UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"

CORSO DI LAUREA IN INGEGNERIA DELLE
TELECOMUNICAZIONI

OTTIMIZZAZIONE DI TECNICHE A BASSA COMPLESSITÀ PER LA CODIFICA DI SEQUENZE VIDEO

Relatore
ch.mo prof. Giovanni Poggi
Correlatore
ing. Luisa Verdoliva

Candidato
Marco Cagnazzo

Anno accademico 2000–2001

Ringraziamenti

Durante la preparazione di questo lavoro di tesi ho avuto la fortuna di collaborare con persone che con il loro elevato spessore professionale e umano hanno reso questa esperienza proficua e stimolante. Desidero perciò qui ringraziare il mio relatore, il professor Poggi, che mi ha guidato con la competenza e la disponibiltà che lo contraddistinguono; il mio correlatore, l'ingegner Verdoliva, che mi ha seguito ed aiutato con preziosi consigli; nonché il responsabile del laboratorio, il dottor De Luca.

Alla fine degli studi universitari, il pensiero va agli amici con i quali ho affrontato questo lungo ed impegnativo percorso: grazie a loro è stato più facile superare le difficoltà e più bello festeggiare i successi.

Ma tutto questo non sarebbe mai stato possibile senza la mia famiglia, che mi ha dato il sostegno e la serenità per affrontare gli studi nel modo migliore che si possa desiderare. Grazie.

Indice

Introduzione

La possibilità di trasferire informazioni multimediali è una delle caratteristiche più interessanti del *networking*, sia dal punto di vista tecnico, dove si prefigura l'impegnativa sfida di fornire servizi avanzati e interattivi ad un grande numero di utenti, sia dal punto di vista economico, dove altrettanto chiaramente si prevede lo sviluppo di un nuovo e vasto mercato [1],[2].

Spesso le applicazioni multimediali sono caratterizzate dall'esistenza di un unico trasmettitore a fronte di un numero elevato di ricevitori. Si pensi ad esempio alla trasmissione di notiziari, di lezioni o di spettacoli. In questi casi, nel luogo dal quale si trasmette, il flusso audio/video analogico viene digitalizzato, cioè trasformato in un flusso di bit, il quale è poi compresso e frammentato in piccoli blocchi, detti *pacchetti*, prima di essere trasmesso. Esistono diversi modi per distribuire i pacchetti agli utenti interessati: ad esempio, le regole dei protocolli di trasmissione di tipo *unicast*, prevedono la trasmissione di una singola copia di ogni pacchetto ad ogni utente, con un evidente spreco di risorse, dal momento che la rete sarebbe inondata da un grande numero di pacchetti tutti uguali. L'alternativa alla trasmissione di tipo unicast è quella cosiddetta *multicast* [3]. Nel multicast uno stesso pacchetto viene indirizzato ad un sottoinsieme di tutte le macchine presenti in una rete, senza però farne un numero di copie superiore a quello strettamente necessario. A tal fine vengono definiti dei *gruppi*, costituiti da tutte e sole le macchine interessate alla ricezione di un particolare messaggio. Pertanto i pacchetti multicast hanno come indirizzo del destinatario non quello di una singola macchina, ma un identificativo del gruppo. Questa tecnica, tra le altre possibili applicazioni, sembra particolarmente

adatta alla trasmissione efficiente di contenuti multimediali: i pacchetti che trasportano l'audio ed il video vengono trasmessi solo ad un particolare gruppo multicast[1]. Sebbene più efficiente in banda rispetto all'unicast, questa tecnica di trasmissione, essendo caratterizzata da un tasso di trasmissione uniforme, può risultare poco adeguata se, ad esempio, la trasmissione avviene tramite l'Internet[2]. In questo caso, i vari utenti hanno accessi a tassi molto diversi tra loro: si pensi agli accessi domestici a bassa o a media velocità, oppure a quelli ad alta velocità attraverso i backbone. Se il tasso di trasmissione della sorgente supera la capacità di uno qualsiasi dei link d'accesso degli utenti, nella rete si verifica una congestione: i pacchetti si cominciano a perdere frequentemente e la qualità del flusso ricevuto peggiora rapidamente. Il problema è che un singolo flusso, a tasso costante, non è in grado di soddisfare i requisiti conflittuali di un insieme eterogeneo di utenti.

Inoltre gli utenti differiscono tra loro anche per quanto riguarda un altra risorsa: la potenza di calcolo di cui dispongono sulla loro macchina. Se l'algoritmo di codifica/decodifica è molto complesso, difficilmente gli utenti che dispongono di computer meno potenti saranno in grado di usufruire del servizio in *tempo reale*.

Di fronte a questo quadro, il CNIT[3], con il *progetto Labnet* si è posto l'obiettivo di realizzare un "sistema integrato di apprendimento" che consenta l'accesso da remoto, da parte di studenti e ricercatori, a laboratori e strutture altamente complesse e costose, tramite un'opportuna infrastruttura di rete e un insieme di tecnologie per supportare le applicazioni multimediali. In particolare, si è preso in esame il problema della trasmissione e ricezione di flussi multimediali su di una rete eterogenea in modo efficiente, facendo sì, cioè, che ogni utente possa ricevere un flusso di qualità proporzionale alle risorse di cui dispone, in termini sia di banda, sia di potenza di calcolo. Per soddisfare questi requisiti, l'algoritmo di codifica da

[1] Il concerto dei Rolling Stones del 18 novembre 1994 fu il primo evento di massa trasmesso su Internet, e ciò fu possibile grazie alla dorsale di supporto del multicast, o *Multicast Backbone*.

[2] L'iniziale maiuscola indica la "rete mondiale". Con "internet" invece si intende un qualsiasi insieme di reti.

[3] Consorzio Nazionale Interuniversitario per le Telecomunicazioni.

usare deve essere *scalabile* e *a bassa complessità*.

La bassa complessità è un requisito che richiede pochi commenti: è chiaro infatti che per poter funzionare in tempo reale ed anche su macchine non particolarmente potenti, l'algoritmo deve essere computazionalmente economico.

L'uso di un algoritmo scalabile (o stratificato) è un approccio comune per affrontare il problema di un insieme eterogeneo di utenti che ricevono flussi multimediali [4],[5]. Invece di distribuire un singolo flusso di qualità elevata attraverso un unico canale sulla rete, la sorgente genera un flusso base caratterizzato da parametri di qualità minimi, e tanti altri sottoflussi, ognuno dei quali permette di migliorare la qualità complessiva. Ogni flusso è indirizzato ad un diverso gruppo multicast, in modo che un utente, cambiando il numero di gruppi a cui aderisce, può modificare le caratteristiche del flusso ricevuto, adattandole alla quantità di risorse di cui dispone.

Il lavoro svolto in questa tesi si pone in una fase del progetto Labnet nella quale è stato già sviluppato un codificatore-decodificatore (*codec*) video che soddisfa i requisiti di bassa complessità e di scalabilità [6], utilizzando la quantizzazione vettoriale gerarchica tabellare (HVQ), la codifica piramidale, ed un codebook strutturato ad albero (TSVQ) [5], [7], [8]. L'obiettivo del presente lavoro è stato, in prima battuta, quello di analizzare approfonditamente i prototipi già realizzati, allo scopo di delinearne con precisione le prestazioni e di individuare i valori ottimali di una serie di parametri di codifica. La seconda e più impegnativa fase del lavoro è stata quella di apportare un insieme di modifiche (anche strutturali) al codec per migliorarne le prestazioni.

Le problematiche appena illustrate sono state affrontate durante lo svolgimento del lavoro di tesi, e vengono riprese nei successivi capitoli, la cui organizzazione è qui brevemente descritta.

- Nel primo capitolo si descrivono le caratteristiche del segnale video e le relative tecniche di codifica, nonché i principali standard esistenti.

- Nel secondo si parla, invece, dei primi prototipi realizzati e di come questi riescano ad implementare una tecnica di codifica

scalabile ed a bassa complessità, tramite l'utilizzo di HVQ e TSVQ. Il capitolo si chiude con l'analisi delle prestazioni dei prototipi al variare di un insieme di parametri d'interesse.

- Infine, nel terzo capitolo, si illustrano le tecniche usate per migliorare le prestazioni del codec, e si forniscono e discutono i risultati.

Capitolo 1

Il segnale video e la sua codifica

1.1 Il segnale video

Una sequenza video consiste in una successione di immagini fisse, dette fotogrammi o *frame*, che si susseguono in modo da dare ad un osservatore l'illusione della continuità. Infatti, quando un'immagine è proiettata sulla retina, essa continua ad essere percepita per alcuni millisecondi prima di sparire[1], per cui, se si proiettano almeno 50 immagini al secondo, l'occhio non si accorge del fatto che sta osservando una successione discreta di immagini invece che una sequenza continua. Tutti i sistemi video sfruttano questo principio per riprodurre immagini in movimento. Quando le immagini sono proiettate con una frequenza inferiore ai 50 Hz, un osservatore umano percepisce il calo di luminosità tra una frame e la successiva; questo fenomeno, detto *flicker*, è molto fastidioso, e va senz'altro evitato. Per quanto riguarda la riproduzione del movimento, un tasso di 25 frame al secondo è sufficiente per dare la sensazione di un movimento fluido; per evitare il flicker senza dover aumentare il frame rate, è sufficiente proiettare più volte la stessa frame.

[1] Questa proprietà è nota col nome di *persistenza*.

1.1.1 Sistemi analogici

Il segnale video viene riprodotto tramite un tubo a raggi catodici, nel quale un flusso di elettroni, generato da un cannone elettronico, è utilizzato per eccitare i fosfòri di uno schermo. L'intensità della radiazione luminosa emessa da un fosfòro è proporzionale alla corrente del fascio stesso, la quale è a sua volta controllata dalla tensione applicata agli elettrodi del cannone elettronico. Per rappresentare un'immagine monocromatica bidimensionale tramite un segnale monodimensionale che varia in funzione del tempo, si effettua una scansione dell'immagine per righe, nel corso della quale, l'ampiezza del segnale video è proporzionale alla luminosità del punto corrente. Gli esatti parametri della scansione variano da paese a paese. Ad esempio, in America ed in Giappone si usano 525 linee di scansione, un rapporto base:altezza della frame (rapporto d'aspetto) di 4:3, e 30 frame al secondo. In Europa invece si usano 625 linee, lo stesso rapporto d'aspetto, e 25 frame al secondo.

Una sequenza video a colori usa lo stesso schema di scansione, ma con tre pennelli elettronici, ognuno dei quali serve per generare uno dei tre colori primari: rosso, verde e blu (cfr. [9, pag. 48-57]). Questa tecnica funziona perché ogni colore può essere ricostruito come sovrapposizione dei tre primari con opportune intensità. Il modo più semplice per rappresentare i colori è quello di assegnare un canale a ciascun colore primario, tuttavia, in questo modo, non è possibile ricevere i programmi trasmessi a colori su apparecchi monocromatici. Per soddisfare questa specifica, i sistemi di televisione a colori prevedono di combinare linearmente i tre segnali relativi alle intensità dei colori primari, in modo da ottenere un segnale di *luminanza* e due di *crominanza*. Il primo segnale è equivalente al segnale video monocromatico; gli altri due, invece, forniscono l'informazione supplementare relativa al colore. La relazione tra i colori fondamentali e il segnale di luminanza dipende dalla diversa sensibilità dell'occhio umano al rosso, al verde e al blu. Detto Y il segnale di luminanza e R, G, B i segnali di intensità del rosso, verde e blu

rispettivamente, si ha:

$$Y = 0.30R + 0.59G + 0.11B \tag{1.1}$$

Esistono invece diversi modi per definire i segnali di crominanza; ad esempio, negli Stati Uniti, lo standard NTSC[2] utilizza come segnali di crominanza i segnali denominati I e Q definiti come:

$$\begin{cases} I = 0.60R - 0.28G - 0.32B \\ Q = 0.21R - 0.52G + 0.31B \end{cases} \tag{1.2}$$

La larghezza di banda dei segnali I e Q è ottimizzata rispetta alla sensibilità dell'occhio umano. I test eseguiti in occasione della definizione dello standard hanno mostrato che per il segnale I è sufficiente una larghezza di banda di $1,5$ MHz, mentre per il segnale Q sono sufficienti 0.5 MHz. Invece il segnale di luminanza occupa circa 5 MHz.

In Europa si usa invece il sistema PAL[3], che utilizza come segnali di crominanza i segnali denominati U e V definiti come:

$$\begin{cases} U = 0.345R - 0.2908G - 0.0542B \\ V = 0.2631R - 0.5174G + 0.7805B \end{cases} \tag{1.3}$$

Anche in questo caso la banda dei segnali di crominanza è dell'ordine del Megahertz.

È interessante notare che, indipendentemente da come vengono definiti i segnali di crominanza, l'occhio umano è molto più sensibile alla luminanza che non alle informazioni di colore, per cui la crominanza può anche essere trasmessa meno accuratamente, senza una percepibile perdita di qualità. Di conseguenza, il segnale di luminanza può essere trasmesso nello stesso canale di un segnale monocromatico, e può essere ricevuto da un apparecchio in bianco e nero. I segnali di crominanza, in entrambi i sistemi europeo ed americano, vengono usati per modulare in quadratura un'apposita portante (detta sottoportante di colore). Il valore della frequenza di tale sottoportante è scelto in modo da minimizzare l'interferenza del

[2] National Television Standards Committee
[3] Phase Alternating Line

segnale di crominanza con quello di luminanza. Inoltre, nel sistema europeo, la sottoportante di colore subisce un inversione di fase per ogni riga[4], il che rende il sistema meno sensibile agli errori di fase.

1.1.2 Sistemi digitali

I sistemi video analogico costituiscono una necessaria premessa ai sistemi digitali, ma, d'ora in avanti, ci occuperemo esclusivamente di questi ultimi. La più semplice rappresentazione di un segnale video digitale consiste in una sequenza di frame, ognuna delle quali è una matrice di "elementi di immagine" detti anche *pixel* (da *picture element*). Possiamo pensare di usare 8 bit per pixel, in modo da rappresentare 256 livelli di grigio, dando luogo ad un segnale video monocromatico di elevata qualità. Nel caso di segnale video a colori, ogni colore primario è rappresentato usando 8 bit (così da avere 24 bit per pixel). In questo modo è possibile rappresentare fino ad oltre 16 milioni di colori, che sono in realtà più di quanti l'occhio umano riesca a distinguere. La geometria è la stessa del caso analogico, solo che le linee di scansione sono sostituite da righe di pixel discreti.

Per riprodurre il movimento è ancora necessario proiettare almeno 25 frame al secondo. Poiché un monitor di buona qualità ridisegna le immagini a 75 Hz, è sufficiente rappresentare tre volte la stessa frame per eliminare il flicker. In altre parole, la fluidità del movimento è determinata dal numero di immagini *differenti* al secondo, mentre il flicker è determinato dal numero di volte che l'immagine è riproiettata sullo schermo. La diversa importanza di questi due parametri è chiarita quando si considera la banda necessaria per la trasmissione di un segnale video digitale. Supponiamo di avere una rappresentazione con 24 bit per pixel, 1024×768 pixel per frame, e 25 frame al secondo, si arriva ad un tasso r pari a 472 milioni di bit per secondo (Mbps[5]). Raddoppiare un tasso già così alto non è certo la miglior soluzione per eliminare il flicker: è sicu-

[4] Da questo deriva il nome dello standard.

[5] Nell'ambito del networking e della teoria dell'informazione, i prefissi kilo, Mega,... nelle misure di *tasso*, indicano le potenze di 10 e non (come nell'informatica) le potenze di 2: 1Mbps è pari a 10^6, e non a $1\,048\,576$ (2^{20}) bit al secondo.

ramente più opportuno trasmettere 25 frame al secondo e far sì che il computer memorizzi ogni immagine e la mostri due volte. Si noti che questa soluzione non è applicabile alla trasmissione del segnale televisivo analogico (gli apparecchi non hanno la possibilità di memorizzare le frame), per la quale sono usate altre tecniche, come la scansione interallacciata.

1.1.3 La ridondanza del segnale video

Dalle considerazioni precedenti scaturisce che la riduzione della banda richiesta è una condizione necessaria per la trasmissione di segnali video digitali tramite una rete di calcolatori. A questo fine è possibile sfruttare la forte ridondanza del segnale video, che si manifesta come ridondanza statistica e come ridondanza psicofisica.

La ridondanza psicofisica del segnale video consiste nel fatto che esso trasporta anche informazioni che un osservatore umano non è in grado di apprezzare, e che perciò possono essere eliminate. In particolare l'occhio umano è meno sensibile a errori sulla luminanza in corrispondenza di brusche discontinuità, sia spaziali (contorni degli oggetti), sia temporali (cambiamenti di scena), o, detto in altre parole, non riesce ad apprezzare le alte frequenze spaziali e temporali del segnale video. Questo significa che è possibile trasmettere meno accuratamente il segnale in corrispondenza di tali discontinuità senza un degrado percepibile della qualità.

La ridondanza statistica consiste invece nel fatto che i pixel che costituiscono le immagini di una sequenza video dipendono statisticamente da quelli vicini spazialmente e da quelli nella stessa area in frame diverse. Si parla quindi di ridondanza spaziale e ridondanza temporale. Tutti i principali algoritmi di codifica video sfruttano entrambi i tipi di ridondanza per ottenere la compressione dei dati.

1.2 Il problema della compressione

Il problema della compressione dati è di portata del tutto generale. Gli algoritmi di compressione possono essere classificati in algoritmi senza perdite (o *lossless*) e con perdite (o *lossy*), a seconda

che sia o meno possibile ricostruire perfettamente i dati iniziali partendo dalla versione compressa. In generale un algoritmo di compressione *lossless* si basa esclusivamente sulle proprietà statistiche dei dati d'ingresso, e permette di ottenere solo bassi fattori di compressione. Gli algoritmi *lossy*, invece, a fronte di una perdita di parte delle informazioni originarie, garantiscono una maggiore efficienza di compressione. Tale perdita di informazione è accettabile quando sia nota la semantica dei dati in ingresso, cioè quando sia possibile prevedere a priori che l'informazione persa è in realtà poco significativa. Queste caratteristiche rendono le tecniche *lossy* particolarmente adatte alla codifica di segnali video (e audio), nei quali esistono molte informazioni non percettibili da parte dell'utente finale. Una delle caratteristiche più interessanti degli algoritmi di compressione *lossy* è allora proprio il modo in cui essi sono in grado di identificare le informazioni utili e separarle da quelle superflue.

Un'altra classificazione prevede di distinguere gli algoritmi di compressione in *simmetrici* e *asimmetrici*, a seconda che la complessità e la quantità di risorse necessarie per l'operazione di codifica siano rispettivamente confrontabili o molto maggiori che per quella di decodifica. Gli algoritmi asimmetrici sono tipicamente in grado di fornire rapporti di compressione maggiori rispetto agli algoritmi simmetrici e sono adatti per applicazioni come la distribuzione su grande scala di prodotti multimediali, ma non sono adeguati per altri tipi di applicazioni, in particolare per quelle che hanno esigenze di funzionare in tempo reale, perché sono caratterizzati da un'algoritmo di codifica tipicamente molto oneroso. In questi casi si può ricorrere con successo agli algoritmi simmetrici, che in genere sono più semplici computazionalmente, ma hanno prestazioni inferiori.

