media dell'intero blocco. I coefficienti legati alle funzioni della prima riga rappresentano contenuti di frequenza spaziale orizzontale, con un periodo via via minore spostandosi verso il margine destro, mentre quelli della prima colonna, frequenze verticali. I coefficienti localizzati all'interno della matrice esprimono contenuti di frequenze spaziali in entrambe le direzioni, con valori di frequenza tanto più elevati, quanto più ci si sposta verso l'angolo in basso a destra. Pertanto, i coefficienti descritti da indici diversi da $(0, 0)$sono indicati come *coefficienti* AC.

L'esperienza pratica mostra come quasi sempre i coefficienti $D(i, j)$ presentino nella regione in alto a sinistra valori ben più elevati di quelli riscontrabili in basso a destra,

[17] Notiamo incidentalmente come le dimensioni definite nella tabella di pag 313 siano multipli interi di 8. Se questo non è il caso, i blocchi ai bordi destro ed inferiore vengono riempiti con pixel scelti in modo da minimizzare le distorsioni risultanti.

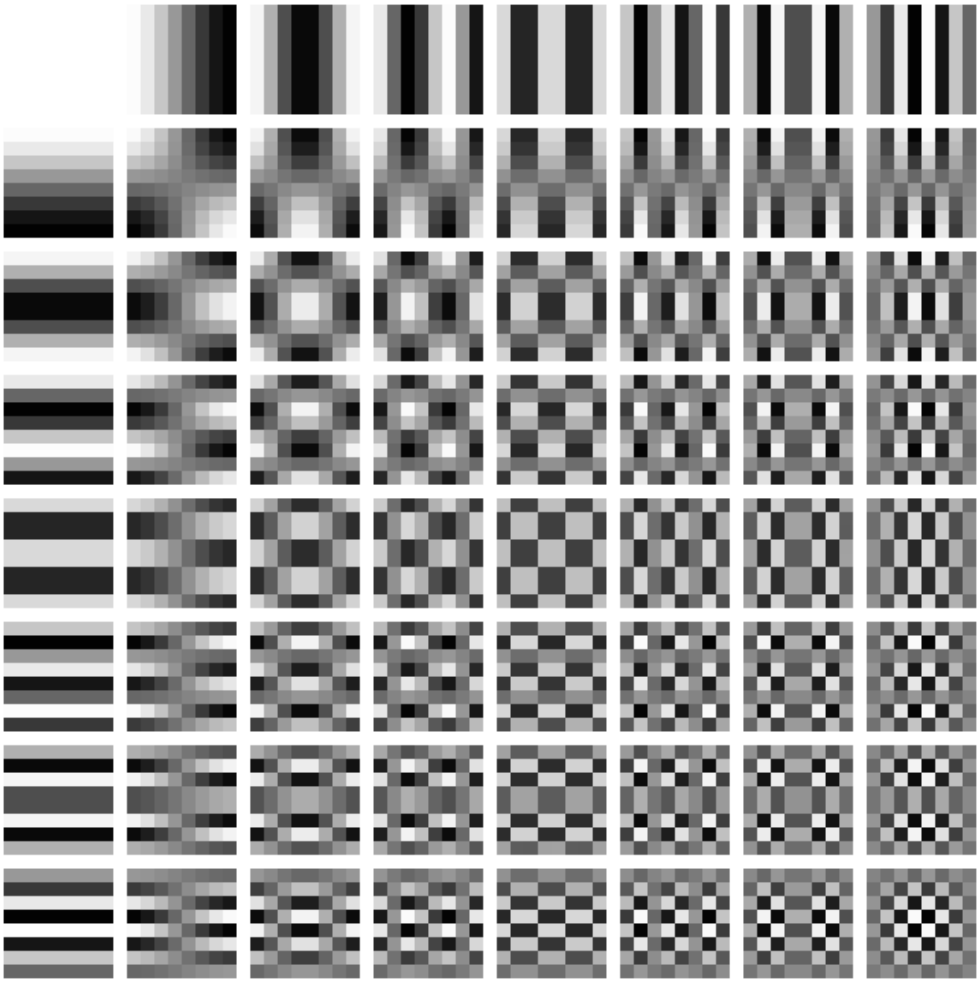


Figura 17.2: Rappresentazione grafica delle superfici DCT

come conseguenza della predominanza dei blocchi posti in corrispondenza ad aree dell'immagine quasi costanti, rispetto a quelli associati alla presenza di contorni netti e particolari dettagliati.

Quantizzazione Questo passo della elaborazione JPEG mira a sfruttare il fenomeno percettivo della ridotta sensibilità dell'occhio umano alle frequenze spaziali più elevate, ovvero la capacità di *filtrare percettivamente* le componenti di errore corrispondenti ai dettagli più minuti. Per questo, il processo di quantizzazione è orientato a ridurre, ed eventualmente sopprimere, le componenti di immagine legate alle frequenze spaziali più elevate, introducendo di fatto *una soglia* sotto la quale si stabilisce di non trasmettere quelle informazioni che tanto non sarebbero percepibili. A questo scopo, ogni coefficiente $D(i,j)$ viene diviso per un coefficiente $Q(i,j)$ dipendente da (i,j), ed il risultato viene arrotondato:

$$B(i,j) = \text{round}\left(\frac{D(i,j)}{Q(i,j)}\right)$$

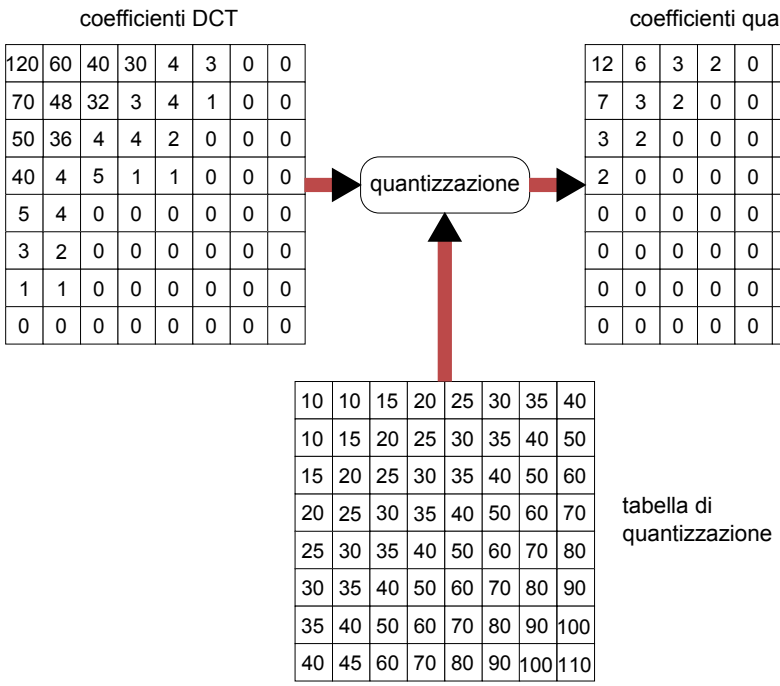


Figura 17.3: Processo di quantizzazione dei coefficienti DCT

Il risultato corrisponde ad un processo di quantizzazione, perché quando in ricezione il processo viene invertito (ri-moltiplicando il coefficiente per la stessa quantità), viene persa la precisione legata all'arrotondamento, e pari alla metà del coefficiente di divisione. La scelta dei $Q(i,j)$ è fatta in modo tale da utilizzare valori più elevati per gli indici (i,j) più elevati, in modo ottenere due risultati: ridurre le componenti ad alta variabilità dell'immagine, e poter usare meno bit per codificare questi valori (più piccoli). Inoltre, molti dei coefficienti con (i,j) elevato, già piccoli di per sè, quando divisi per un coefficiente di quantizzazione più elevato, non *sopravvivono* all'operazione di arrotondamento, in modo che tipicamente la parte in basso a destra della matrice $B(i,j)$ sarà tutta pari a zero, facilitando il compito della codifica run-lengh dello stadio successivo.

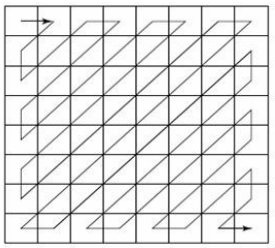
Esempio La figura 17.3 mostra un esempio di matrice di coefficienti DCT, assieme alla tabella di quantizzazione, ed al risultato dell'operazione. Notiamo come il valore dei coefficienti di quantizzazione aumenti allontanandosi dal coefficiente DC, e come nella matrice dei coefficienti quantizzati siano *sopravvissuti* solo i coefficienti relativi alle frequenze spaziali più basse.

Sebbene esistano delle tabelle di quantizzazione predefinite, i valori effettivi possono essere variati in base ad un compromesso tra qualità che si intende conseguire e fattore di compressione; tali valori vengono poi acclusi assieme al bitstream codificato durante la fase di formattazione, in modo che il processo di quantizzazione possa essere invertito in fase di riproduzione dell'immagine.