1.2.1 Tecniche di compressione lossless

Le tecniche di compressione senza perdite affondano le loro radici nei principi della teoria dell'informazione [10]. Dal punto di vista della teoria dell'informazione un segnale video può essere visto come una sequenza di simboli appartenenti ad un alfabeto di cardinalità $N = 2^{n_b}$, dove n_b indica il numero di bit per pixel con cui è codificata

l'immagine. Si indichi allora con e_i l'evento corrispondente all'emissione dell' i-esimo simbolo, e con p_i la probabilità di tale evento. Si definisce l'*autoinformazione* associata all'evento e_i come:

$$I(e_i) \triangleq \log \frac{1}{p_i} \tag{1.4}$$

A volte l'autoinformazione è detta più semplicemente *informazione*. La base del logaritmo è determinata dal numero di stati usati per rappresentare la sorgente d'informazione. Normalmente si usa la base 2: in tal modo l'autoinformazione è espressa in numero di bit per simbolo. Notiamo che, con questa definizione, l'autoinformazione di un evento è tanto maggiore quanto più esso è improbabile, il che si accorda con il significato ordinario del termine informazione: infatti il verificarsi di un evento certo ha un contenuto informativo nullo; al contrario, il verificarsi di un evento altamente improbabile porta con sé molta informazione. Dal concetto di autoinformazione discende quello di *entropia* di un simbolo di sorgente definita come il valor medio dell'autoinformazione. Indicata con X la variabile aleatoria associata al generico simbolo emesso dalla sorgente S (assunta stazionaria per ipotesi), si ha:

$$\begin{aligned} H(X) &\triangleq \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i} \\ &= -\sum_{i=1}^{N} p_i \log_2 p_i \end{aligned} \tag{1.5}$$

in cui $H(X)$ è espressa in bit/simbolo. La (1.5) può essere generalizzata se, invece di considerare la probabilità di emissione di un singolo simbolo, si passa a considerare *blocchi* di simboli. Detto $\mathcal{X}$ l'insieme dei possibili valori che X può assumere e detta $X^{(n)} = (X_1, X_2, ..., X_n)$ una sequenza di n simboli di sorgente, si definisce *entropia di ordine* n dei simboli di sorgente, la quantità:

$$H(X^{(n)}) = \sum_{x^{(n)} \in \mathcal{X}^n} -p(x^{(n)}) \log_2 p(x^{(n)}) \tag{1.6}$$

dove $p(x^{(n)})$ è la distribuzione di probabilità di $X^{(n)}$. Dal concetto di entropia si passa poi a quello di *tasso entropico*, definito per sorgenti stazionarie, come:

$$H(S) = \lim_{n\to\infty} \frac{1}{n} H(X^{(n)}) \tag{1.7}$$

Mentre $H(X)$ rappresenta il contenuto informativo di un singolo simbolo, il tasso entropico è il contenuto informativo medio per simbolo di una sequenza lunga a piacere. Intuitivamente, l'entropia di una sorgente sarà tanto minore quanto più prevedibile è il suo comportamento. Più precisamente, il tasso entropico può essere interpretato come numero medio di bit per simbolo necessario per rappresentare il contenuto informativo della sorgente. In effetti il primo teorema di codifica di sorgente, dovuto a Shannon, afferma proprio che un segnale può essere codificato con un numero di bit per simbolo arbitrariamente vicino al tasso entropico della sorgente, purché i simboli vengano raggruppati in blocchi sufficientemente lunghi. Le tecniche di codifica cosiddette *entropiche* sono quelle in cui si cerca di individuare un codice le cui prestazioni si avvicinino al limite teorico. Esempi di tecniche di codifica entropica sono la codifica di Huffman e la codifica aritmetica.

I parametri in base ai quali si valuta una tecnica di compressione lossless sono il *tasso di codifica*, pari al numero di bit necessario a rappresentare un simbolo d'ingresso, e la *complessità* della tecnica. Si introduce anche il *rapporto di compressione*, cioè il rapporto tra il numero di bit necessario per rappresentare un simbolo d'ingresso e quello necessario a rappresentare un simbolo d'uscita.

1.2.2 La codifica di Huffman

La codifica di Huffman [10] è una tecnica di codifica a *lunghezza variabile*, perché i vari simboli sono codificati con stringhe di bit di diverse lunghezze. In particolare si assegnano ai simboli più probabili delle *parole codice* più brevi, in modo da minimizzare la lunghezza media del codice, cioè il numero medio di bit di codifica per simbolo. Per illustrare questo metodo, consideriamo un'esempio. Nella tabella 1.1 sono mostrate le statistiche del primo ordine

di una certa sorgente S, caratterizzata da un alfabeto $\mathcal{A}$ costituito da sei simboli.

Simbolo	Probabilità
A	0.4
B	0.2
C	0.15
D	0.15
E	0.05
F	0.05

Tabella 1.1: Esempio della codifica di Huffman: alfabeto d'ingresso

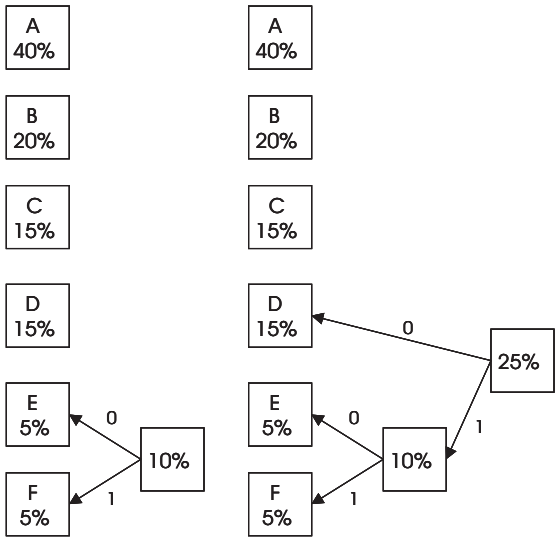

Figura 1.1: Esempio per la codifica di Huffman: primi stadi della costruzione del'albero

L'entropia dell'alfabeto d'ingresso si valuta con la (1.5). Si ottiene:

$$\begin{aligned} H(S) &= -0.4\log_2 0.4 - 0.2\log_2 0.2 - 0.15\log_2 0.15 - 0.15\log_2 0.15 + \\ &\quad -0.05\log_2 0.05 - 0.05\log_2 0.05 \cong 2.25 \text{ bit/simbolo} \end{aligned}$$

Il primo passo della codifica di Huffman consiste nella creazione

di un albero a cui risulterà associato il codice. Ogni simbolo $s_i \in \mathcal{A}$, insieme alla sua probabilità, è associato ad un nodo di un albero binario. Si individuano i due nodi a probabilità minore, siano quelli associati ai simboli s_a e s_b, e si genera un terzo nodo (indichiamolo con s_{ab}) di cui i nodi relativi a s_a e s_b sono i figli, e a cui si associa una probabilità $p_{ab} = p_a + p_b$ (vedere la figura 1.1). Dopodiché il procedimento si itera, considerando però un nuovo insieme di nodi, dato da $\mathcal{A} \cup \{s_{ab}\} - \{s_a, s_b\}$. L'algoritmo termina quando tutti simboli vengono associati ai nodi dell'albero (vedere la figura 1.2).

A questo punto, per individuare la stringa di bit con cui è codificato ogni simbolo, basta seguire il percorso che sull'albero va dalla radice verso il nodo associato la simbolo, ed aggiungere alla stringa uno 0 quando si va in una direzione ed un 1 quando si va nell'altra. Nella tabella 1.2 è illustrato il codice di Huffman per il nostro esempio.

Si riscontra immediatamente che effettivamente le parole codice più brevi sono assegnate ai simboli più probabili. Valutiamo adesso la lunghezza media del codice. Si ha:

$$\overline{L} \triangleq \sum_{i=1}^{N} p_i l_i \tag{1.8}$$

dove con l_i si intende la lunghezza della parola codice associata all'i-esimo simbolo. Si ottiene in questo caso:

$$\overline{L} = 0.4 \cdot 1 + 0.2 \cdot 3 + 0.15 \cdot 3 + 0.15 \cdot 3 + 0.05 \cdot 4 + 0.05 \cdot 4 = 2.3 \text{ bit/simbolo}$$

Come si vede il valore ottenuto per $\overline{L}$ è prossimo al limite teorico dato dall'entropia.

Si può inoltre dimostrare che nessun codice univocamente decifrabile permette di ottenere una lunghezza media inferiore a quella del codice di Huffman, che quindi si presenta come la soluzione ottima per i problemi di codifica *lossless*. Tuttavia, è possibile realizzare la codifica di Huffman solo se si conosce esattamente la statistica dell'alfabeto d'ingresso, e in molti casi pratici, ci si deve accontentare di una stima di tali statistiche. A tale proposito insorgono due ordini di problemi. Il primo è che il segnale da comprimere può

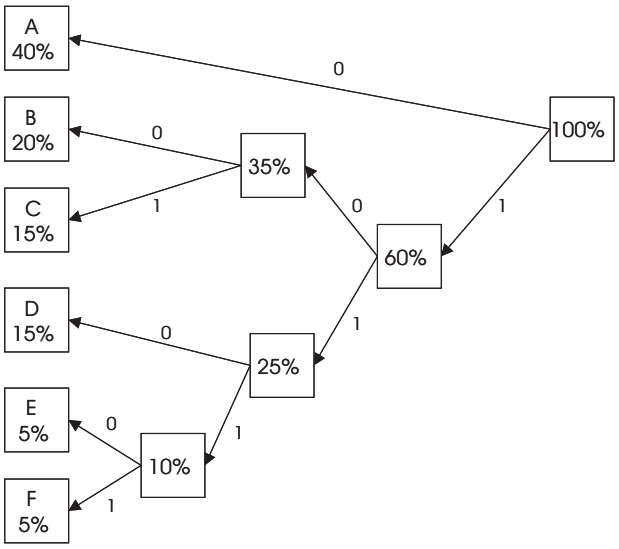

Figura 1.2: Esempio per la codifica di Huffman: l'albero completo

Simbolo	Probabilità	Parola codice
A	0.4	0
B	0.2	100
C	0.15	101
D	0.15	110
E	0.05	1110
F	0.05	1111

Tabella 1.2: Codice di Huffman per l'esempio di tab. 1.1

essere non stazionario cioè le caratteristiche statistiche variano nel tempo, rendendo poco accurata la stima. Il secondo è legato alla dimensionalità degli alfabeti di ingresso: questa può essere talmente grande che solo la fase di raccolta dei dati per la stima delle statistiche può risultare estremamente costosa in termini di memoria. D'altra parte non è raro avere la necessità di usare alfabeti d'ingresso molo grandi. Il punto è che le prestazioni di una tecnica di codifica sono tanto migliori quanto più si riescono a raggruppare i simboli emessi dalla sorgente in blocchi *lunghi*, che costituiscono poi i nuovi simboli della sorgente: solo così infatti, la lunghezza media del codice può avvicinarsi al tasso entropico. Ad esempio,

si consideri una sorgente binaria, di cui ogni simbolo è codificato separatamente: bisogna allora usare un bit per simbolo, indipendentemente dall'entropia della sorgente. Il problema della codifica di Huffman è che la sua complessità cresce esponenzialmente con la lunghezza dei blocchi, e allora, nei casi in cui è necessario usare blocchi di simboli molto lunghi può essere praticamente impossibile utilizzare questa tecnica.

1.2.3 Codifica aritmetica

Anche se la codifica di Huffman è teoricamente la migliore, non sempre è possibile o conveniente usarla. In particolare, per generare il codice di Huffman per blocchi di simboli di lunghezza n, è necessario calcolare la probabilità di ogni possibile blocco e costruire l'albero corrispondente; se si vuole passare a blocchi di lunghezza $n+1$, anche ammesso di poter facilmente ricavare le nuove statistiche, bisogna ripetere tutta la procedura. Uno schema migliore dovrebbe essere facilmente esteso a blocchi via via più lunghi, senza dover fare di nuovo tutti i calcoli. La codifica aritmetica raggiunge questo obiettivo. L'idea di base è quella di associare biunivocamente, ad ogni possibile blocco da codificare, un intervallo in $[0, 1]$ di ampiezza pari alla probabilità del blocco, e di trasmettere il numero corrispondente al centro dell'intervallo con una precisione sufficiente ad individuare univocamente l'intervallo in decodifica. Si può allora dimostrare che, una volta individuato il centro dell'intervallo corrispondente ad un blocco di lunghezza n, per codificare un ulteriore simbolo (arrivando ad un blocco di lunghezza $n+1$), è sufficiente fare 2 moltiplicazioni e 2 addizioni. In altre parole, la complessità dell'algoritmo è *lineare* con la lunghezza del blocco.

1.2.4 Codifica di Lempel-Ziv-Welch

E' una tecnica di codifica basata su dizionario. Si effettua una scansione dei simboli d'ingresso che vengono raggruppati in stringhe: se la stringa corrente è presente nel dizionario, essa viene codificata con l'indice corrispondente. Il dizionario viene costruito nel corso della codifica (e ricostruito in decodifica) in modo da adattarsi alle

statistiche del segnale, che quindi non devono essere note a priori. Per questo motivo l'algoritmo di Lempel-Ziv-Welch è un algoritmo *universale*.

1.2.5 Codifica Run Length

In un'immagine digitalizzata spesso i dati possono contenere valori che si ripetono un certo numero di volte: si pensi a delle aree di colore uniforme come ad esempio le regioni interne (lontane dai bordi) di un oggetto. È possibile comprimere dati di questo tipo se si individuano tali stringhe di valori (dette *run*). Esse infatti possono essere codificate trasmettendo una sola volta il valore che si ripete seguito dal numero di ripetizioni. È però necessario "avvisare" il decodificatore che i due prossimi simboli hanno un particolare significato. Per fare questo è necessario premettere alla coppia valore-numero di ripetizioni, un opportuna sequenza che non può essere usata per codificare dati ordinari.

Questa tecnica è evidentemente molto più semplice della codifica di Huffman; può funzionare bene (anche con segnali non stazionari), ma soltanto se le "corse" sono sufficientemente frequenti e lunghe.

1.2.6 Tecniche di compressione lossy

Anche se non è sempre necessaria, la conoscenza della semantica dei dati di ingresso consente di applicare i metodi di compressione lossy con particolare efficienza, visto che diventa possibile scartare solo i dati meno significativi. Ad esempio, per la compressione di immagini, sappiamo che i dati di ingresso sono valori di luminanza (ed eventualmente di crominanza); conoscendo le principali caratteristiche del sistema di visione umano, è possibile individuare quali informazioni sono meno rilevanti, e quindi possono essere scartate senza compromettere la qualità soggettiva dell'immagine.

I parametri per valutare una tecnica di compressione lossy comprendono senz'altro il tasso di codifica (o il rapporto di compressione) e la complessità, come nel caso lossless; tuttavia a questi è necessario aggiungere delle informazioni sull'entità della perdita

di informazione, cioè sulla *distorsione* introdotta dall'algoritmo. La misura di distorsione da usare dovrebbe essere facilmente trattabile analiticamente, e dovrebbe essere anche significativa da un punto di vista soggettivo. Questi due obiettivi sono però in contrasto tra loro, in quanto una misura della distorsione, che tenga conto del modo con cui le informazioni sono percepite da un osservatore umano, tipicamente risulta abbastanza complessa.

La misura più utilizzata per la distorsione è l'*errore quadratico medio*, indicato spesso con l'abbreviazione *MSE* (Mean Square Error) e definito come:

$$\overline{\varepsilon^2} \triangleq E[(X - \hat{X})^2] \tag{1.9}$$

dove $E[\cdot]$ è l'operatore di media statistica, X è il generico valore in ingresso, e $\hat{X}$ è il corrispondente valore decodificato. Questa misura di distorsione soddisfa il primo dei requisiti, e solo parzialmente il secondo (almeno nel caso video). Per questo motivo è opportuno affiancare, a misure basate sull'errore quadratico medio, delle misure che tengano conto della qualità soggettiva dei dati decodificati.

Di solito, le caratteristiche statistiche della sorgente non sono note e, in questo caso, la (1.9) non può essere usata. Al posto della media statistica si può allora impiegare uno stimatore come ad esempio la media temporale; si ha allora:

$$\hat{\varepsilon}^2 \triangleq \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{1.10}$$

dove N è un numero opportuno di campioni.

Esistono altre misure di distorsione collegate con l'MSE. Si introducono il *rapporto segnale rumore*, indicato con *SNR* (Signal to Noise Ratio):

$$\mathrm{SNR} \triangleq 10 \log_{10} \frac{E[X^2]}{E[(X - \hat{X})^2]} = 10 \log_{10} \frac{E[X^2]}{\overline{\varepsilon^2}} \tag{1.11}$$

ed il *rapporto segnale rumore di picco*, o *PSNR* (Peak Signal to Noise

Ratio):

$$\mathrm{PSNR} \triangleq 10 \log_{10} \frac{X_{\max}^2}{E[(X - \hat{X})^2]} = 10 \log_{10} \frac{X_{\max}^2}{\overline{\varepsilon^2}} \tag{1.12}$$

Nel caso di immagini (o di sequenze video) spesso si assume $X_{\max}^2 = 255$ (massimo valore della luminanza o dell'intensità di un colore quando queste grandezze sono rappresentate su 8 bit). In questo caso, l'errore quadratico medio ed il PSNR sono equivalenti. Notiamo che tanto nella (1.11) quanto nella (1.12), gli operatori di media statistica possono essere sostituiti con quelli di media temporale.

1.2.7 Quantizzazione scalare

La quantizzazione [11] è l'operazione centrale di ogni operazione di compressione lossy, anzi, come sarà chiaro più avanti, qualsiasi tipo di compressione può, concettualmente, essere visto come un caso particolare di quantizzazione vettoriale.

Cominciamo a parlare della quantizzazione scalare. Per definizione, la quantizzazione scalare è una corrispondenza $\mathcal{Q}$ che associa ad ogni elemento di $\mathbb{R}$ un elemento di un sottoinsieme discreto di $\mathbb{R}$ stesso, detto *codebook*. Si ha:

$$\mathcal{Q} : \mathbb{R} \to \mathcal{C} = \{y_1, y_2, ..., y_N\} \subset \mathbb{R} \tag{1.13}$$

Il codebook $\mathcal{C}$ è costituito da N elementi y_i, detti livelli d'uscita (o di restituzione), o anche codeword. Si definisce poi il *tasso di codifica* R:

$$R \triangleq \log_2 N \tag{1.14}$$

Il tasso di codifica, espresso in bit per simbolo, indica quanti bit sono necessari per rappresentare un livello di restituzione. Si introducono poi le *regioni* o *celle di quantizzazione*, indicate con $\mathcal{R}_i$, e definite come:

$$\mathcal{R}_i \triangleq \{x \in \mathbb{R} : \mathcal{Q}(x) = y_i\} \tag{1.15}$$

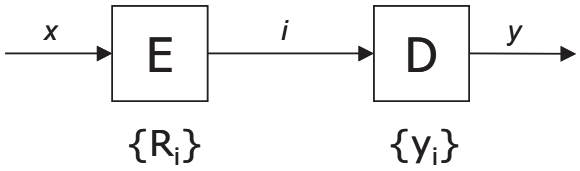

Figura 1.3: Il quantizzatore come cascata di un codificatore e di un decodificatore

Ci occuperemo solo dei *quantizzatori regolari*, cioè quelli per i quali:

- ogni regione $\mathcal{R}_i$ è un intervallo:

$$\mathcal{R}_i = (x_{i-1}, x_i) \tag{1.16}$$

con

$$\begin{aligned} \cup_i \mathcal{R}_i &= \mathbb{R} \\ \mathcal{R}_i \cap \mathcal{R}_j &= \varnothing \quad \forall i \neq j \end{aligned}$$

- ogni livello di restituzione appartiene alla cella corrispondente:

$$x_{i-1} < y_i < x_i \tag{1.17}$$

I valori $\{x_i\}$ sono detti *soglie di decisione*. Notiamo che l'operazione di quantizzazione è univocamente definita una volta che siano state assegnate le regioni di quantizzazione (o l'insieme delle soglie di decisione) ed i corrispondenti livelli di restituzione (cioè il codebook).

Tanto la quantizzazione scalare, quanto quella vettoriale, può essere vista come sequenza di due operazioni: una di *codifica*, che associa ad ogni valore d'ingresso x un indice i, univocamente specificato dall'insieme $\{\mathcal{R}_i\}$ delle regioni di quantizzazione; ed una di decodifica, che associa ad ogni indice i il livello di restituzione y_i, e che è quindi univocamente specificata dal codebook. In formule, il codificatore realizza la corrispondenza:

$$\mathcal{E} : x \in \mathbb{R} \rightarrow i \in \{1, ..., N\} : x \in \mathcal{R}_i \tag{1.18}$$

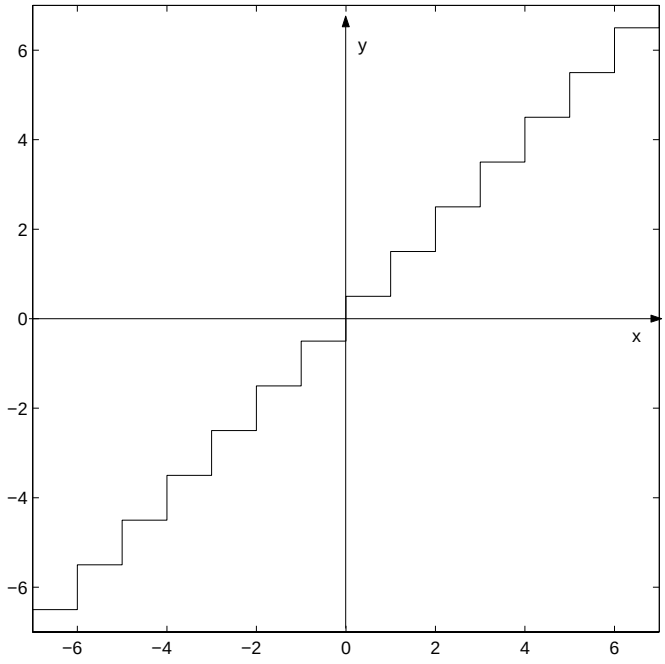

Figura 1.4: Quantizzatore uniforme

ed il decodificatore, invece:

$$\mathcal{D} : i \in \{1, ..., N\} \rightarrow y_i \in \mathcal{C} \tag{1.19}$$

I parametri tramite i quali si misurano le prestazioni di un quantizzatore sono, conformemente a quanto detto nel paragrafo 1.2.6, il tasso di codifica e la distorsione introdotta, misurata tipicamente come errore quadratico medio.

Il tipo più semplice di quantizzatore è il *quantizzatore uniforme* (vedere figura 1.4) che nel caso in cui l'ingresso varia in un intervallo limitato è caratterizzato da celle di ampiezza costante e da livelli di restituzione posti al centro di ogni cella:

$$x_i - x_{i-1} = \Delta \qquad \forall i \in \{1, ..., N\} \tag{1.20}$$

$$y_i = \frac{x_i + x_{i-1}}{2} \qquad \forall i \in \{1, ..., N\} \tag{1.21}$$

Il quantizzatore uniforme è molto semplice, e, come si può facil-

mente dimostrare, per segnali d'ingresso limitati, è quello che minimizza l'errore massimo di quantizzazione data la cardinalità del codebook. Queste proprietà di semplicità e robustezza giustificano l'ampio uso dei quantizzatori uniformi nella pratica, anche se essi non sono quelli ottimi (in termini di distorsione ottenibile con un certo tasso).

1.2.8 Quantizzazione predittiva

Se i campioni da quantizzare non sono indipendenti è possibile effettuare una predizione del prossimo campione a partire dai precedenti. Nella figura 1.5 è illustrato lo schema generale della codi-

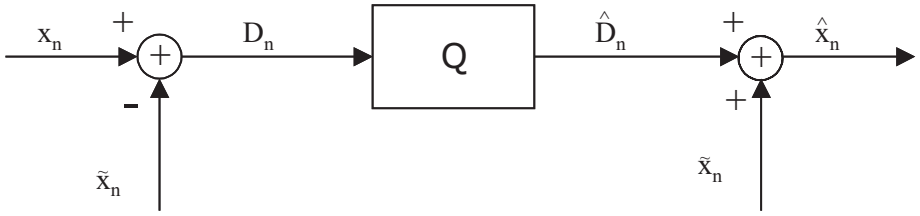

Figura 1.5: Schema generale di un quantizzatore predittivo

fica predittiva. Anziché codificare il simbolo corrente x_n, si valuta la differenza D_n tra la predizione $\tilde{x}_n$ e il campione, detta *errore di predizione*, e se ne trasmette una versione quantizzata $\hat{D}_n$. Se la predizione è accurata, l'errore di predizione avrà una dinamica più piccola del segnale originario, e quindi a parità di tasso di codifica, si può ottenere una distorsione minore. In ricezione si valuta la stessa predizione $\tilde{x}_n$ del simbolo corrente e la si corregge con l'errore di predizione quantizzato $\hat{D}_n$. È immediato verificare che l'errore commesso quantizzando x_n con $\hat{x}_n$ è lo stesso che si commette quantizzando D_n con $\hat{D}_n$.

Il valore predetto $\tilde{x}_n$ è ottenuto come funzione dei valori passati. La funzione ottima di predizione dipende dalla distribuzione di probabilità dei campioni d'ingresso, e solo nel caso gaussiano è lineare. In pratica però la predizione è comunque ottenuta semplicemente come combinazione lineare dei valori precedenti. Spesso addirittura la predizione *coincide* col campione precedente: in questo modo ciò

che si codifica è la differenza prima del segnale d'ingresso. Questa tecnica consente di ottenere buoni risultati se il segnale d'ingresso è lentamente variabile.

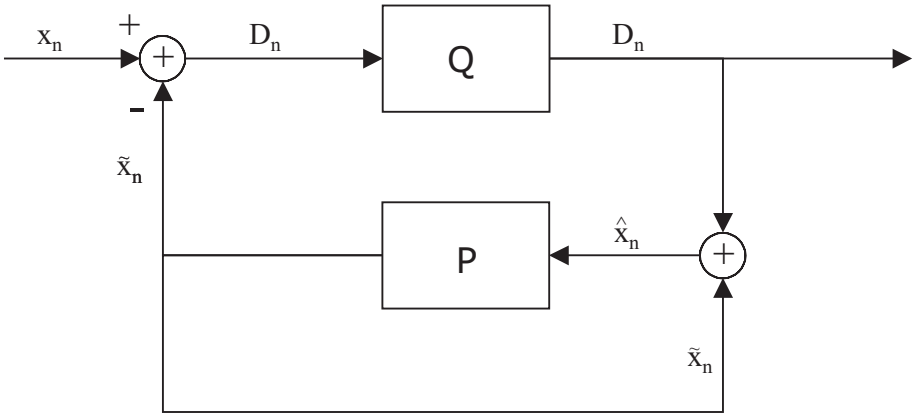

Figura 1.6: Quantizzazione predittiva: codificatore

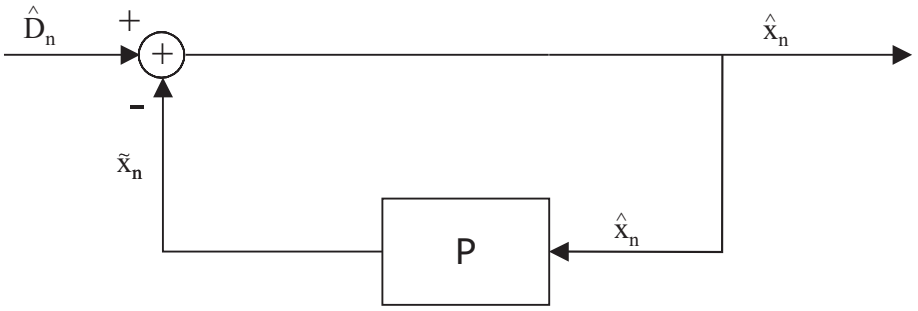

Figura 1.7: Quantizzazione predittiva: decodificatore

In figura 1.6 si illustra lo schema di un codificatore predittivo. Si noti che i valori usati per predire x_n sono quelli che anche in ricezione si è in grado di ricostruire esattamente, e cioè quelli ottenibili a valle della quantizzazione. Se invece si predicesse il campione corrente a partire dai valori dei campioni precedenti non quantizzati, l'errore di predizione sarebbe poco significativo in ricezione, perché la stima di x_n da cui si parte è diversa.

In figura 1.7 è illustrato lo schema del decodificatore: si noti come esso coincida con il loop di retroazione del codificatore.

Gli schemi di codifica predittiva sono molto usati perché, nonostante la loro semplicità, in molte situazioni consentono di ottenere buone prestazioni. Tuttavia essi sono soggetti al problema della *propagazione degli errori*: se il valore $\hat{D}_n$ non viene ricevuto correttamente (ad es. perché il canale è rumoroso), la decodifica di x_n risultà errata; ma visto che la predizione di x_{n+1} dipende da x_n, anche questa risulterà affetta da errore, e così via per tutti i campioni successivi: un solo errore (sull' n-esimo campione) si propaga potenzialmente a tutti i campioni successivi. Questo problema si può risolvere trasmettendo periodicamente un campione invece che il relativo errore di predizione, in modo da azzerare la propagazione dell'errore.

1.2.9 Quantizzazione vettoriale

Fin dalla pubblicazione dei primi lavori di Shannon, apparve chiara la possibiltà di migliorare le prestazioni di un codificatore operando congiuntamente su più campioni, invece che su uno per volta. Nel caso della quantizzazione, si può allora pensare di operare su gruppi di k campioni, realizzando la cosiddetta *quantizzazione vettoriale* (VQ) [11]. La VQ realizza una partizione dello spazio k-dimensionale in regioni di decisione, per poi associare, ad ogni gruppo di k campioni in ingresso appartenente alla i-esima regione, la parola codice $\mathbf{y}_i$. Allora la VQ è una corrispondenza:

$$\mathcal{Q} : \mathbb{R}^k \rightarrow \mathcal{C} \subset \mathbb{R}^k \tag{1.22}$$

dove il *codebook* $\mathcal{C}$ è formato da vettori di $\mathbb{R}^k$:

$$\begin{aligned} \mathcal{C} &\triangleq \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\} \\ \mathbf{y}_i &\in \mathbb{R}^k \ \forall i \in \{1, ..., N\} \end{aligned} \tag{1.23}$$

Si introduce poi il *tasso di codifica* R, che ha sempre il significato di numero di bit necessari per rappresentare un campione d'ingresso quantizzato:

$$R = \frac{1}{k} \log_2 N \tag{1.24}$$

Notiamo che, nel caso i simboli di ingresso siano blocchi di pixel di un'immagine, R è espresso in bit/pixel. L'insieme delle regioni di quantizzazione è $\Omega = \{\mathcal{R}_i\}_{i=1,...,N}$. Risulta:

$$\mathcal{R}_i \triangleq \{\mathbf{x} \in \mathbb{R}^k : \mathcal{Q}(\mathbf{x}) = \mathbf{y}_i\} \quad \forall i \in \{1, ..., N\} \tag{1.25}$$

$$\mathcal{R}_i \subseteq \mathbb{R}^k \quad \forall i \in \{1, ..., N\} \tag{1.26}$$

$$\mathcal{R}_i \cap \mathcal{R}_j = \oslash \quad \forall j \neq i \tag{1.27}$$

$$\cup_i \mathcal{R}_i = \mathbb{R}^k \tag{1.28}$$

Da queste definizioni si deduce che la quantizzazione scalare è un caso particolare di quantizzazione vettoriale, con $k = 1$. Si può anche considerare un quantizzatore scalare come un quantizzatore vettoriale di dimensionalità $k > 1$, ma con regioni di decisione delimitate da iperpiani ortogonali agli assi.

Analogamente al caso scalare si introduce il concetto di *quantizzatore regolare*.

- Un quantizzatore regolare ha celle convesse:

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{R}_i \Rightarrow \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in \mathcal{R}_i \quad \forall \lambda \in [0, 1] \tag{1.29}$$

- Ogni codeword appartiene alla corrispondente regione di decisione:

$$\mathbf{y}_i \in \mathcal{R}_i \quad \forall i \tag{1.30}$$

La VQ tipicamente permette di migliorare le prestazioni (in termini di distorsione in funzione del tasso) rispetto al caso scalare, per due motivi. Il primo è che con la quantizzazione vettoriale si è in grado di sfruttare l'eventuale dipendenza statistica esistente tra più campioni. Il secondo è che con la VQ possiamo considerare regioni di decisione di forma qualsiasi nello spazio k-dimensionale. Inoltre la VQ ha un altro vantaggio rispetto alla quantizzazione scalare, e cioè quello di consentire di ottenere tassi frazionari; per chiarire questo basta confrontare le equazioni (1.24) e (1.14).

Condizioni di ottimalità di un quantizzatore vettoriale

Ricordiamo che un quantizzatore (vettoriale) equivale alla cascata di un codificatore ed un decodificatore; il primo è completamente definito una volta che sia assegnato l'insieme $\Omega = \{\mathcal{R}_i\}$ delle regioni di quantizzazione; il secondo lo è una volta che sia assegnato il codebook $\mathcal{C} = \{\mathbf{y}_i\}$.

L'ottimizzazione del quantizzatore dovrebbe realizzarsi operando congiuntamente su Ω e $\mathcal{C}$. In pratica, però, noi disponiamo di condizioni di ottimalità (relativamente semplici) su Ω fissato $\mathcal{C}$, e su $\mathcal{C}$ fissato Ω. In altre parole, possiamo individuare il codificatore ottimo per un dato decodificatore, ed il decodificatore ottimo per un dato codificatore. Ricordiamo che l'obiettivo dell'algoritmo di ottimizzazione è quello di minimizzare la distorsione fissata la cardinalità N del codebook e la dimensionalità k dei vettori. La misura di distorsione considerata è quella definita dalla (1.9) o dalla (1.10).

$$D = E[||\mathbf{x} - \mathcal{Q}(\mathbf{x})||^2] \tag{1.31}$$

Fissato $\mathcal{C}$, la regola di ottimalità per Ω è quella della minima distorsione. Detta $d(\cdot,\cdot)$ la metrica usata per valutare la distorsione (nella 1.31 si è scelta la distanza euclidea), la scelta ottimale delle regioni di distorsione è:

$$\mathcal{R}_i = \{\mathbf{x} : \ d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \ \forall j \neq i\} \tag{1.32}$$

In questo modo assegnamo ogni possibile vettore d'ingresso al più vicino (secondo la metrica $d(\cdot,\cdot)$) elemento del codebook, determinando così la migliore scelta per Ω assegnato $\mathcal{C}$. La (1.32) è nota anche come condizione del "vicino più vicino" o *nearest neighbour* (NN). Poiché tale condizione è necessaria per l'ottimalità del quantizzatore vettoriale, possiamo limitarci a considerare i quantizzatori per i quali essa è sempre soddisfatta. A questo punto, le prestazioni del quantizzatore dipendono solo dal codebook.

Fissato Ω, bisogna scegliere le codeword $\mathbf{y}_i$ in modo da minimizzare la distorsione media complessiva D. Indichiamo con $B \subseteq \mathbb{R}^k$ l'insieme dei possibili valori d'ingresso, e con $f_{\mathbf{X}}(\mathbf{x})$ la densità di

probabilità del vettore d'ingresso $\mathbf{X}$. Dalla (1.31) si ha:

$$D = \int_B f_{\mathbf{X}}(\mathbf{x})||\mathbf{x} - \mathcal{Q}(\mathbf{x})||^2 d\mathbf{x} \tag{1.33}$$

$$= \sum_{i=1}^{N} \int_{\mathcal{R}_i} f_{\mathbf{X}}(\mathbf{x})||\mathbf{x} - (\mathbf{y}_i)||^2 d\mathbf{x} \tag{1.34}$$

$$\tag{1.35}$$

Imponiamo la condizione di ottimalità:

$$0 = \frac{\partial D}{\partial y_{ij}} \tag{1.36}$$

$$= -\int_{\mathcal{R}_i} f_{\mathbf{X}}(\mathbf{x}) \cdot 2(x_j - y_{ij}) d\mathbf{x} \tag{1.37}$$

Quindi:

$$y_{ij} = \frac{\int_{\mathcal{R}_i} x_j f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}}{\int_{\mathcal{R}_i} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}} \tag{1.38}$$

e, infine:

$$\mathbf{y}_i = \frac{\int_{\mathcal{R}_i} \mathbf{x} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}}{\int_{\mathcal{R}_i} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x}} \tag{1.39}$$

$$= E[\mathbf{X}|\mathbf{X} \in \mathcal{R}_i] \tag{1.40}$$

La 1.40 è nota come condizione del *centroide*: la codeword $\mathbf{y}_i$ deve coincidere con il centroide della i-esima regione di quantizzazione.

Esiste infine una terza condizione necessaria per l'ottimalità di un codebook, ed è la cosiddetta condizione della frontiera a probabilità zero: la probabilità che un elemento di $\mathbb{R}^k$ appartenga alla frontiera di una qualsiasi regione di quantizzazione deve essere nulla. Questa condizione è banale nel caso continuo (le frontiere hanno volume nullo), ma non nel caso discreto.

Supponiamo di avere un quantizzatore vettoriale che soddisfa le tre condizioni appena descritte. Ciò garantisce che esso sia globalmente o almeno localmente ottimo? Ricordiamo che il nostro obiettivo è quello di minimizzare la distorsione D, che è una funzione di Ω e $\mathcal{C}$. Se assumiamo che la condizione NN sia sempre soddi-

sfatta, possiamo considerare D come funzione del solo codebook, cioè funzione di kN variabili reali. Allora quando diciamo che un codebook è localmente ottimo intendiamo che ogni *piccola* perturbazione del codebook non fa diminuire la distorsione; un codebook globalmente ottimo è tale invece che *qualunque* altro codebook ha una distorsione non inferiore.

Se abbiamo un codebook che soddisfa le condizioni necessarie di ottimalità, esso è localmente ottimo[6]. È importante rilevare che la distorsione introdotta da un codebook localmente ottimo potrebbe essere molto maggiore di quella relativa all'ottimo globale.

Bisogna infine notare che spesso non si hanno a disposizione le statistiche complete dei campioni d'ingresso, ma bisogna stimarle a partire da un certo insieme di dati prodotti dalla sorgente stessa, cioè da un insieme di campioni capaci di rappresentare le statistiche della sorgente, il cosiddetto *training set*.

L'algoritmo di Lloyd generalizzato

L'algoritmo di Lloyd generalizzato (*GLA*) permette di migliorare le prestazioni di un'assegnato codebook[7], attraverso una procedura iterativa.

Cominciamo a descrivere l'algoritmo di Lloyd nel caso in cui la *pdf* congiunta del vettore d'ingresso sia nota. Il generico passo dell'iterazione consiste nelle due seguenti operazioni:

- Dato il codebook $\mathcal{C}^{(m)} = \{\mathbf{y}_i^{(m)};\ i = 1, ..., N\}$, trovare la partizione ottimale dello spazio applicando la condizione del vicino più vicino:

$$\mathcal{R}_i^{(m+1)} = \{\mathbf{x} : d(\mathbf{x}, \mathbf{y}_i) < d(\mathbf{x}, \mathbf{y}_j)\ \forall j \neq i\} \tag{1.41}$$

  Se un valore di $\mathbf{x}$ è equidistante da due o più codeword, $\mathbf{x}$ può essere assegnato ad una qualsiasi tra le celle corrispondenti.

- Usare la condizione del centroide per individuare il codebook

[6] Almeno nel caso di valori d'ingresso discreti, sotto ipotesi poco restrittive.

[7] Esistono molti modi per generare un codebook di partenza: il più semplice è quello di campionare il training set.

ottimo $\mathcal{C}^{(m+1)}$ per l'insieme di regioni appena individuato:

$$\mathbf{y}_i^{(m+1)} = E[X|X \in \mathcal{R}_i^{(m+1)}] \tag{1.42}$$

È banale dimostrare che ogni applicazione dell'iterazione di Lloyd riduce, o al più, lascia invariata la distorsione media del codebook. L'algoritmo di Lloyd consiste semplicemente nell'applicare iterativamente i passi appena descritti, fino a quando la riduzione relativa della distorsione nell'ultima iterazione risulta inferiore ad un certo valore di soglia.

Questa forma dell'algoritmo richiede che la pdf dell'ingresso sia nota, e che ad ogni passaggio si calcolino i centroidi delle celle, valutando degli integrali multipli su regioni di $\mathbb{R}^k$. Questo risulta generalmente impossibile per via analitica, anche nel caso di pdf con espressione semplice, e l'unico modo per risolvere il problema rimane il calcolo numerico. In ogni caso, spesso una adeguata descrizione analitica della pdf dell'ingresso non è disponibile, mentre è in genere facile procurarsi un insieme di campioni ricavato dalla sorgente per via empirica, cioè un training set.