Codifica entropica Questo passo è un processo senza perdita, nel senso che non aggiunge altre distorsioni oltre a quelle introdotte dal passo di quantizzazione, ma è essenzia-

le ai fini della compressione, e sfrutta le caratteristiche statistiche del risultato delle elaborazioni precedenti. Come posto in evidenza nello schema di pag. 318, la codifica entropica adotta due diverse procedure per i coefficienti DC e AC, che in entrambi i casi culminano con uno stadio di codifica a lunghezza variabile mediante codici di Huffman.

Vettorizzazione Le matrici 8x8 relative ai blocchi di elaborazione visti fin qui vengono ora trasformate in sequenze lineari da un processo di scansione a *zig zag* delle stesse, come mostrato dalla figura seguente.



La sequenza così ottenuta presenta il coefficiente DC in testa, a cui fanno seguito i rimanenti 63 coefficienti AC, ordinati in base al massimo valore di frequenza spaziale che rappresentano. Se applichiamo la scansione zig-zag ai valori riportati nell'esempio precedente, otteniamo come risultato la sequenza

12 6 7 3 3 2 2 2 2 0 0 0 ..... 0 0

Codifica differenziale I blocchi adiacenti generalmente possiedono coefficienti DC molto simili tra loro, in virtù dell'omogeneità di ampie zone dell'immagine (pensiamo ad un porzione di cielo). Per questo motivo, anziché codificarli in modo indipendente, i singoli coefficienti DC di blocchi consecutivi vengono sottratti l'uno all'altro, e viene codificata solo la loro differenza. Ad esempio, se una sequenza di coefficienti DC risultasse pari a 12 13 11 11 10 ..., il risultato di questo processo di codifica differenziale darebbe luogo alla sequenza 12 1 −2 0 −1 ... (infatti, il valore *precedente* al primo coefficiente si assume pari a zero). Dato che differenze in valore assoluto piccole sono relativamente più frequenti di differenze grandi, si è scelto di adottare per queste una codifica a lunghezza di parola variabile, realizzata prima descrivendo ogni valore di differenza mediante la coppia (sss, valore), in cui sss rappresenta il numero di bit necessario per rappresentare il valore, e quindi concatenando una codeword di Huffman corrispondente ad sss, al codice lineare che rappresenta il valore.

Esempio Per chiarire le idee, mostriamo le corrispondenze citate mediante due tabelle, che poi applichiamo all'esempio precedente.

differenza	*N. di bit (sss)*	*valore codificato*	
0	0		
-1, 1	1	1=1	-1=0
-3, -2, 2, 3	2	2=10	-2=01
		3=11	-3=00
-7...-4, 4...7	3	4=100	-4=011
		5=101	-5=010
		6=110	-6=001
		7=111	-7=000
-15...-8, 8...15	4	8=1000,	-8=0111
⋮		⋮	

sss	codeword di Huffman
0	010
1	011
2	100
3	00
4	101
5	110
6	1110
7	11110
⋮	⋮
11	111111110

Tornando dunque al nostro esempio della sequenza differenziale $12\ \ 1\ \ -2\ \ 0\ \ -1\ \ldots$, in termini di coppie (sss, valore) questa diviene (4, 12), (1, 1), (2, -2), (0, 0), (1, -1),... e quindi, sostituendo ad sss il relativo codice di Huffman preso dalla seconda colonna della seconda tabella, ed ai valori la loro rappresentazione indicata dalla terza colonna della prima tabella, otteniamo la sequenza di bit 101 1100, 011 1, 100 01, 010, 011 0,... in cui si sono mantenute le virgole per chiarezza. In definitiva, abbiamo usato un totale di 23 bit per rappresentare 5 differenze, che ne avrebbero richiesti 45 se codificate con 9 bit.

Codifica run-lenght Viene applicata alla sequenza di coefficienti AC che è il risultato dello *zig-zag scan*. In base all'effetto congiunto delle caratteristiche dei coefficienti della DCT, e del processo di quantizzazione, la sequenza degli AC in uscita dal vettorizzatore presenta lunghe sequenze di zeri, consentendo di conseguire buoni rapporti compressione mediante l'uso di una codifica *run-lenght*, realizzata scrivendo gli AC come una sequenza di coppie *(skip,* ACN*)*, in cui *skip* rappresenta il numero di zeri nel run, e ACN è il coefficiente AC non nullo che viene dopo la sequenza di zeri. Quindi, il campo ACN viene espresso a sua volta nella forma *sss, valore*, come indicato dalla prima tabella riportata nell'ultimo esempio. Infine, la coppia *skip, sss* viene rappresentata con una codeword di Huffman individividuata in un nuovo codebook appositamente definito.