Supponiamo di avere un insieme di osservazioni,

$$\mathcal{T} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_M\}, \tag{1.43}$$

dove M è la dimensione del training set[8]. Il training set è usato per valutare le condizioni di ottimalità tramite la media temporale invece che la media statistica. La distorsione associata al quantizzatore $\mathcal{Q}(\cdot)$ viene definita come:

$$D[\mathcal{Q}(\cdot)] = \frac{1}{M} \sum_{i=1}^{M} ||\mathbf{u}_i - \mathcal{Q}(\mathbf{u}_i)||^2 \tag{1.44}$$

Allora, assegnato un codebook di partenza, è evidente che per minimizzare la distorsione (1.44), Bisogna codificare ogni $\mathbf{u}_i$ con la codeword più vicina. Il training set viene allora suddiviso nei sot-

[8] A volte si definisce *training ratio* il rapporto $\beta = M/N$. Affinché il codice progettato sia sufficientemente robusto, è opportuno che sia $\beta \gg 1$.

toinsiemi $\mathcal{T}_i$ definiti in questo modo:

$$\mathcal{T}_i = \{\mathbf{u}_k : \ ||\mathbf{u}_k - \mathbf{y}_i|| \leq ||\mathbf{u}_k - \mathbf{y}_j|| \quad \forall j \neq i\} \tag{1.45}$$

Si noti l'analogia tra la (1.45) e la (1.32). Per minimizzare la distorsione rispetto al codebook, bisogna imporre:

$$\begin{aligned} 0 &= \frac{\partial D}{\partial y_{ij}} \\ &= \frac{\partial}{\partial y_{ij}} \sum_{n=1}^{N} \sum_{\mathbf{u}_k \in \mathcal{T}_n} ||\mathbf{u}_k - \mathbf{y}_n||^2 \\ &= \frac{\partial}{\partial y_{ij}} \sum_{\mathbf{u}_k \in \mathcal{T}_i} (u_{kj} - y_{ij})^2 \\ &= \sum_{\mathbf{u}_k \in \mathcal{T}_i} -2(u_{kj} - y_{ij}) \end{aligned} \tag{1.46}$$

Da cui si ricava:

$$\begin{aligned} y_{ij} &= \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{u}_k \in \mathcal{T}_i} u_{kj} \\ \mathbf{y}_i &= \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{u}_k \in \mathcal{T}_i} \mathbf{u}_k \end{aligned} \tag{1.47}$$

Si noti l'analogia con l'equazione (1.40).

A questo punto l'iterazione di Lloyd per i dati empirici può essere definita in questo modo:

- Dato il codebook $\mathcal{C}^{(m)} = \{\mathbf{y}_i^{(m)};\ i = 1, ..., N\}$, trovare la partizione ottimale che suddivide il training set $\mathcal{T}$ in sottoinsiemi $\mathcal{T}_i$ usando la condizione (1.45) (e con un'opportuna regola per il caso di dati equidistanti).
- Data la nuova partizione, determinare il codebook $\mathcal{C}^{(m+1)} = \{\mathbf{y}_i^{(m+1)}\}$ usando la (1.47).

Ora che l'iterazione di Lloyd è stata definita sia per il caso in cui è nota la pdf, sia per il caso in cui si parta dai dati, l'algoritmo generalizzato di Lloyd può essere riassunto nei seguenti passi.

1. Si comincia con un codebook $\mathcal{C}^{(1)}$. Si pone $m = 1$.

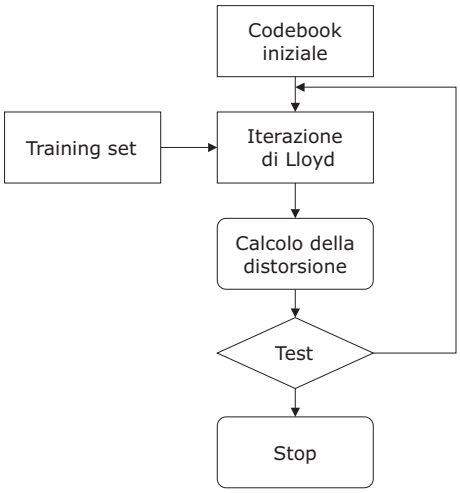

Figura 1.8: Diagramma di flusso del GLA

2. Dato il codebook $\mathcal{C}^{(m)}$, determinare il codebook $\mathcal{C}^{(m+1)}$ usando l'iterazione di Lloyd.

3. Calcolare la distorsione media per $\mathcal{C}^{(m+1)}$. Se la variazione relativa della distorsione è al disotto di una fissata soglia, l'algoritmo termina. Altrimenti si pone $m \leftarrow m + 1$ e si torna al punto 2.

Questo algoritmo è equivalentemente descritto col diagramma di flusso di figura 1.8.

Le prestazioni della VQ ed il "muro computazionale"

La quantizzazione vettoriale è, in linea di principio, la tecnica di compressione ottima, nel senso che qualsiasi tecnica di compressione può essere vista come caso particolare della quantizzazione vettoriale. Consideriamo infatti una qualsiasi tecnica di compressione che opera su dati d'ingresso di dimensione k; ad esempio, nel caso di compressione di immagini, $k = n \times m$, dove n e m sono le dimensioni orizzontale e verticale dell'immagine. Ad ogni immagine la tecnica di compressione considerata associa in modo univoco una particolare stringa di bit di uscita, che ipotizziamo sia di lunghezza finita. Possiamo allora interpretare tale operazione come una VQ

che opera su vettori d'ingresso appartenenti a $\mathbb{R}^{n \times m}$, mentre il codebook è formato da tutti i possibili file d'uscita prodotti dalla tecnica considerata.

Una volta che la generica tecnica di compressione è stata interpretata come quantizzazione vettoriale, le sue prestazioni possono essere migliorate usando il GLA.

Questo ragionamento ha un valore puramente teorico. Infatti un quantizzatore vettoriale progettato per quantizzare congiuntamente tutti i pixel che compongono un'immagine 512x512, con un tasso (ragionevole) di $R = 0.1$ bit/pixel, la cardinalità del codebook dovrebbe essere $N = 2^{0.1 \cdot 512^2} \cong 10^{7891}$.

Generalizzando il discorso, il problema è che la complessità della VQ è esponenziale con la lunghezza dei vettori usati, e quindi, l'unico modo per utilizzarla senza scontrarsi con il cosiddetto "muro computazionale", è quello di imporre dei vincoli su quelle che sono le caratteristiche dei vettori d'ingresso, del codebook, o del modo in cui si effettua la quantizzazione. Sono state così introdotte numerose tecniche di VQ "vincolata" che, pur essendo tutte subottime rispetto al caso generale, introducono un favorevole trade-off tra prestazioni e complessità. In altre parole, è spesso possibile ridurre la complessità di diversi ordini di grandezza, pagando solo una leggera perdita nelle prestazioni.

1.2.10 Tecniche di trasformata e DCT

La codifica con trasformata è uno schema alternativo alla VQ: infatti, uno dei concetti chiave di questa tecnica è la possibilità di decorrelare i campioni che costituiscono un vettore, in modo che la perdita derivante dall'uso della SQ al posto dei complessi algoritmi di VQ sia meno grave. Lo schema generale della codifica mediante trasformata è presentato in figura 1.9. Un blocco di dati di dimensione k è trasformato tramite la matrice ortonormale [9] $\mathbf{A}$. Ogni componente di $\mathbf{y}$ viene quantizzata scalarmente e indipendentemente dalle altre, poi il vettore $\hat{\mathbf{y}}$ subisce la trasformazione inversa $\mathbf{A}^{-1}$ e si ottiene così $\hat{\mathbf{x}}$, cioè la versione quantizzata del vettore d'ingresso.

[9] Una matrice $\mathbf{A}$ è detta *ortonormale* o *unitaria* se e solo se $\mathbf{A}^T\mathbf{A} = \mathbf{I}$

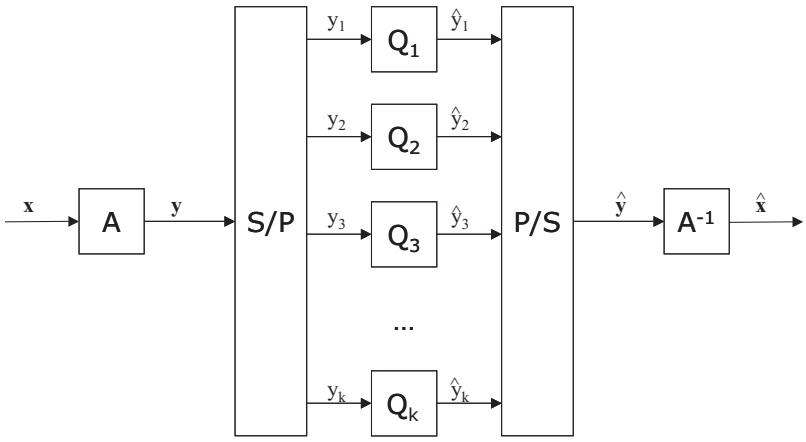

Figura 1.9: Codifica con trasformata: schema generale

Il fatto che $\mathbf{A}$ sia unitaria significa che la trasformazione applicata è una rotazione rigida nello spazio $\mathbb{R}^k$, e che l'errore di quantizzazione commesso su $\mathbf{y}$ è uguale a quello commesso su $\mathbf{x}$:

$$\begin{aligned} ||\mathbf{y}-\hat{\mathbf{y}}||^2 &= (\mathbf{y}-\hat{\mathbf{y}})^T(\mathbf{y}-\hat{\mathbf{y}}) \\ &= [\mathbf{A}(\mathbf{x}-\hat{\mathbf{x}})]^T[\mathbf{A}(\mathbf{x}-\hat{\mathbf{x}})] \\ &= (\mathbf{x}-\hat{\mathbf{x}})^T\mathbf{A}^T\mathbf{A}(\mathbf{x}-\hat{\mathbf{x}}) \\ &= ||\mathbf{x}-\hat{\mathbf{x}}||^2 \end{aligned} \tag{1.48}$$

Questo risultato ci consente di concentrare l'attenzione sulla quantizzazione di $\mathbf{y}$.

La trasformazione lineare $\mathbf{A}$ è applicata allo scopo di concentrare l'energia del vettore d'ingresso in poche componenti del vettore $\mathbf{y}$, che risulta a tutti gli effetti una codifica lossless[10] di $\mathbf{x}$, più adatta di tale vettore ad essere quantizzata efficientemente, perché ci possono essere molti coefficienti y_i che contengono pochissima o nessuna energia, e che quindi possono essere completamente eliminati. I punti chiave della codifica mediante trasformata risultano essere:

- La compattazione dell'energia su poche componenti. Ciò si ottiene scegliendo la matrice $\mathbf{A}$ della trasformazione.

[10] A meno di errori dovuti alla precisione limitata con cui i numeri sono rappresentati nei sistemi digitali.

- L'assegnazione delle risorse di codifica ai diversi coefficienti della trasformata. Questo si realizza con la progettazione dei quantizzatori.

Per quanto riguarda quest'ultimo punto, osserviamo che, assegnate le statistiche del vettore d'ingresso ed il numero totale B di bit con cui rappresentare $\mathbf{y}$, la distorsione introdotta dal quantizzatore dipende da come vengono assegnati i bit b_i con cui si quantizza l'i-esima componente di $\mathbf{y}$. Si tratta in altri termini di un problema di estremo vincolato, in cui bisogna minimizzare la funzione:

$$D = D(b_1, b_2, ..., b_k) \tag{1.49}$$

rispetto alle variabili $b_1, b_2, ..., b_k$, con il vincolo:

$$\sum_{i=1}^{k} b_i = B \tag{1.50}$$

Questo problema può essere risolto analiticamente nel caso in cui valga l'approssimazione di *alta risoluzione*, che consiste nelle seguenti condizioni: il numero di codeword N deve essere molto grande; l'ampiezza della più grande cella di quantizzazione deve essere molto piccola; la pdf dei campioni d'ingresso deve essere praticamente costante in una cella di quantizzazione. In formule:

$$N \rightarrow \infty \tag{1.51}$$

$$\Delta_i^{(max)} \rightarrow 0 \tag{1.52}$$

$$f_X(x) \cong f\left(\frac{x_i + x_{i+1}}{2}\right) \qquad \forall x \in \mathcal{R}_i \tag{1.53}$$

Se queste ipotesi sono verificate, si può dimostrare che l'assegnazione ottima dei bit di codifica è:

$$b_i = \frac{B}{k} + \frac{1}{2}\log\left[\frac{\sigma_i^2 h_i}{\sigma_{\mathrm{GM}}^2 h_{\mathrm{GM}}}\right] \tag{1.54}$$

dove σ_i^2 è la varianza dell'i-esima componente di $\mathbf{y}$, h_i è una costante che dipende dalla pdf di y_i, ed il pedice GM indica la media geometrica. Nel caso in cui tutte le componenti abbiano la stessa

distribuzione, la (1.54) si può interpretare così: i bit disponibili B vengono equamente ripartiti tra le k componenti, a meno di un termine di correzione che è positivo se la componente considerata ha varianza superiore alla media geometrica delle varianze, negativo altrimenti. In altre parole, l'assegnazione ottima impone di codificare con più bit le componenti meno prevedibili.

Si può inoltre facilmente verificare che, sempre nelle ipotesi di alta risoluzione, la distorsione risultante dall'assegnazione ottima è proporzionale a σ^2_{GM}. Tale quantità è tanto minore quanto più i valori di σ_i^2 sono disuniformi: ciò giustifica analiticamente, seppure solo in un caso particolare, l'esigenza, introdotta in base a ragionamenti euristici, di una trasformazione lineare che sia in grado di compattare le energie, cioè di massimizzare la varianza di alcuni coefficienti, e rendere molto piccola quella degli altri. La *trasformata di Karhunen-Lòeve* (KLT) è da questo punto di vista la trasformata ottima, nel senso che compatta meglio le energie, decorrela completamente le componenti del vettore e, almeno nel caso gaussiano, minimizza σ^2_{GM}. Tuttavia, essa è poco usata in pratica, perché richiede il calcolo della matrice di covarianza dei blocchi d'ingresso e dei relativi autovalori. In particolare, le caratteristiche statistiche del segnale d'ingresso spesso non sono note, ma devono essere stimate a partire dai dati. Ciò che però rende difficile valutare le statistiche di segnali come le immagini naturali o il segnale video è la loro non stazionarietà.

La trasformata coseno discreta

A differenza della KLT, la *trasformata coseno discreta* (DCT) fa parte delle trasformate che non dipendono dai dati, cioè la cui matrice di trasformazione è assegnata una volta per tutte. Ciò significa, da un lato, rinunciare a dei gradi di libertà, e quindi, presumibilmente, ridurre le prestazioni; dall'altro semplificare la progettazione e la realizzazione del sistema. Come poi vedremo in realtà le prestazioni della DCT sono più che soddisfacenti, tanto è vero che essa è la trasformata di gran lunga più utilizzata nell'ambito della compressione di immagini.

La DCT è strettamente legata alla *trasformata discreta di Fourier*[11] (DFT), e i coefficienti della trasformata rappresentano le frequenze spaziali presenti nel vettore d'ingresso. Nelle immagini naturali, la maggior parte dell'energia è concentrata alle basse frequenze; anche le alte frequenze però hanno un contenuto informativo rilevante, in quanto individuano i contorni delle figure che, in un'immagine codificata di buona qualità, devono essere nitidi. L'occhio umano però è meno sensibile alle alte frequenze spaziali, per cui esse possono essere in generale quantizzate più grossolanamente di quanto non si faccia con le basse frequenze, senza che ciò determini un sensibile peggioramento della qualità percepita dell'immagine.

I coefficienti della matrice di trasformazione **A** della DCT sono i seguenti.

$$a_{ij} = \begin{cases} \sqrt{\frac{1}{N}} & \text{per } i = 0 \text{ e } j = 0, 1, ..., N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N} & \text{per } i = 1, 2, ..., N-1 \text{ e } j = 0, 1, ..., N-1 \end{cases} \tag{1.55}$$

È facile verificare che la matrice **A** è unitaria e che le sue righe rappresentano, la prima una sequenza costante, e, le altre delle sequenze sinusoidali a frequenza crescente. Allora il vettore $\mathbf{y} = \mathbf{A}\mathbf{x}$ è costituito dalle proiezioni di **x** su varie frequenze, e quindi fornisce informazioni sul contenuto spettrale di **x**.

Notiamo che nella (1.55), si fa riferimento ad una matrice di trasformazione che accetta in ingresso dei vettori, che possono essere parte di righe o di colonne dell'immagine. È però possibile introdurre la DCT bidimensionale, che permette di ricavare le frequenze spaziali sia orizzontali che verticali di un blocco rettangolare di pixel. Anche la DCT bidimensionale è una trasformata ortonormale (e quindi invertibile); inoltre è una trasformata separabile, cioè equivalente a due operazioni di DCT monodimensionale (prima sulle righe e poi sulle colonne del blocco).

Per quant0 riguarda la complessità, esistono degli algoritmi ve-

[11] La DCT di una sequenza $x(n)$ di lunghezza N coincide con i primi N campioni della DFT di $x(n) + x(2N - n - 1)$, a meno di un cambiamento di fase per ottenere risultati reali.

loci per il calcolo della DCT, che richiedono un numero di operazioni che cresce come $n \log n$ piuttosto che come n^2.

La DCT è una trasformata che permette di compattare soddisfacentemente l'energia e soprattutto di separare le informazioni più importanti da quelle meno rilevanti: come anticipato, infatti, da una parte nelle immagini naturali, la maggior parte dell'energia si concentra alle basse frequenze spaziali, per cui molti coefficienti della DCT risulteranno nulli o molto piccoli; dall'altra, l'occhio umano è più sensibile alle basse frequenze che non alle alte. Allora le risorse di codifica saranno assegnate in modo privilegiato alle basse frequenze, permettendo una compressione che non introduce un degrado significativo nella qualità percepita dell'immagine.

1.3 La compressione del segnale video

Il segnale video è caratterizzato da un'elevata ridondanza spaziale ed un'elevata ridondanza temporale. Un algoritmo di codifica per questo tipo di segnale deve sfruttare entrambe le caratteristiche per poterlo comprimere soddisfacentemente.

1.3.1 La compressione spaziale

La compressione spaziale consente di ridurre il tasso di codifica di un immagine fissa sfruttando l'elevata correlazione esistente tra blocchi di pixel adiacenti. Questo risultato può essere ottenuto usando tecniche in grado di operare congiuntamente su più campioni, come la VQ e la codifica con trasformata. Quest'ultima tecnica è quella in pratica più diffusa per realizzare la compressione spaziale.

1.3.2 La compressione temporale

Generalmente le sequenze video contengono un alto grado di ridondanza temporale: se si considerano dei gruppi di frame consecutive, queste saranno spesso molto simili, perché, ad esempio, lo sfondo non è cambiato, e le uniche differenze sono dovute a figure che si muovono rispetto ad esso. È allora possibile evitare di codificare

nuovamente per ogni frame quelle informazioni che in qualche modo sono reperibili nelle frame precedenti. Le tecniche di compressione che sfruttano questo principio sono note come tecniche di compressione temporale, perché sfruttano la correlazione esistente tra blocchi di pixel all'interno di una sequenza video rispetto alla variabile temporale. Le principali tecniche di compressione temporale sono il *Conditional Replenishment* (CR) e la *compensazione del movimento*, o *Motion Compensation* (MC).

In entrambe le tecniche ogni frame è suddivisa in blocchi di pixel, comunemente detti *macroblocchi*. Nel CR, ogni macroblocco di una frame è confrontato con il macroblocco che nella frame precedente occupa la stessa posizione.

Figura 1.10: Conditional Replenishment: confronto tra macroblocchi

Per stabilire se i due macroblocchi sono simili si calcola una metrica, che può essere ad esempio l'energia della differenza tra i due blocchi, e la si confronta con una soglia opportunamente fissata. Se la distanza tra i blocchi è inferiore alla soglia i due blocchi sono ritenuti simili, e quindi non sarà necessario codificare il macroblocco. Bisognerà però avvertire il decodificatore che il CR su questo blocco ha avuto successo, inserendo nel flusso di codifica un bit di

sincronizzazione (o *side information*), ad esempio un "1".

Se invece i due blocchi sono ritenuti diversi, come ad esempio quelli evidenziati in figura 1.10, bisogna trasmettere un bit di sincronizzazione che indica il fallimento del CR (in questo esempio, uno "0"), seguito da tutti i bit necessari per codificare il macroblocco. In decodifica, quando si riceve un bit di sincronizzazione pari a "1", si recupera il macroblocco dalla frame precedente; quando invece si riceve uno "0", si deduce che il CR non ha avuto successo e che i bit seguenti sono quelli che codificano il macroblocco.

Questa tecnica, molto semplice, consente di ottenere risultati soddisfacenti quando la sequenza è sufficientemente stazionaria. Si pensi però ad una sequenza video in cui la telecamera ruota, ad esempio per mostrare un panorama. In questo caso, sebbene due frame consecutive siano molto simili tra loro, il CR fallirà quasi sempre, perché due macroblocchi nella stessa posizione potranno essere abbastanza diversi. La tecnica della MC risulta in questo caso più adeguata. L'unica differenza rispetto al CR è che nella MC si calcola la distanza del macroblocco corrente non solo dal macroblocco nella stessa posizione nella frame precedente, ma anche da tutti quelli che si trovano in posizioni vicine. La minima tra queste distanze viene confrontata con la soglia e, se inferiore, si può evitare di trasmettere il nuovo blocco. Bisogna però trasmettere anche il *vettore di movimento*, che individua la posizione relativa tra i due macroblocchi e che consente al decodificatore di ricostruire correttamente la scena.

L'estensione dell'area di ricerca per la MC determina un tradeoff tra la capacità di compressione e la complessità dell'algoritmo: infatti ogni operazione di calcolo della distanza è tipicamente molto onerosa in termini computazionali; d'altra parte, più è grande tale area, più possibilità ha l'algoritmo di individuare con successo il movimento. In quest'ottica, il CR può essere visto come un algoritmo di MC con area di ricerca di ampiezza nulla.

Anziché sostituire i macroblocchi, nelle tecniche di MC e CR si può realizzare una codifica predittiva: quando la ricerca ha successo, invece che non trasmettere per niente il macroblocco, si trasmette l'errore di predizione. In questo modo, si ha una minore

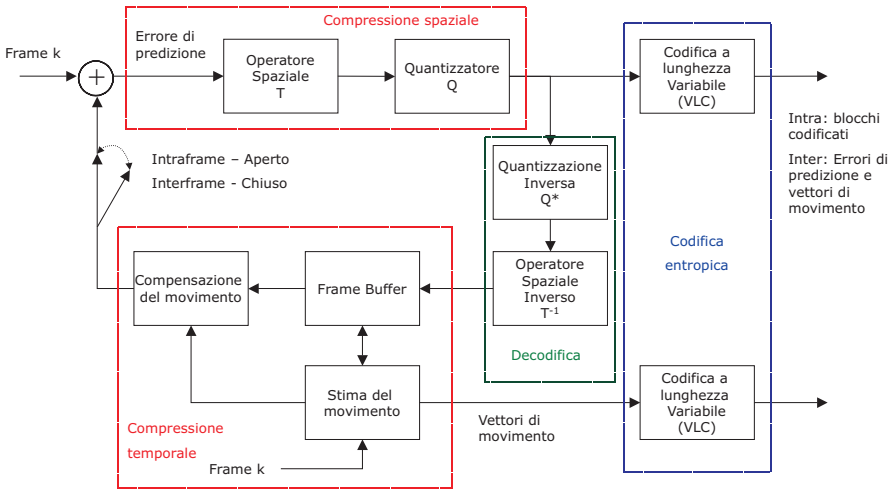

Figura 1.11: Schema generale di un codificatore video

compressione, ma una maggiore qualità della sequenza ricostruita. La strategia illustrata in precedenza, può essere vista come un caso particolare di questa tecnica predittiva, nel quale l'errore di predizione è quantizzato con zero bit.

1.3.3 Schema generale di un sistema di codifica video

In generale, i codificatori video realizzano la compressione riducendo la ridondanza spaziale e la ridondanza temporale del segnale video [12]. Anche i principali standard internazionali di codifica seguono questo approccio. Nella figura 1.11, è presentato uno schema molto generale di codificatore video, in cui son presenti due modalità di funzionamento, dette *interframe* ed *intraframe*.

In modalità interframe l'anello di retroazione genera una predizione dei blocchi della frame corrente usando le frame memorizzate nel frame buffer. La predizione è realizzata tramite l'algoritmo di compensazione del movimento, che genera anche i vettori di movimento.

Periodicamente, il codificatore passa in modalità intraframe, (disattivando il loop di retroazione), nella quale le immagini sono codificate direttamente, senza codifica predittiva. Ciò è necessario per implementare le seguenti funzionalità.

Accesso casuale. Se il codificatore lavorasse sempre in modalità predittiva, sarebbe necessario decodificare tutte le prime n frame per poter decodificare la frame $n+1$ e le successive. Inserendo invece periodicamente una frame *intra*, è sufficiente attendere la prima di tali frame nel flusso per poter cominciare la decodifica

Fast Forward. Per poter scorrere rapidamente la sequenza è sufficiente decodificare solo le frame *intra*, che sono codificate indipendentemente dal resto.

Controllo della propagazione degli errori. Con la codifica predittiva, l'errore nella decodifica di una frame si propaga alle successive. Inserendo periodicamente una frame codificata in modo indipendente, si evita che questi errori si propaghino indefinitamente.

La contropartita dell'uso della codifica intra è che questa è molto meno efficiente dal punto di vista della compressione rispetto alla codifica inter.

Passiamo ora a descrivere un po' più nel dettaglio i blocchi dello schema in figura 1.11.

La prima parte del codificatore effettua la compressione spaziale, tramite l'operatore T, che è tipicamente una trasformata lineare bidimensionale e unitaria, in grado di concentrare l'energia del segnale d'ingresso in pochi coefficienti, e di decorrelare i campioni del segnale. La trasformata più usata è la DCT. Il quantizzatore opera sui coefficienti già trasformati: si ha una compressione lossy, che in genere può ridurre di molto il tasso di codifica, perché la trasformata genera molti coefficienti nulli o quasi nulli, che possono essere scartati. Tipicamente il quantizzatore che si usa a valle della trasformata è scalare, ma in uno schema di codifica video, la compressione spaziale può anche essere completamente affidata alla quantizzazione vettoriale. Infine, la codifica entropica (VLC, codifica a lunghezza variabile) può essere utilizzata per eliminare la ridondanza statistica residua presente tra i coefficienti quantizzati. Anche i vettori di movimento sono tipicamente molto correlati tra lo-

ro, per cui è possibile comprimere queste informazioni con la VLC.

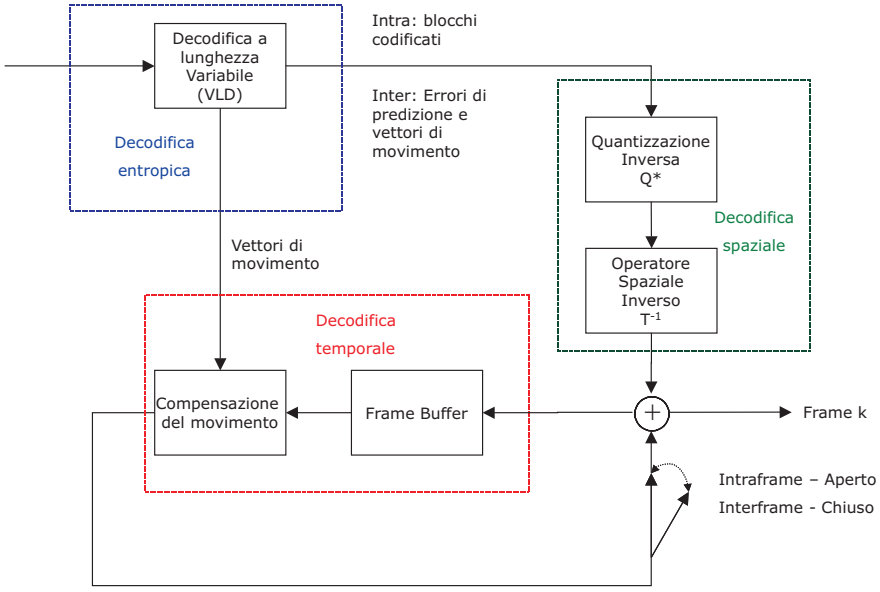

Figura 1.12: Schema generale di un decodificatore video

Affinché la codifica predittiva sia corretta è necessario usare gli stessi dati che saranno disponibili in decodifica, e cioè quelli che si ottengono dopo la trasformata e la quantizzazione. Il blocco di stima e compensazione del movimento opera come illustrato nel paragrafo 1.3.2.

Il decodificatore (figura 1.12) inverte le operazioni del codificatore: in modalità intra, decodifica i blocchi dell'immagine; in modalità inter, recupera i vettori i movimento e corregge i blocchi di riferimento con l'errore di predizione.

1.4 Gli standard di compressione

1.4.1 Lo standard JPEG

Lo standard JPEG[12], creato nel 1991 nell'ambito dell'ISO[13], è un'insieme di algoritmi di compressione, inizialmente destinato solo alle immagini, e poi esteso anche alle sequenze video, con il Motion JPEG (MJPEG). In questa sede ci interessa però soprattutto descrivere il trattamento di immagini fisse, in quanto gli standard H.261 e MPEG utilizzano la stessa tecnica per la codifica intra.

Lo standard JPEG comprende quattro possibili modalità di codifica:

1. Codifica sequenziale lossy basata su DCT. È uno schema di base, supportato da tutte le implementazioni di codificatori e decodificatori JPEG.

2. Codifica espansa lossy basata su DCT. È un ampliamento dello schema di base.

3. Codifica lossless sequenziale. Consente di ricostruire esattamente l'immagine di partenza.

4. Codifica gerarchica. L'immagine è codificata in risoluzioni multiple.

Lo standard contiene anche una seconda parte in cui si descrivono i procedimenti da usare per verificare se un'implementazione è conforme con lo standard.

Modalità sequenziale lossy

L'algoritmo di codifica nella modalità base può essere schematizzato in sei passaggi, come illustrato in figura 1.13. L'immagine sorgente è trasformata nelle componenti YC_rC_b, dove Y indica, come al solito, la componente di luminanza[14], mentre i segnali di crominanza

[12] Joint Photographic Expert Group.
[13] International Standard Organization.
[14] Se l'immagine è in toni di grigio, la luminanza è l'unica componente presente

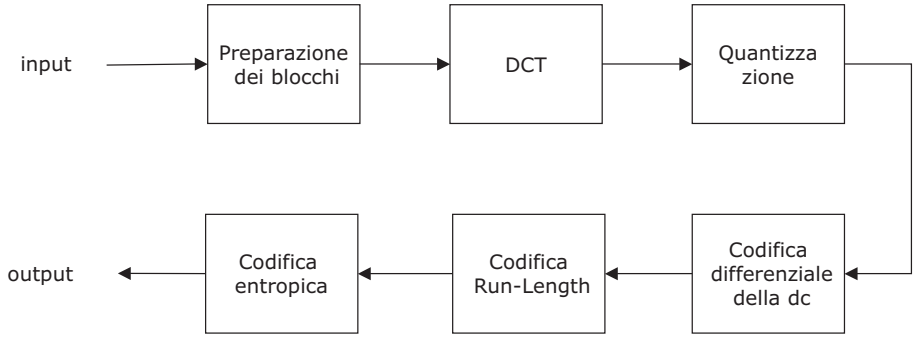

Figura 1.13: L'algoritmo JPEG in modalità sequenziale lossy

sono definiti come $C_r = R - Y$ e $C_b = B - Y$, dove R e B sono le componenti relative ai colori rosso e blu. I campioni di ognuna delle tre componenti devono essere rappresentati con 8 bit per pixel, per cui i loro valori vanno da 0 a 255; inizialmente si sottrae 128 ad ogni campione, sicché il *range* dei valori diventa {-128,-127,...,127}. Le componenti di crominanza sono sottocampionate di un fattore due sia orizzontalmente che verticalmente, pertanto, se l'immagine originale era composta dalle tre matrici relative ai colori primari ognuna di dimensioni $n \times m$, dopo questa prima fase avremo una matrice di luminanza di dimensioni $n \times m$ e due di crominanza di dimensioni $n/2 \times m/2$. Il numero di campioni si è dimezzato, ma un'osservatore difficilmente si accorge della differenza, visto che l'occhio umano è molto più sensibile alla luminanza che alla crominanza. Il primo passo dell'algoritmo termina con la divisione di ognuna delle tre matrici in blocchi di 8×8 campioni.

Il secondo passo dell'algoritmo consiste nell'applicare ad ognuno dei blocchi la DCT che produce un altro blocco 8×8, con il coefficiente di posto (0,0) che rappresenta il valor medio del blocco originario, mentre gli altri elementi indicano quanta energia è associata ad ogni frequenza spaziale. Normalmente i coefficienti della DCT decrescono rapidamente al crescere della frequenza spaziale. In teoria, la DCT è perfettamente invertibile; in pratica, la precisione limitata con cui si rappresentano i numeri, e le approssimazioni con cui si valutano le funzioni trigonometriche causano degli errori, che

danno luogo ad una piccola (di fatto trascurabile) perdita di qualità.

Il terzo passo consiste nella quantizzazione e nell'eliminazione dei coefficienti meno significativi. Questa operazione si effettua dividendo ogni coefficiente per un opportuno peso, scelto da una tabella, e approssimato all'intero più vicino. Poiché tali pesi crescono rapidamente a partire dal coefficiente (0,0), i coefficienti relativi alle più alte frequenze spaziali vengono quasi tutti azzerati, tranne quelli realmente significativi (si veda la figura 1.14 per un esempio). Questa strategia, da un lato, permette di azzerare molti coefficienti, perché nelle immagini naturali la maggior parte dell'energia è concentrata alla basse frequenze; dall'altro, comporta una perdità di qualità percettiva minima, perché l'occhio umano è poco sensibile alle alte frequenze spaziali. Si noti che la tabella dei pesi per la quantizzazione non è specificata dallo standard: ogni applicazione può sceglierla in modo da gestire il trade off tra tasso e qualità dell'immagine (pesi più elevati consentono una maggiore compressione a scapito della qualità dell'immagine ricostruita).

Nel passo successivo si effettua una codifica predittiva del coefficiente (0,0) calcolando la differenza tra questo valore ed il quello dello stesso coefficiente nel blocco precedente. Visto che queste quantità sono i valori medi dei blocchi, dovrebbero cambiare lentamente, e quindi l'errore di predizione in genere è molto piccolo.

Nel quinto passo i coefficienti quantizzati vengono ordinati tramite la "scansione a zig-zag" della matrice (vedere figura 1.15). In questo modo si riesce a concentrare alla fine del blocco la maggior parte dei coefficienti nulli, il che consente di applicare efficientemente la codifica run-length. L'ultimo passo consiste nell'applicare una codifica entropica[15] all'output del codificatore run-length.

La decodifica di un immagine compressa viene effettuata invertendo le operazioni fatte in fase di codifica.

Con questa tecnica normalmente si riesce ad ottenere un fattore di compressione da 10 a 20 senza degrado apparente della qualità.

[15] È una versione modificata dell'algoritmo di Huffman.

Coefficienti DCT

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Coefficienti quantizzati

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Tabella di quantizzazione

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Figura 1.14: Calcolo dei coefficienti quantizzati della DCT

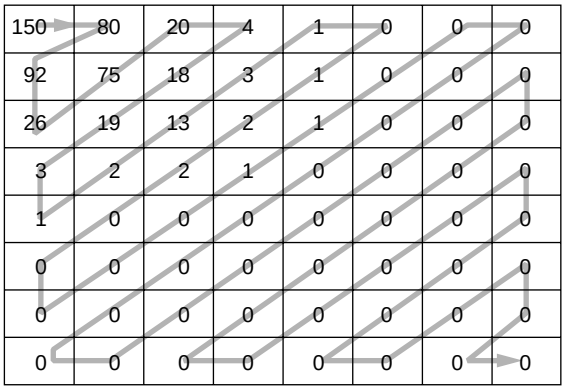

Figura 1.15: L'ordine in cui i coefficienti quantizzati sono trasmessi

Modalità espansa

La modalità espansa differisce da quella di base perché ha alcune opzioni in più:

- è possibile avere un'immagine sorgente codificata con un numero di bit per pixel che va da 8 a 12;
- permette di usare anche la codifica aritmetica per la compressione lossless a valle della run-length;
- definisce una tecnica di codifica progressiva, oltre a quella sequenziale.

L'ultima caratteristica è, in effetti, quella più interessante e può essere realizzata tramite due tecniche: la selezione spettrale e l'approssimazione successiva. Nel primo caso, dopo aver codificato tutti i blocchi di un'immagine, vengono trasmessi tutti insieme i coefficienti relativi alle bande di frequenza più basse. In ricezione ciò permette di ricostruire una prima versione approssimata dell'immagine. Successivamente vengono trasmesse le altre bande, che consentono di migliorare la qualità dell'immagine decodificata. Invece, con il metodo delle approssimazioni successive, nella prima trasmissione, si inviano solo i bit più significativi dei coefficienti; poi, un bit alla volta, si migliora la precisione di tutti i coefficienti. Le due tecniche di codifica progressiva possono essere usate sia separatamente, sia insieme.

Modalità lossless sequenziale e modalità gerarchica

La modalità lossless sequenziale utilizza una combinazione di codifica predittiva e codifica entropica. Come tutti gli schemi senza perdite il fattore di compressione che si riesce a conseguire non è molto elevato: tipicamente varia tra 2 a 3.

La modalità gerarchica implementa una tecnica di codifica piramidale (vedere anche il paragrafo 2.5), che consente di decodificare l'immagine a risoluzioni spaziali multiple.

1.4.2 Lo standard H.261

La sigla H.261 indica un insieme di norme[16] stabilite dall'ITU[17] e finalizzate alla videotelefonia ed alla video conferenza su rete ISDN[18] a $p \times 64$ kbps, con $p = 1, ..., 30$ [12]. Questo standard garantisce un ritardo di non più di 150 ms per frame nella fase di codifica, in modo da rendere possibili applicazioni di videoconferenza bidirezionali ed in tempo reale.

Gli schemi del codificatore e del decodificatore H.261 sono molto simili agli schemi generali illustrati nelle figure 1.11 e 1.12. In particolare il codificatore, rappresentato nella figura 1.16, differisce dallo schema generale per la presenza dei blocchi per il controllo del tasso d'uscita. Per il resto, la codifica in modalità intraframe è identica alla codifica JPEG, con l'uso della DCT, la successiva quantizzazione scalare e la scansione a "zig-zag" della matrice dei coefficienti. Il controllo del tasso avviene monitorando il buffer di trasmissione: quando il grado di riempimento supera una certa soglia, è necessario ridurre il tasso, quando è al di sotto di un'altra, può essere aumentato. Queste modifiche si conseguono principalmente operando sulla tabella di quantizzazione: se è necessario ridurre il tasso, i pesi vengono aumentati. Si ottiene così una rappresentazione più grossolana, ma molti coefficienti vanno a zero, in modo da aumentare il fattore di compressione. Riducendo i pesi della tabella si ottiene evidentemente l'effetto opposto.

Il codificatore H.261 può operare nelle due modalità intraframe ed interframe. In modalità intraframe i blocchi vengono interamente codificati; in modalità interframe viene codificato l'errore di predizione, usando (opzionalmente) la tecnica di motion compensation. Diversamente dallo schema generale proposto in precedenza, nello standard H.261 la scelta della modalità di compressione non è fatta per un intera frame, ma blocco per blocco. Questa scelta ha il vantaggio di non generare un forte picco nell'andamento del tasso di trasmissione, in quanto nessuna frame è codificata interamente in modalità intra (a meno di un brusco cambiamento di scena,

[16] Si parla di *raccomandazione* H.261.
[17] International Telecommunications Union
[18] Integrated Services Digital Network.

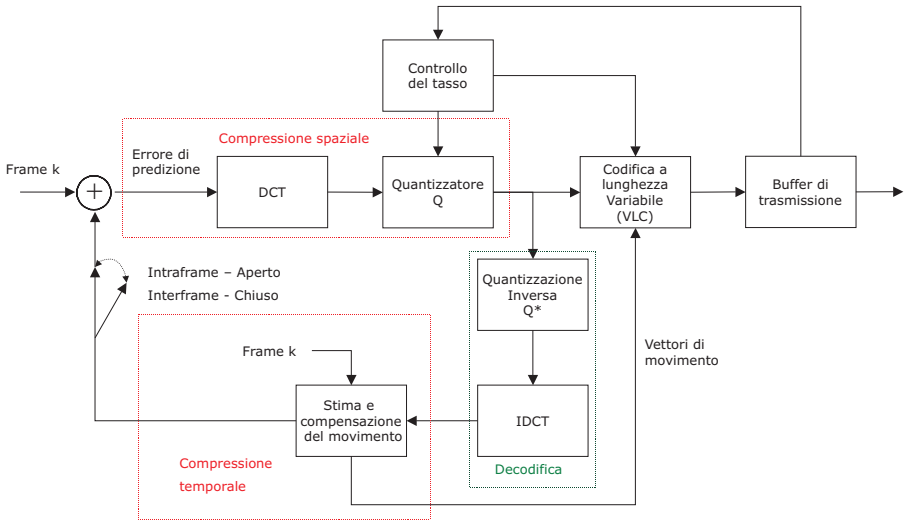

Figura 1.16: Codificatore H.261

che, in applicazioni di video telefonia, non dovrebbe essere consueto). L'assenza di frame periodicamente codificate in modalità intra rende impossibile un'operazione come il fast forward, che però non ha senso in queste applicazioni. Il vero svantaggio dell'assenza di una frame in codifica intra è che è più difficile garantire il recupero degli errori e l'accesso casuale. A proposito di quest'ultimo problema, quando si accede casualmente ad una sequenza codificata con H.261, all'inizio si riusciranno a decodificare e visualizzare solo i macroblocchi dove c'è movimento. Per superare questo problema, è possibile far sì che il codificatore trasmetta comunque, periodicamente e progressivamente, anche i blocchi di sfondo (cioè quelli che non sono stati trasmessi da più di un certo numero di frame) in modalità intra. In questo modo, la codifica intra, invece di essere concentrata in una sola frame, causando un improvviso picco nel tasso, che è sempre difficile da gestire da parte della rete, viene distribuita in più frame, provvedendo così ad una sorta di *shaping* a priori del traffico. Questa tecnica non è prevista esplicitamente nello standard, ma è compatibile con esso, in quanto il decodificatore rimane comunque in grado di ricostruire correttamente la sequenza trasmessa.