Esempio Applicando la codifica run-length alla sequenza dei coefficienti AC individuati nell'esempio di vettorizzazione, ossia alla sequenza 12 6 7 3 3 2 2 2 2 0 0 0 ..... 0 0, porta ad una sequenza di coppie *(skip,* ACN*)*, pari a *(0,6), (0,7), (0,3), (0,3), (0,3), (0,2), (0,2), (0,2), (0,2) (0,0)* in cui l'ultima coppia *(0,0)* indica la fine del blocco, che in fase di decodifica viene quindi ricostruito riempiendolo di zeri. Viceversa, una sequenza di coppie *(skip,* ACN*)*, pari a *(0,6), (0,7), (3,3), (0,-1), (0,0)* corrisponde ad una sequenza di coefficienti AC 6 7 0 0 0 3 -1 0 0 ..... 0. Sostituendo ai termini ACN dell'ultima sequenza di coppie *(skip,* ACN*)*, la coppia *sss, valore*, e codificando quindi il termine *valore* come indicato nella prima tabella dell'esempio precedente, si ottiene *(0, 3, 110), (0, 3, 111), (3, 2, 11), (0, 1, 0), (0,0)*. Il bitstream finale viene quindi realizzato sostituendo alle attuali coppie *skip, sss*, le rispettive codeword individuate alla colonna *Run/Size* della tabella a pagina 150 e segg. delle specifiche ITU-T T.81 `http://www.digicamsoft.com/itu/itu-t81-154.html`, ottenendo *(100, 110), (100, 111), (111110111, 11), (00, 0), (1010)*, e producendo così un totale di 30 bit per rappresentare i 63 coefficienti AC.

Formattazione Lo standard JPEG definisce, oltre alla sequenza di operazioni indicata, anche il formato di trama con il quale devono essere memorizzato il bitstream finale. La struttura risultante è gerarchica, e mostrata alla figura 17.4 . Al livello superiore troviamo un *frame header* che contiene le dimensioni complessive dell'immagine, il numero ed il tipo di componenti usate (CLUT, RGB, $\mathrm{YC}_b\mathrm{C}_r$, etc), ed il formato di campionamento (4:2:2, 4:2:0, etc.). Al secondo livello, troviamo uno o più *Scan*, ognuno preceduto da una intestazione in cui viene riportata l'identità del componente (R, G, B, o y, C_b, C_r), il numero di bit usato per rappresentare ogni coefficiente di DCT, e la tabella di quantizzazione usata per quella componente. Ogni Scan è composta da uno o più *segmenti*, preceduti da un'ulteriore intestazione, che contiene il codebook di Huffman usato per rappresentare i valori dei blocchi del segmento, nel caso non siano stati usati quelli standard. Infine, nel segmento trovano posto le sequenze di blocchi dell'immagine, così come risultano dopo lo stadio di codifica entropica.

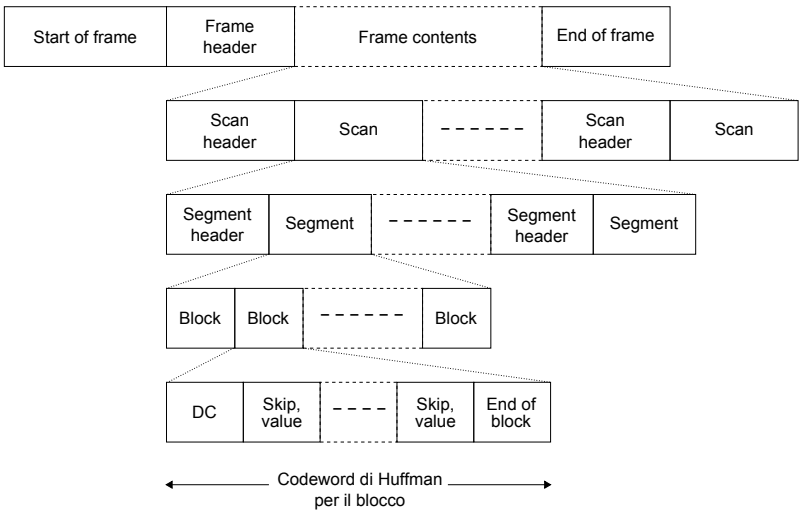


Figura 17.4: Formato del bitstream per la codifica JPEG

17.1.4 Codifica audio

17.1.5 Codifica video

17.1.5.1 Standard video

17.2 Codifica di canale

Molto probabilmente, questa sezione la scriverò un altro anno. Forse.