1.4.3 Lo standard MPEG-1

Per affrontare il problema della compressione di sequenze video, nel 1988, all'interno dell'ISO, fu istituito il gruppo di lavoro denominato *MPEG* (Moving Picture Experts Group), con l'obiettivo di elaborare uno standard in grado produrre un output di qualità analoga a quella di un videoregistratore (frame di 352×240 pixel) ad un tasso di circa 1.5Mbps, in modo da permettere la memorizzazione di interi film su CD-ROM [12], [13]. Le applicazioni a cui è destinato lo standard sono quelle asimmetriche (cfr. par. 1.2), come ad esempio la memorizzazione e la distribuzione di film su supporti ottici o su rete, in cui la codifica è realizzata una volta (da parte di chi ha prodotto il flusso multimediale), e la decompressione molte (da parte di chi vuole usufruirne). In queste applicazioni asimmetriche, la complessità e il ritardo dell'operazione di codifica non costituisce un problema; invece il decodificatore deve essere semplice, veloce, ed economico, in modo da poter essere facilmente distribuito su larga scala.

Gli standard MPEG constano di tre parti principali:

Video, che tratta la compressione del segnale video.

Audio, che tratta la compressione del segnale audio.

Sistemi, che si occupa della sincronizzazione e della multiplazione dei flussi numerici compressi audio e video.

A queste parti se ne può aggiungere un'altra che tratta la compatibilità e la conformità allo standard. Qui ci occuperemo soltanto della parte relativa al segnale video.

Lo standard MPEG-1 si basa pesantemente su H.261 (e quindi, come quest'ultimo, su JPEG), per cui descriveremo principalmente le differenze tra i due. Queste differenze riguardano sostanzialmente due tecniche disponibili in MPEG-1, ma non in H.261: la predizione della MC con accuratezza al mezzo pixel, la MC bidirezionale, e la presenza di frame con codifica intra.

Nello standard H.261, l'accuratezza della MC è ristretta ad un numero intero di pixel. Tuttavia, un oggetto che si muove, non

necessariamente si riposiziona sulla griglia dei pixel, ma può portarsi in posizioni intermedie. Nello standard MPEG, per consentire un'accuratezza al mezzo pixel, l'immagine di luminanza è interpolata di un fattore due in entrambe le direzioni, e su questa immagine si può ricavare il vettore di movimento con la precisione richiesta. Questo vettore, scalato di 2, è poi usato come vettore di movimento per le immagini di crominanza, cosa non ottimale ma che consente un notevole risparmio di calcoli. I vettori di movimento sono codificati differenzialmente rispetto a quelli dei macroblocchi circostanti, perché tali vettori sono fortemente correlati, anzi molto spesso sono praticamente uguali.

Per illustrare le altre differenze rispetto a H.261, è opportuno far riferimento ai diversi tipi di frame presenti nello standard MPEG-1.

1. Frame I (Intracoded): frame codificate in JPEG. Queste frame sono necessarie per consentire l'accesso casuale, il fast forward, e per limitare la propagazione degli errori (cfr. par. 1.3.3)

2. Frame P (Predictive): sono codificate usando l'algoritmo di MC relativamente alla precedente frame di tipo I o di tipo P, similmente a quanto accade nell'H.261. Garantiscono una compressione maggiore rispetto alle frame I grazie alla MC. Esse servono anche da riferimento per le frame B e per le successive frame P.

3. Frame B (Bidirectional): permettono che i macroblocchi siano codificati usando un algoritmo di MC bidirezionale, cioè facendo riferimento sia alla precedente sia alla successiva frame di tipo I o P. Questo determina un fattore di compressione tipicamente maggiore di quello delle frame P. Per non complicare troppo la struttura, le frame B non vengono usate come riferimento, per cui non possono neanche propagare gli errori. In figura 1.17 sono illustrati i riferimenti usati per l'algoritmo di MC per le varie frame.

4. Frame D (DC-coded): frame a bassa risoluzione, ottenute decodificando solo il coefficiente DC di ogni macroblocco. Le frame

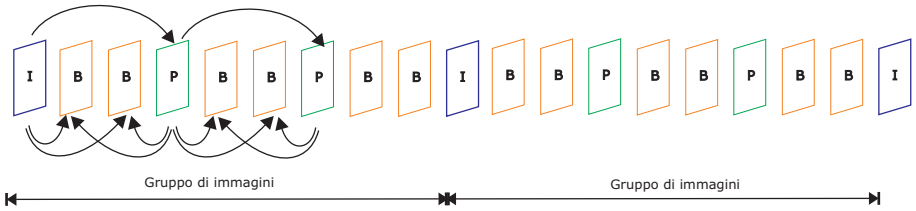

Figura 1.17: I gruppi di immagini (GOP) in MPEG.

D non sono usate in combinazione con gli altri tipi, e, anzi, sono usate abbastanza raramente; sono state definite per permettere il fast forward su dispositivi sequenziali come i nastri magnetici.

In MPEG ogni sequenza video è divisa in uno o più *gruppi di frame* o *GOP*, ognuno dei quali è a sua volta composto da una o più frame, di cui almeno una deve essere di tipo I. Le frame di tipo I e P sono dette *anchor frame*; normalmente, il numero di frame comprese tra due anchor frame è indicato con M, e quello tra due frame I con N. In figura 1.17, dove è illustrato l'insieme di riferimenti usati per la motion compensation, si ha $M = 3$ e $N = 9$. Scegliere valori grandi per N e M significa avere molte frame di tipo B, che hanno la migliore compressione; d'altra parte, così facendo, diminuisce la correlazione tra le frame P, ed aumenta la complessità del codificatore. Infatti, con riferimento alla figura 1.17, l'ordine di codifica deve essere 1,4,2,3,7,5,6,... visto che non è possibile codificare la frame 2 prima di aver codificato la 1 e la 4, e così via. Al decodificatore, l'ordine delle frame decodificate è ancora 1,4,2,3,7,5,6,..., ma alla fine devono essere riordinate, per cui dopo aver decodificato la frame 4, questa deve rimanere memorizzata mentre si decodificano e mostrano le frame 2 e 3. Questa tecnica quindi richiede memoria e comporta ritardi.

Per permettere una realizzazione economica del terminale d'utente, lo standard definisce dei vincoli per alcuni parametri, come illustrato in tabella 1.3.

In conclusione, lo standard MPEG è stato pensato principalmente per la memorizzazione di flussi multimediali su memorie di mas-

sa. A causa della presenza delle frame B, ci può essere un notevole ritardo tra codifica e decodifica. Il codificatore è molto più complesso del decodificatore, a causa dell'ampio intervallo di ricerca per la MC, della tecnica per l'accuratezza al mezzo pixel, e per l'uso della MC bidirezionale. La sintassi dell'MPEG-1 può supportare una vasta gamma di frame rate e risoluzioni, e, similmente ad altri standard di codifica video, esso non specifica tutti i parametri di codifica (come ad esempio l'area di ricerca per la MC, i pesi della quantizzazione,...) stimolando così la competizione tra le diverse implementazioni.

Parametro	Vincolo
dimensione orizzontale	≤ 720 pixel
dimensione verticale	≤ 576 pixel
numero di MB per frame	≤ 396
numero di MB al secondo	$\leq 396 \times 25 = 330 \times 30$
frame rate	≤ 30 frame/s
bit rate	≤ 1.86 Mbps
decoder buffer	≤ 376832 bit

Tabella 1.3: Parametri vincolati in MPEG1

1.4.4 Lo standard MPEG-2

Lo standard MPEG-2 è stato sviluppato dal comitato MPEG per coprire un insieme di applicazioni non previste dal primo standard. In MPEG-1, il ridotto bit rate massimo consente la codifica solo a risoluzioni medio-basse, e non è previsto il supporto per il formato *interallacciato* usato in televisione [12], [13]. Inizialmente MPEG-2 era stato progettato per comprimere video con qualità di tipo televisivo, con tassi da 4 a 6 Mbps, mentre MPEG-3 era orientato alla HDTV[19]. Successivamente MPEG-3 è confluito in MPEG-2.

Lo standard MPEG-2 ha come obiettivo la codifica del video ad alta qualità, a tassi da 4 a 15 Mbps, per applicazioni come il *Video on Demand* (*VoD*), la trasmissione televisiva digitale (Digital Video

[19] High Definition TeleVision.

Broadcast, *DVB*), e la memorizzazione su supporti ad alta capacità come i DVD[20].

Per quanto riguarda le tecniche di codifica, MPEG-2 è molto simile a MPEG-1, con l'utilizzo della DCT, delle tecniche di MC, la presenza delle frame I, P, e B (le frame D non sono più supportate). Alcune differenze riguardano la dimensione dei blocchi per la DCT (da 8×8 a 10×10) e la necessità di MPEG-2 di supportare il formato interallacciato.

Lo standard MPEG-2 fornisce un vasto insieme di funzionalità, che possono essere combinate in modo flessibile. Per non avere un numero troppo elevato di combinazioni possibili, che causerebbe dei problemi di gestione, vengono definiti i *profili* e i *livelli*. I livelli stabiliscono le caratteristiche minime del flusso che il codificatore deve essere in grado di supportare, mentre i profili stabiliscono le funzionalità disponibili.

Livello	width [pixel]	height [pixel]	frame rate [frame/s]	bit rate [Mbps]	buffer [bit]
Low	352	288	30	4	475136
Main	720	576	30	15	1835008
High-1440	1440	1152	60	60	7340032
High	1920	1152	60	60	9781248

Tabella 1.4: Livelli previsti in MPEG-2

Le tabelle 1.4 e 1.5 descrivono rispettivamente i livelli ed i profili disponibili in MPEG-2. Si noti che in ogni profilo sono disponibili anche le funzionalità dei profili precedenti, per cui nella tabella sono evidenziate, per ognuno di essi, solo le caratteristiche aggiuntive rispetto al precedente. Le combinazioni tra profili e livelli definiscono le modalità di funzionamento dei codificatori (e dei decodificatori) MPEG-2, ma non tutte le possibili coppie sono raccomandate dallo standard[21]. Il profilo simple è definito per applicazioni che necessitano di ritardi contenuti e si realizza semplicemente eliminando le frame B. Il profilo principale è quello più importante e largamente usato, per generiche applicazioni video ad alta qualità, come VoD,

[20] Digital Versatile Disc.

[21] La maggior parte delle applicazioni usa la combinazione di livello principale e profilo principale.

Profilo	Funzionalità disponibili
Simple	Codifica non scalabile, gestione di flussi interallacciati, accesso casuale, non sono disponibili le frame B
Main	Gestione delle frame bidirezionali
SNR scalable	Due livelli di scalabilità in SNR
Spatial scalable	Due livelli di scalabilità in risoluzione
High	Frequenza di campionamento della crominanza raddoppiata

Tabella 1.5: Profili previsti in MPEG-2

DVD, DVB, e HDTV. Il profilo SNR-scalable supporta diversi gradi di qualità del segnale video, mentre il profilo Spatial-scalable supporta diverse risoluzioni (il concetto di scalabilità è più approfonditamente trattato nei paragrafi 2.1, 2.4, 2.5, e 2.6).

In sintesi, lo standard MPEG-2 è orientato ad applicazioni video di elevata qualità, con tassi superiori ai 2 Mbps. E' adatto sia per la codifica progressiva sia per quella interallacciata, ed adotta un'algoritmo di compressione che è molto simile a quello di MPEG-1. Inoltre, in MPEG-2 vengono introdotti i profili ed i livelli per gestire le combinazioni tra le funzionalità offerte. Anche in MPEG-2 si evita di definire tutti i parametri, in modo che le varie implementazioni possano ricercare la migliore combinazione.

Capitolo 2

La codifica scalabile a bassa complessità

2.1 Gli obiettivi del progetto

Il nostro problema è la realizzazione di un sistema software capace di consentire la trasmissione e la ricezione di sequenze video in *tempo reale* ad un insieme di utenti che dispongono di risorse molto diverse tra loro, sia in termini di *potenza di calcolo*, che di *larghezza di banda*[1] del collegamento. In particolare vogliamo che il flusso possa essere prodotto o fruito da ogni utente nel modo migliore possibile e che ciò possa avvenire in tempo reale anche quando la potenza di calcolo è limitata.

Questi requisiti si traducono in un insieme di specifiche abbastanza vincolanti, per quanto riguarda sia le problematiche relative alla trasmissione su rete, sia la codifica del flusso video. A proposito di quest'ultimo problema nasce l'esigenza di usare un algoritmo che sia *scalabile* ed *a bassa complessità*.

Mentre il concetto di bassa complessità è abbastanza semplice, è opportuno chiarire quello di scalabilità. Un flusso video scalabile ha una semantica tale che da esso è possibile estrarre dei sottoflussi che siano ancora significativi, cioè dai quali sia ancora possibile

[1] Il termine larghezza di banda è spesso usato, nell'ambito della telematica, con il significato di tasso di trasmissione disponibile, e misurato in bit al secondo. Anche qui verrà seguito quest'uso del termine, molto diffuso sebbene improprio.

ottenere la sequenza video, seppure dovendo rinunciare ad alcune caratteristiche. Il vantaggio è che la trasmissione e l'elaborazione di un flusso ridotto richiede un quantità di risorse che può adattarsi alla disponibilità locale. In particolare, in questo progetto, si è dato spazio a tre tipi di scalabilità:

Scalabilità in frame rate. L'utente può decidere di ricevere soltanto una frame ogni N, ma deve essere comunque in grado di decodificare le frame che riceve. In questo modo, la riproduzione del movimento sarà meno fluida, ma il tasso di trasmissione si riduce, ed il tempo massimo a disposizione per la decodifica di una frame aumenta. Ciò significa che anche con scarse risorse sarà possibile accedere alla sequenza video.

Scalabilità in risoluzione. L'utente può decidere di ricevere la sequenza video a diverse risoluzioni, in modo da adattarsi anche alle caratteristiche del terminale d'utente. Scegliendo le risoluzioni più basse, l'immagine sarà più piccola (o più sfocata se viene interpolata), ma le risorse, in termini di banda e di capacità di calcolo, saranno impiegate meno pesantemente.

Scalabilità in qualità (SNR). Ci può essere infine il caso in cui l'utente non vuole rinunciare né alla risoluzione né alla fluidità dei movimenti, e invece è disponibile a ricevere una sequenza più rumorosa. La contropartita deve essere ancora una volta la possibilità di decodificare la sequenza anche con risorse scarse.

Le tecniche che sono state utilizzate per realizzare il codec video devono soddisfare a questo insieme abbastanza stringente di specifiche. Le tecniche di compressione spaziale e temporale a bassa complessità che sono state utilizzate sono, rispettivamente, la VQ gerarchica e tabellare (HVQ), ed il conditional replenishment. Le caratteristiche di scalabilità sono state invece ottenute usando un'opportuna gestione gerarchica del CR per la scalabilità in frame rate, la codifica piramidale per la scalabilità in risoluzione, e i codebook con struttura ad albero per la scalabilità in qualità. Tutte queste tecniche vengono illustrate nel seguito di questo capitolo.

2.2 La compressione spaziale

La quantizzazione vettoriale è una tecnica di compressione in grado, potenzialmente, di sfruttare al meglio le dipendenze tra i pixel che costituiscono un'immagine. Tuttavia, nell'algoritmo di VQ ordinaria l'operazione di codifica è piuttosto complessa computazionalmente, in quanto richiede che si calcoli la distanza del vettore d'ingresso da tutti i vettori del codebook. Ciò non può essere realizzato in tempo reale, se non nel caso in cui si usino vettori di dimensioni ridotte, rinunciando quindi di fatto ai vantaggi della VQ, che sono più evidenti quando si possono usare vettori abbastanza lunghi.

2.2.1 La VQ tabellare

Una prima idea che sembra promettente per la realizzazione in tempo reale della VQ, anche con vettori d'ingresso non troppo piccoli, è quella d'effettuare la codifica utilizzando delle tabelle. Si tratta quindi di valutare in anticipo, per ogni possibile vettore d'ingresso $\mathbf{x}$, formato da k pixel, qual è, tra i vettori di un codebook precedentemente costruito, quello che meglio lo rappresenta. I risultati sono riportati in una tabella, a cui si accede utilizzando come indirizzo gli elementi di $\mathbf{x}$. La fase di costruzione della tabella può essere molto onerosa, ma viene fatta una volta per sempre. La tabella viene poi usata per la codifica di ogni vettore, semplicemente accedendo ad una sua locazione.

Questa tecnica sembra molto interessante, ma va sicuramente affinata. Proviamo infatti a valutare quali sarebbero la dimensioni della tabella nel caso in cui si volessero quantizzare immagini in toni di grigio, con $n = 8$ bit per pixel (bpp), usando vettori di $k = 8$ pixel ad un tasso di $R = 1$ bpp (quindi il codebook ha $N = 2^{kR} = 256$ codeword). I possibili vettori d'ingresso, sono $(2^n)^k = 256^8$, e per ognuno di essi, bisogna riservare in tabella lo spazio necessario a rappresentare l'indice che individua il vettore che lo codifica, cioè $\log_2 N = kR = 8$ bit. Le dimensioni risultanti della tabella sarebbero allora di $256^8 \times 8$ bit, pari a circa 17 miliardi di Gigabyte, un valore chiaramente non compatibile con l'attuale tecnologia. Notiamo per

inciso che la VQ tabellare ha le stesse prestazioni della VQ ordinaria purché usi la stessa metrica per la costruzione delle tabelle.

Un modo per superare questo ostacolo computazionale è rappresentato dall'approccio gerarchico. Prima però di descrivere la VQ gerarchica (HVQ), torniamo sul nostro esempio, e valutiamo le dimensioni che dovrebbe avere la tabella nel caso in cui si usassero vettori di dimensione $k = 2$, con lo stesso codebook di 256 elementi. Dovremmo riservare sempre 1 byte per elemento della tabella, ma questa volta, ne saranno necessari solo 256^2, per una dimensione finale della tabella di 64 kB [2]. Allora una tecnica di VQ puramente tabellare può essere implementata solo con tassi di codifica molto alti, corrispondenti a fattori di compressioni insoddisfacenti (nell'esempio precedente, il tasso di codifica è pari a 4 bpp e il rapporto di compressione è 2).

2.2.2 La VQ gerarchica

Utilizzando una tecnica di codifica di tipo gerarchico, è possibile implementare la VQ tabellare con tassi di codifica adeguati alle applicazioni. Il prezzo da pagare, come vedremo, sarà una degradazione delle prestazioni rispetto al caso della VQ ordinaria. La tecnica di HVQ che sta per essere presentata e che è stata usata nell'implementazione del codificatore è quella descritta in [7].

Nella figura 2.1 è illustrato lo schema della HVQ, con i valori effettivamente usati per i vari parametri. Supponiamo di avere in ingresso dei dati già quantizzati a $r_0 = 8$ bit per simbolo, come è il caso della codifica di sequenze video, e consideriamo vettori d'ingresso di dimensione k. Questi campioni vengono ripartiti in blocchi di $k_0 = 2$ elementi appartenenti alla stessa riga e che costituiscono l'ingresso del primo stadio del quantizzatore. Il quantizzatore codifica tali blocchi in modo tabellare: i $k_0 r_0$ bit che rappresentano il vettore d'ingresso sono usati per accedere ad una tabella di $2^{k_0 r_0}$ locazioni, in cui sono memorizzati gli indirizzi delle codeword che meglio approssimano il vettore corrispondente. Le codeword hanno lunghezza k_0 e vengono scelte in un codebook prefissato di cardinalità N_1,

[2] I multipli del byte sono sempre intesi come potenze di 2: 64kB=65536 byte. Si confronti la nota 5, pagina 4.

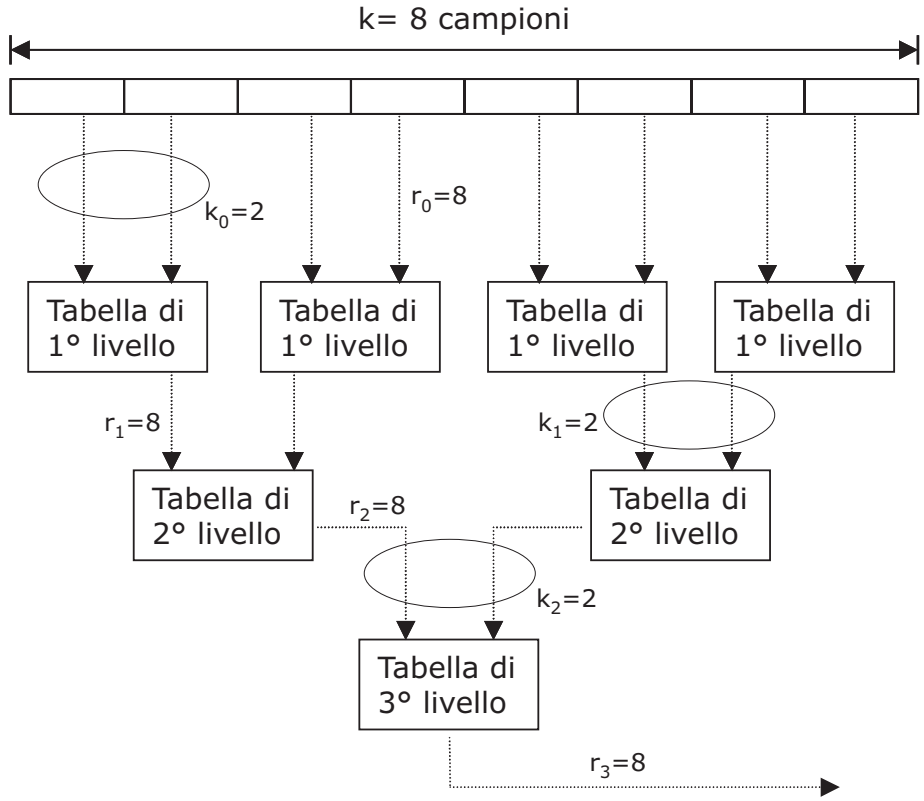

Figura 2.1: Schema della VQ gerarchica

pertanto la dimensione della tabella sarà uguale a $2^{k_0 r_0} \log_2 N_1$ bit. In uscita dal primo stadio avremo allora un indirizzo di lunghezza $r_1 = \log_2 N_1$ bit, che rappresenta la coppia di pixel d'ingresso. Questo stadio produce allora una compressione di $k_0 r_0 / r_1$. Nello stadio successivo, si considerano blocchi di k_1 indirizzi prodotti dallo stadio precedente e raggruppati per colonne. Ognuno di tali blocchi rappresenta un vettore di $k_0 \times k_1$ pixel, che viene nuovamente quantizzato in forma tabellare: si usano $k_1 r_1$ bit per accedere ad un'altra tabella, che permette di scegliere quale delle N_2 codeword di un ulteriore codebook di dimensionalità $k_0 \times k_1$ codifica meglio il blocco di pixel corrispondente ai bit del livello precedente. In uscita avremo allora una stringa di $r_2 = \log_2 N_2$ bit, e il rapporto di compressione risulta $r_0 k_0 k_1 / r_2$.

Generalizzando il discorso, il codificatore è costituito da M stadi,

di cui, l'm-esimo accetta in ingresso vettori di k_{m-1} campioni, ognuno rappresentato con r_{m-1} bit. Questi $k_{m-1}r_{m-1}$ bit sono usati per accedere ad una tabella di $2^{k_{m-1}r_{m-1}}$ elementi, che restituisce l'indirizzo della codeword che meglio rappresenta il vettore d'ingresso, espresso su r_m bit. Il rapporto di compressione allo stadio m è pari a $r_0/r_m \prod_{i=0}^{m-1} k_i$, e la dimensione della tabella relativa è di $r_m 2^{k_{m-1}rm-1}$ bit.

Nel nostro caso, si è assunto $k_m = 2$ e $r_m = 8$ per ogni stadio. In questo modo, le tabelle sono tutte di 64 kB, i codebook sono tutti costituiti da $N = 256$ elementi, ed il rapporto di compressione dello stadio m è pari a 2^m. In figura 2.1 è illustrato lo schema corrispondente della HVQ a $M = 3$ stadi. Nei prototipi realizzati si è considerato un numero di stadi pari a 3 o a 4, per un rapporto di compressione pari a 8 o a 16. D'ora in poi, per semplicità di notazione, considereremo sempre $k_m = 2$ e $r_m = 8$.

Valutiamo la complessità computazionale dell'algoritmo, con i fissati valori dei parametri. Per quantizzare un vettore di 2^m elementi sono necessari m stadi. Nel primo stadio si effettuano 2^{m-1} accessi in tabella, nel successivo la metà e così via, per un totale di $2^m - 1$ accessi. Al crescere di m, la complessità dell'algoritmo tende quindi ad un accesso in tabella per campione. Si comprende allora che questa tecnica è estremamente semplice e veloce, e quindi del tutto adatta alle nostre esigenze.

La tabella del primo stadio rappresenta una corrispondenza tra una coppia di indici in $\{0, 1, ..., 255\}$ e un altro indice, appartenente ancora allo stesso insieme. Se $m > 1$ i due indici d'ingresso identificano una coppia di codeword del codebook dello stadio $m - 1$. Questa coppia di codeword è una versione quantizzata di un blocco di 2^m pixel d'ingresso. Se $m = 1$ i due valori d'ingresso sono invece proprio due valori di luminanza. In ogni caso, a partire dalla coppia d'ingresso, è possibile ricostruire il blocco di pixel che deve essere quantizzato. Tale blocco è confrontato con tutti i vettori del codebook di livello m, e nella tabella si inserisce l'indice relativo a quello più simile, secondo un'opportuna misura di distorsione. Pertanto ogni stadio deve codificare dei blocchi già affetti da rumore di quantizzazione a causa degli stadi precedenti. Questo è il motivo per il

quale le prestazioni della HVQ sono inferiori alla VQ ordinaria. Ciò costituisce il prezzo da pagare per avere un algoritmo poco complesso, e quindi capace di funzionare in tempo reale anche su macchine non troppo potenti.

2.2.3 Misure di distorsione percettive

Uno dei vantaggi della HVQ è che le tabelle vengono costruite "off-line", e quindi la complessità di questa operazione non incide sulla complessità della codifica o della decodifica; si possono allora inglobare, nella generazione delle tabelle, delle misure di distorsione anche molto complesse, che ad esempio tengano conto delle caratteristiche del sistema di visione umano.

Una misura di distorsione abbastanza generale e, al tempo stesso, sufficientemente semplice da poter essere trattata analiticamente, è:

$$d(\mathbf{x}, \hat{\mathbf{x}}) = (\mathbf{T}\mathbf{x} - \mathbf{T}\hat{\mathbf{x}})^T \mathbf{W} (\mathbf{T}\mathbf{x} - \mathbf{T}\hat{\mathbf{x}}) \tag{2.1}$$

Questa misura rappresenta l'errore commesso rappresentando $\mathbf{x}$ con $\hat{\mathbf{x}}$, pesato secondo la matrice $\mathbf{W}$, nello spazio trasformato secondo la matrice $\mathbf{T}$. Spesso la matrice dei pesi usata è diagonale, per cui la (2.1) si semplifica in:

$$\begin{aligned} d(\mathbf{x}, \hat{\mathbf{x}}) &= \sum_{j=1}^{k} w_j (y_j - \hat{y}_j)^2 && (2.2) \\ &= d_T(\mathbf{y}, \hat{\mathbf{y}}) && (2.3) \end{aligned}$$

dove abbiamo indicato con $\mathbf{y} = \mathbf{T}\mathbf{x}$ i vettori dello spazio trasformato, con y_j la generica componente di $\mathbf{y}$, e con $d_T(\cdot, \cdot)$ la misura di *distorsione pesata* nello spazio trasformato. La matrice dei pesi consente di dare una diversa importanza agli errori commessi sulle componenti del vettore nel nuovo dominio: nel caso della DCT ciò significa assegnare un peso diverso agli errori commessi sulle diverse bande di frequenza spaziale, con la possibilità di rispettare la sensibilità relativa dell'occhio umano. Usando una tale misura di distorsione per la creazione delle tabelle, avremo come risultato che la codifica

assocerà ad un certo vettore d'ingresso, non quello a minima distanza, ma quello che dovrebbe essere considerato, con una certa approssimazione[3], più simile ad esso da un osservatore umano.

2.2.4 La HVQ con pesi percettivi

Costruendo le tabelle della HVQ con una misura di distorsione come la (2.2), si ha un algoritmo di codifica che unisce alla semplicità della HVQ, la possibilità di tener conto delle caratteristiche del sistema visivo umano. Questa tecnica è detta "Weighted Transform HVQ" (*WTHVQ*) [7]. In linea di principio, l'algoritmo per la realizzazione della WTHVQ è lo stesso della HVQ ordinaria, a meno di usare una misura percettiva di distorsione. Se però si opera nello spazio trasformato si può avere qualche risparmio computazionale.

Il primo passo della realizzazione di un sistema WTHVQ consiste nella generazione degli m codebook. In questa fase si può ad esempio usare un training set ottenuto prelevando, da un insieme di immagini, dei blocchi di pixel di dimensioni opportune, e operando poi la DCT su questi blocchi. Su questi dati si può applicare il GLA, utilizzando come misura di distorsione la d_T definita nella (2.3) e ottenendo il codebook nel dominio trasformato.

Il secondo passo è la costruzione delle tabelle. Cominciamo dal primo livello (si veda la figura 2.2). Per ognuna delle 2^{16} possibili

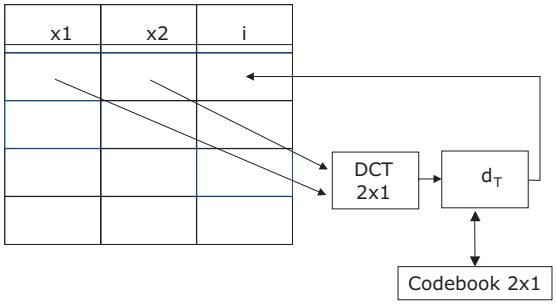

Figura 2.2: Costruzione della tabella nel primo stadio della WTHVQ

combinazioni di coppie di pixel d'ingresso viene calcolata la DCT. Il

[3] Si ricordi che il modello usato per descrivere il sistema di visione umano è molto semplificato.

vettore risultante viene confrontato con tutti i vettori del codebook usando la metrica d_T definita nella (2.3), e nella tabella viene inserito l'indice corrispondente alla codeword che minimizza questa misura di distorsione.

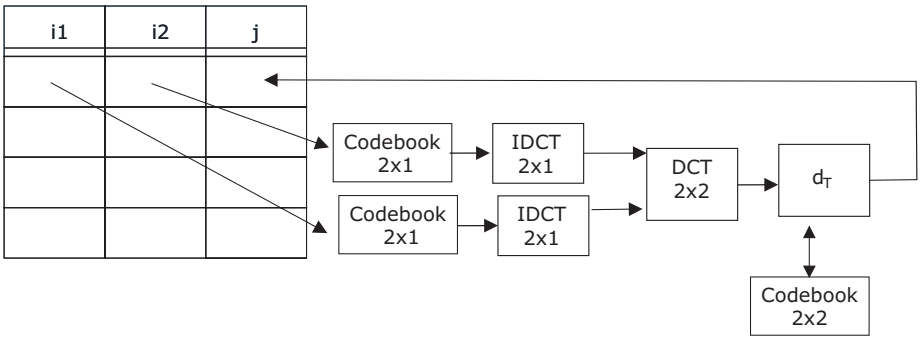

Figura 2.3: Costruzione della tabella nel secondo stadio della WTHVQ

La figura 2.3 illustra come costruire la tabella relativa al secondo stadio. Gli ingressi adesso sono gli indici del codebook di primo livello prodotti dallo stadio precedente, raggruppati per colonne. Ad ognuna delle 2^{16} possibili coppie di indici d'ingresso si associa la corrispondente coppia di codeword nello spazio trasformato. Le due codeword vengono antitrasformate con la DCT inversa 2×1 e poi il blocco 2×2, che si ottiene unendo le due codeword nel dominio originario, viene trasformato nuovamente secondo la DCT 2×2. Il risultato viene confrontato con tutti i vettori del codebook 2×2 trasformato, e si sceglie la codeword che minimizza la misura di distorsione d_T.

Le tabelle degli stadi successivi si costruiscono in modo analogo, ricordando che, per sfruttare al meglio la correlazione spaziale tra i pixel, è opportuno che i vari stadi lavorino alternativamente sulle righe e sulle colonne.

Riportiamo infine nelle tabelle 2.1 e 2.2 alcuni parametri prestazionali ottenuti comprimendo la stessa immagine con diverse tecniche: JPEG, VQ ordinaria, VQ gerarchica, e HVQ con pesi percettivi

[7]. Notiamo che l'algoritmo JPEG offre le migliori prestazioni, ma richiede tempi di calcolo non compatibili con un elaborazione in tempo reale. La VQ ordinaria ha prestazioni decisamente inferiori di quelle di JPEG e tempi di codifica molto lunghi: infatti nella VQ ordinaria, ogni blocco dell'immagine va confrontato con tutte le codeword per decidere qual è quella che lo deve rappresentare. L'introduzione della tecnica gerarchica abbatte i tempi di codifica a prezzo di una lieve perdita in prestazioni. Infine, l'uso di pesi percettivi riduce ulteriormente (sebbene di molto poco) le prestazioni in termini di PSNR, ma la qualità soggettiva delle immagini decodificate migliora sensibilmente.

Compressione	JPEG	VQ	HVQ	WTHVQ
2:1	46.9	41.7	41.7	40.0
4:1	40.8	35.9	35.3	34.8
8:1	37.7	32.5	31.8	31.7
16:1	34.7	30.5	29.7	29.8

Tabella 2.1: PSNR nella compressione dell'immagine "Lena"

Compressione	JPEG	VQ	HVQ	WTHVQ
2:1	250 ms	800 ms	12 ms	12 ms
4:1	250 ms	800 ms	24 ms	24 ms
8:1	250 ms	800 ms	27 ms	27 ms
16:1	250 ms	800 ms	30 ms	30 ms

Tabella 2.2: Tempi di codifica dell'immagine "Lena"

2.3 La compressione temporale

Per quanto riguarda la scelta dell'algoritmo per la compressione temporale l'esigenza di bassa complessità riduce le possibili tecniche ad una sola: il Conditional Replenishment. Rispetto a quanto già detto nel paragrafo 1.3.2, è sufficiente aggiungere poche precisazioni.

- Il confronto viene fatto sui blocchi di pixel rappresentati dalle codeword.

- L'errore di predizione non è proprio trasmesso, cioè è quantizzato con zero bit.

- La metrica utilizzata per stabilire se due blocchi sono sufficientemente simili è l'errore quadratico medio, cioé l'energia della differenza tra i due blocchi:

$$d = \sum_{i=1}^{k} (x_i^{(n)} - x_i^{(n-1)})^2 \tag{2.4}$$

  dove $x_i^{(n)}$ è l'i-esimo valore di luminanza del blocco corrente nella frame n, e $x_i^{(n-1)}$ è la luminanza dello stesso pixel nella frame $n-1$.

Infine, bisogna aggiungere che il codificatore deve generare periodicamente una frame in modalità "intra" per consentire l'accesso casuale ed il recupero degli errori.

2.4 La scalabilità in frame rate

Come anticipato nel paragrafo 2.1, l'algoritmo di codifica utilizzato deve generare un flusso di bit scalabile. In particolare, la scalabilità in frame rate consiste nella possibilità di ricevere solo una frame ogni N ed essere comunque in grado di decodificare correttamente la sequenza video. Bisogna però ricordare che il CR introduce dipendenza nella codifica e nella decodifica delle varie frame. Se un utente vuole accedere al flusso video con un frame rate dimezzato rispetto a quello totale, egli riceverà, ad esempio, la frame k, mentre non riceverà quella $k-1$. Tuttavia la codifica della frame k fa riferimento alla $k-1$, e quindi l'utente non sarà in grado di ricostruire la frame ricevuta.

Per superare questo problema è sufficiente organizzare i *livelli temporali* come illustrato in figura 2.4. Sono disponibili tre livelli temporali. Un utente può decidere se ricevere solo una frame ogni quattro, una ogni due, oppure tutte, e, corrispondentemente, riceverà solo il primo livello temporale, solo i primi due, oppure tutti. Il primo livello temporale è allora composto solo dalle frame k, $k+4$,

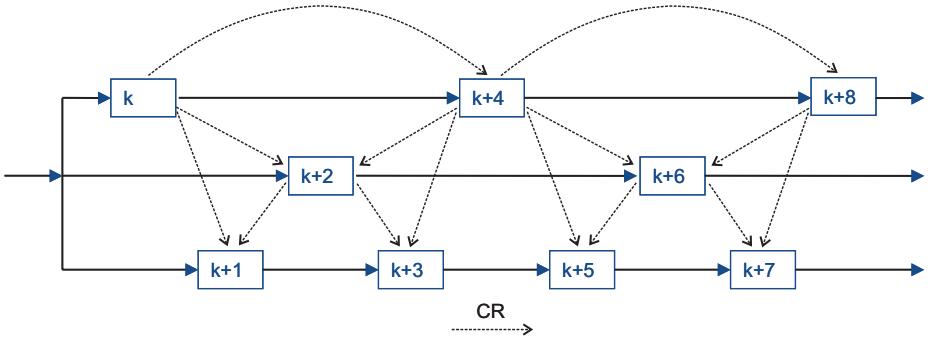

Figura 2.4: Struttura gerarchica dei livelli temporali

$k+8$, ..., il secondo dalle restanti frame pari, il terzo da tutte le frame dispari.

Le frame sono raggruppate 4 alla volta e quindi codificate: si codificano le frame da $k+1$ a $k+4$, poi quelle da $k+5$ a $k+8$ e così via. La codifica della frame $k+4$ non può avvenire usando come riferimento nessuna delle frame $k+1$, $k+2$ e $k+3$, perché chi riceve solo il primo livello non ne dispone. Allora si usa come riferimento per il CR la frame k. Questo assicura che il primo livello temporale può essere decodificato indipendentemente dagli altri.

La frame $k+2$ può essere codificata dopo la $k+4$. Allora è possibile effettuare un CR bidirezionale, usando come riferimenti le frame k e $k+4$ (ma non $k+1$ e $k+3$) in modo analogo alla tecnica di MC bidirezionale introdotta in MPEG (vedere pagina 48). Infine, le due frame del terzo livello temporale possono essere codificate per ultime, ed entrambe con il CR bidirezionale.

Questa tecnica garantisce la possibilità di decodificare il flusso anche con frame rate ridotto ad un quarto o alla metà di quello originale senza nessuna ulteriore elaborazione.

Osserviamo che il rapporto di compressione tipicamente cresce al crescere del livello temporale, sia perché passando dal primo al secondo ed al terzo livello si introduce il CR bidirezionale, sia perché le frame di riferimento diventano via via temporalmente più vicine a quella da codificare, il che aumenta la probabilità che esse siano simili.

2.5 La scalabilità in risoluzione

La scalabilità in risoluzione consiste nella possibilità di estrarre dal flusso codificato delle sequenze video a risoluzione diversa.

In ingresso abbiamo una sequenza video con un livello di risoluzione fissato detto livello *intermedio*. L'obiettivo è quello di generare un flusso codificato dal quale sia possibile estrarre la sequenza ad una risoluzione inferiore, uguale o superiore a quella originaria. A tal fin si usa la tecnica di codifica piramidale, di cui in figura 2.5 è illustrato il codificatore, e, in figura 2.6, il decodificatore.

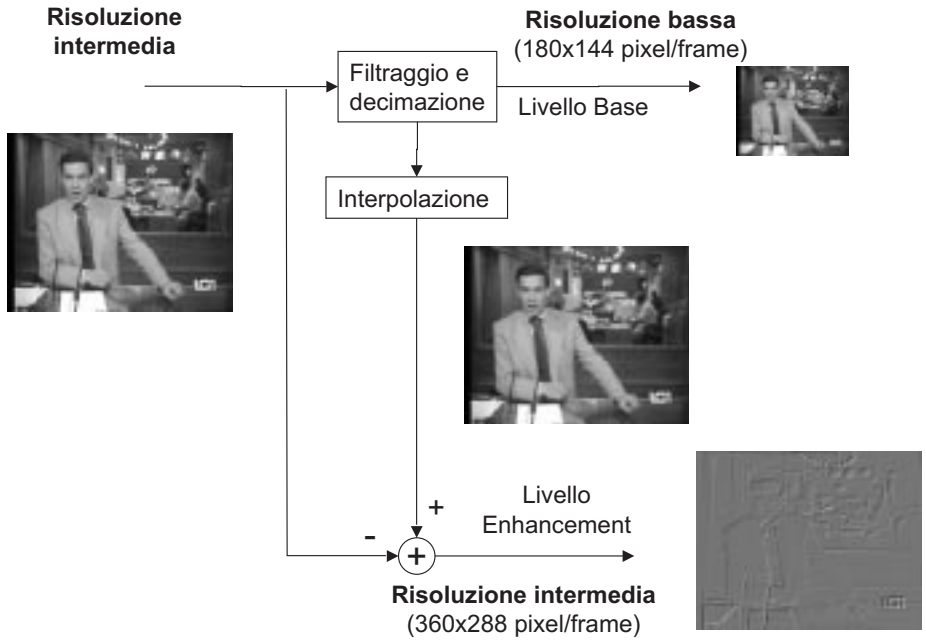

Figura 2.5: Codifica piramidale: trasmettitore

Cominciamo a descrivere il codificatore. Per ottenere un'immagine a risoluzione più bassa è sufficiente decimare i campioni del segnale di ingresso, eventualmente realizzando il filtraggio antialiasing prima del campionamento. Questa è un'operazione onerosa computazionalmente, tuttavia si è appurato che, usando un semplice filtro che fa la media di quattro pixel adiacenti, le prestazioni migliorano notevolmente e la complessità aumenta in modo compatibile con le applicazioni in tempo reale.

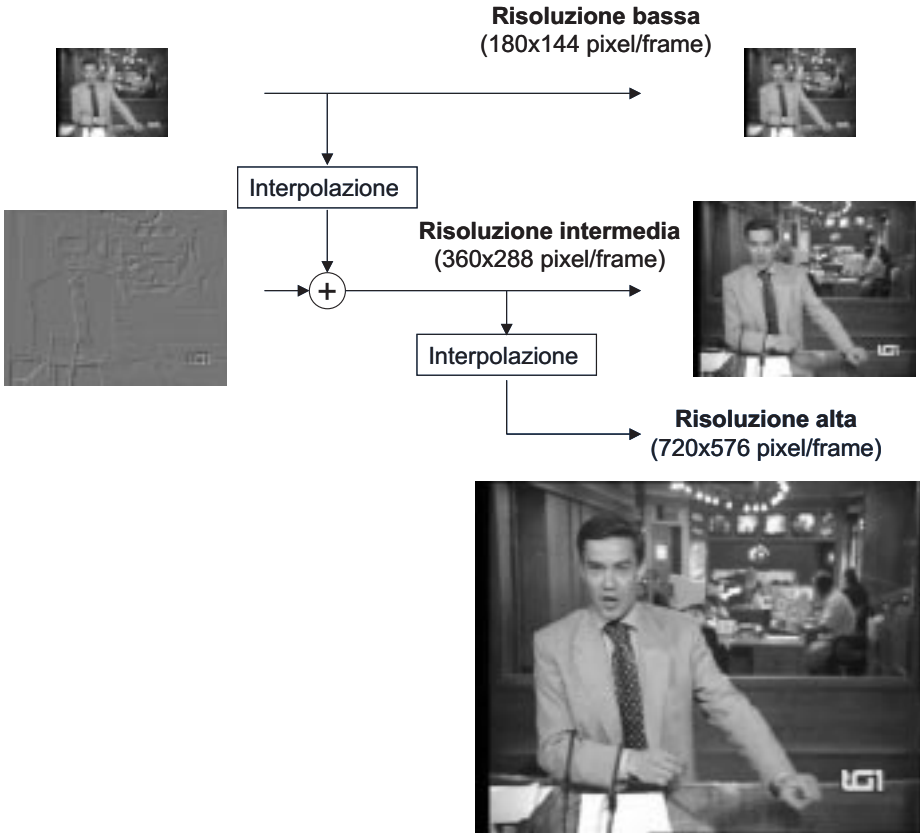

Figura 2.6: Codifica piramidale: ricevitore

La sequenza di immagini a bassa risoluzione costituisce il cosiddetto livello *base* del codec. L'utente che riceve solo tale livello è in grado di ottenere direttamente la sequenza con la risoluzione minima. Se si vuole la sequenza alla risoluzione originaria è necessario ricevere un ulteriore livello, detto *enhancement*. Per codificare l'immagine a risoluzione originaria, si sfrutta il fatto che il decodificatore ha già ricevuto l'immagine al livello base, per cui si può realizzare una codifica predittiva del livello enhancement. La predizione viene fatta semplicemente interpolando l'immagine del livello base, sicché, valutando la differenza tra questa immagine e quella originale, si ottiene l'errore di predizione.

Il funzionamento del ricevitore è abbastanza semplice (si veda la figura 2.6): il flusso al livello base può essere direttamente deco-

dificato, e inoltre da esso si ricava l'immagine interpolata, a cui si somma l'errore di predizione, in modo da ottenere l'immagine alla risoluzione originaria. Se si dispone di abbondanti risorse di potenza di calcolo è anche possibile aumentare la risoluzione, operando un'ulteriore interpolazione dell'immagine al livello enhancement.

2.6 La scalabilità in qualità

La scalabilità in qualità (o in PSNR) consiste nella possibilità di ricevere solo una parte del flusso codificato, senza per questo dover rinunciare alla risoluzione o al frame rate, ma accettando di avere una sequenza più rumorosa, cioè meno simile all'originale. Per ottenere questo risultato si è fatto uso di codebook strutturati ad albero, normalmente utilizzati negli algoritmi di *TSVQ* (Tree-Structured Vector Quantization) [8], [11]. Nella TSVQ esistono più codebook, ognuno dei quali è associato ad un livello di un albero m-ario di profondità n, e costituisce un affinamento del codebook di livello precedente. Molto spesso, come anche nel nostro caso, si usa $m = 2$ (si veda la figura 2.7).

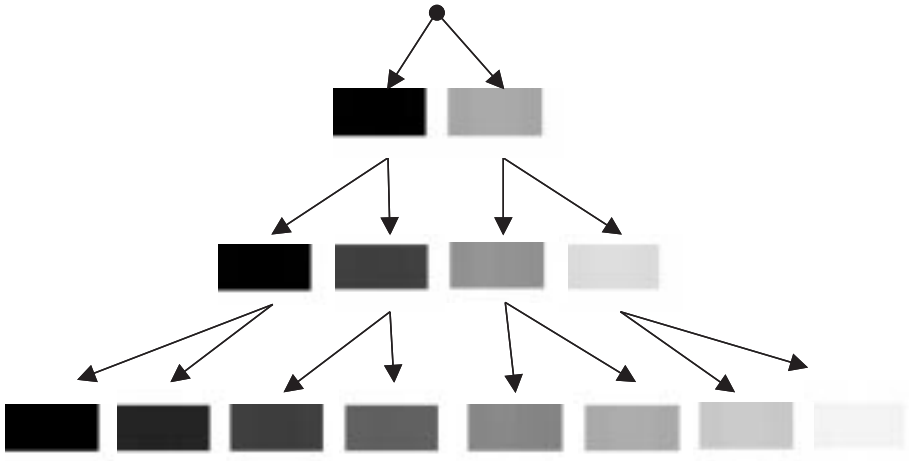

Figura 2.7: Primi livelli di un codebook ad albero ($m = 2$).

Per generare un codebook strutturato ad albero si usa una variante del cosiddetto algoritmo dello *splitting*, che a sua volta è usa-

to per generare un codebook a partire da un assegnato training set. Anche lo splitting è un algoritmo ricorsivo, il cui primo passo consiste nel trovare il centroide della sequenza di training. Tale vettore, che costituisce anche il codebook di livello zero, viene *scisso* in una coppia di vettori (la fase di "splitting"), che viene usata come codebook di partenza per il GLA, che infine fornisce il codebook di primo livello, o codebook ad un bit: $\mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2\}$. A questo punto, nell'algoritmo dello splitting, si generano altri due vettori codice, e si riapplica il GLA, ottenendo così il codebook a due bit. Invece, per generare un codebook con struttura ad albero, il training set $\mathcal{T}$ è ripartito in due sottoinsiemi $\mathcal{T}_1$ e $\mathcal{T}_2$, costituiti dai vettori di training rispettivamente più vicini alla prima o alla seconda codeword. L'algoritmo si applica ora ricorsivamente ad ognuna delle coppie $\mathcal{T}_i, \mathbf{y}_i$: il vettore codice è scisso in due, e su tale coppia si applica il GLA, usando però come training set solo $\mathcal{T}_i$. Questo significa che le due codeword risultanti, che nell'albero saranno associate ai nodi figli di $\mathbf{y}_i$, codificheranno gli stessi vettori che nel codebook precedente erano codificati dal vettore codice genitore, ma mediamente meglio, proprio perché abbiamo due vettori invece che uno. Iterando il procedimento si ottiene una famiglia di codebook con la desiderata struttura ad albero.

Vediamo come viene effettuata la codifica di un vettore nella TSVQ, per capire come si realizza la scalabilità in qualità. Il vettore d'ingresso $\mathbf{x}$ viene confrontato con i due vettori del codice di primo livello. Si individua quello che meglio rappresenta il vettore d'ingresso, che, nell'esempio riportato nella figura 2.8 è quello di destra. Allora si può scrivere il primo bit che individua la codifica di $\mathbf{x}$. Nell'esempio, si scrive "1" quando si sceglie il nodo di destra, e "0" quando si sceglie quello di sinistra. Si passa poi a confrontare il vettore $\mathbf{x}$ con i figli del nodo prescelto, e si ripete ricorsivamente lo stesso procedimento. Supponendo di avere un albero ad 8 livelli, alla fine si ottiene una stringa di 8 bit che indica quale dei 256 vettori dell'ultimo codebook codifica il vettore d'ingresso. In ricezione questa stringa viene utilizzata per ripetere sull'albero lo stesso percorso fatto in fase di codifica, fino ad individuare qual è il vettore che rappresenta $\mathbf{x}$.

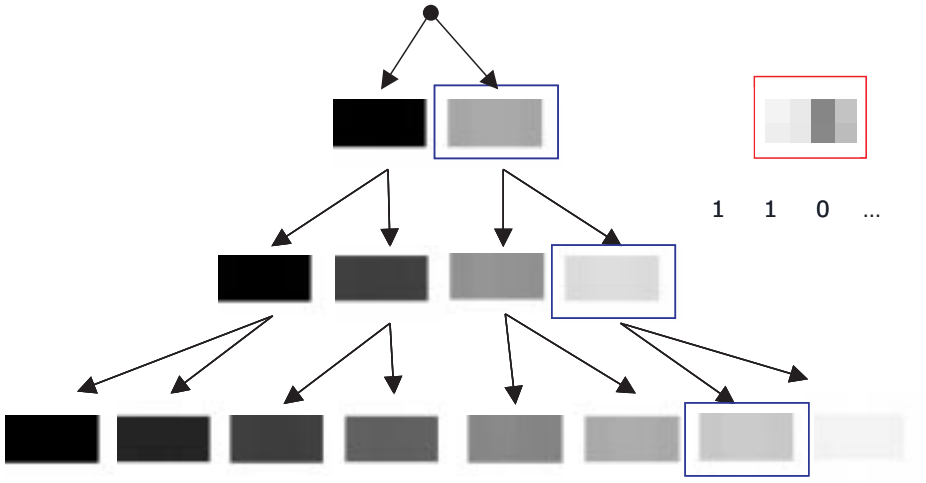

Figura 2.8: Codifica di un vettore con la TSVQ

Supponiamo però di non ricevere tutti gli otto bit della stringa, ma, ad esempio, soltanto sette. In tal caso, non potremo ripetere tutto il percorso fatto sull'albero in codifica, ma ci dovremo fermare al settimo livello. Potremo così ottenere il genitore della codeword effettiva, che è un'approssimazione più grossolana del vettore d'ingresso. Se poi riceviamo un numero di bit inferiore, dovremo accontentarci di scegliere la parola codice all'interno di codebook sempre più piccoli, e quindi in grado di descrivere sempre meno bene lo spazio dei vettori d'ingresso. In altre parole, per ridurre il tasso di codifica, basta accettare un peggioramento delle prestazioni in termini di SNR (scalabilità in qualità) che riusciamo ad ottenere attraverso la TSVQ.

Le prestazioni della TSVQ risultano inferiori a quella della VQ ordinaria, sia perché la scelta del codebook è vincolata alla struttura gerarchica in cui è ripartito lo spazio dei vettori d'ingresso, sia perché la strategia di codifica basata sulla scansione dell'albero non garantisce di trovare il vettore codice effettivamente più vicino al vettore d'ingresso.

Vediamo adesso come integrare le tecniche di HVQ e di TSVQ per avere un algoritmo di compressione spaziale scalabile in qualità ed a bassa complessità. La tecnica risultante è detta *TS-HVQ* [8].

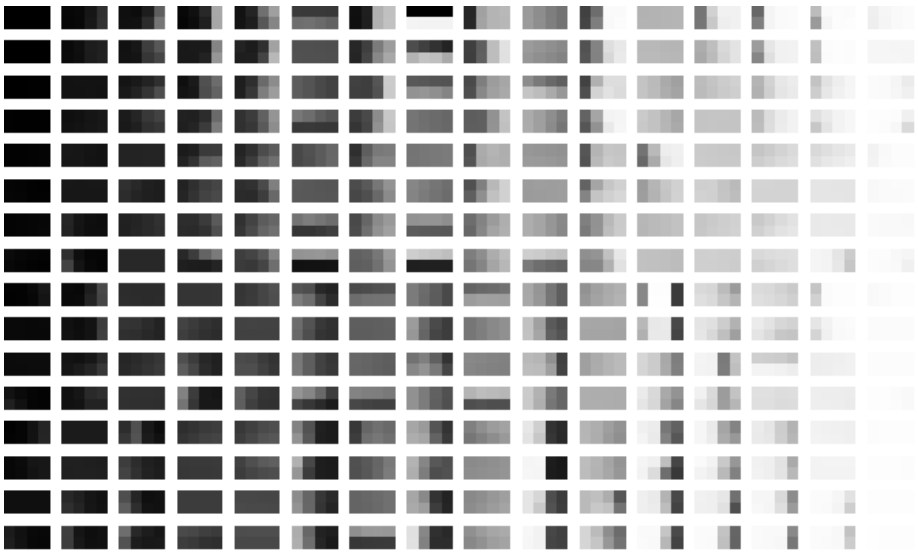

Figura 2.9: Codebook ad 8 bit con struttura ad albero

La struttura della TS-HVQ è la stessa della HVQ, con la differenza che i codebook dell'ultimo stadio devono essere costruiti con la TSVQ. Le tabelle si costruiscono come nell'HVQ ordinaria, ma all'ultimo stadio bisogna generare tante tabelle quanti sono i livelli presenti nell'albero.

La codifica nella TS-HVQ viene realizzata nello stesso modo in cui si realizza nella HVQ. In fase di decodifica, invece, a seconda di quanti bit si ricevono, si deve usare una diversa tabella, in modo da simulare la tecnica di decodifica della TSVQ. Così facendo si riesce ad implementare una tecnica di codifica che ha la scalabilità della TSVQ e la semplicità della HVQ. Tuttavia sia l'una che l'altra tecnica, come abbiamo visto, introduce un certo degrado delle prestazioni.

2.7 Il CR sugli indirizzi

La TSVQ introduce come "effetto collaterale" la proprietà di *ordinamento* dei codebook, che consiste nel fatto che vettori codice con indirizzo vicino sono simili tra loro, e che deriva dal modo in cui il codebook è costruito. Infatti le codeword con indirizzo vicino sono

state generate usando un sottoinsieme del training set relativamente piccolo e che comunque è costituito da vettori abbastanza simili tra loro, in quanto appartenenti ad una stessa regione di decisione per un codebook di livello superiore. Questa proprietà è messa in evidenza nella figura 2.9, dove è rappresentato il codebook con struttura ad albero per il livello base. Le codeword sono ordinate in colonne: si nota subito che quelle vicine sono anche simili tra loro.

Da queste considerazioni nasce l'idea di effettuare il CR confrontando gli indirizzi delle codeword, piuttosto che i valori di luminanza dei pixel [6]. Più precisamente, si opera così: la frame corrente viene quantizzata e poi scandita, e ogni codeword viene confrontata con quella nella stessa posizione nella frame di riferimento. Il confronto viene fatto però unicamente sulla base degli indirizzi delle codeword, assumendo che se due codeword hanno indirizzi vicini, sono anche simili, e quindi si può evitare di trasmettere la nuova. Questa tecnica caratterizza il codec proposto rispetto a quelli descritti in letteratura.

È possibile inoltre usare macroblocchi di $n \times m$ codeword, ed effettuare il test del CR su di essi piuttosto che sulle singole codeword, in modo da ridurre il numero di bit di sincronia (uno ogni nm codeword invece che uno ogni codeword). La distanza tra due macroblocchi $\mathbf{A}$ e $\mathbf{B}$ è valutata come "energia" delle differenze d'indirizzo:

$$d(\mathbf{A}, \mathbf{B}) = \sum_{\substack{i=0,\ldots,n \\ j=0,\ldots,m}} (A_{i,j} - B_{i,j})^2 \tag{2.5}$$

dove $A_{i,j}$ $[B_{i,j}]$ è l'indirizzo della codeword i, j nella matrice $\mathbf{A}$ $[\mathbf{B}]$.

Il vantaggio di questa tecnica è che in questo modo è i calcoli necessari per effettuare il CR su ogni macroblocco codeword si riducono da mnk prodotti e mnk sottrazioni (k è la dimensione di un vettore) ad mn prodotti ed mn sottrazioni. Questo vantaggio computazionale si paga con una perdita di sensibiltà su quello che è l'effettivo errore introdotto dal CR. In altre parole, anche se la metrica calcolata è piccola, non possiamo essere sicuri che la differenza

tra i macroblocchi sia altrettanto piccola (potrebbero essere capitate delle codeword vicine come indirizzo, ma abbastanza diverse), e viceversa.

Visto che nel nostro progetto il problema della complessità è centrale, è sicuramente opportuno valutare quanto l'utilizzo di questa tecnica, che chiamiamo *CR sugli indirizzi*, incida sulle prestazioni del codec. In questo modo possiamo valutare se il vantaggio computazionale compensi o meno il presumibile calo delle prestazioni.

2.8 I prototipi sviluppati

Nell'ambito di uno schema generale di codifica basato sulla TS-HVQ ed il Conditional Replenishment, sono stati sviluppati due prototipi che sfruttano in modo diverso le idee emerse e discusse nella prima parte di questo capitolo. In un primo prototipo, che non è sostanzialmente diverso da quello proposto in [5], si è cercato di massimizzare la qualità del flusso video prodotto, anche a scapito della semplicità computazionale. In un secondo si è cercato di minimizzare la complessità, senza tenere conto del degrado delle prestazioni, e utilizzando l'idea innovativa del CR sugli indirizzi. I due prototipi risultanti sono stati allora denominati rispettivamente "fedele" e "veloce". Nei paragrafi che seguono ne sono illustrate in dettaglio le caratteristiche, ed analizzate approfonditamente le prestazioni, in modo da stabilire quale tra i due sfrutta meglio il trade-off tra complessità, qualità, e tasso trasmissivo.

Entrambi i prototipi accettano in ingresso delle sequenze video con le caratteristiche:

- la sequenza video d'ingresso è formata da una sequenza di frame in toni di grigio, ad una risoluzione di 720×576 pixel;
- ogni pixel è rappresentato con 8 bit, in modo da avere 256 possibili livelli di luminanza.

Inoltre i principali parametri della codifica sono uguali:

- prima di ogni elaborazione, la risoluzione è portata a 360×288 pixel per ridurre la complessità;

- per la sequenza d'uscita abbiamo una risoluzione di 180×144 pixel al livello base, una di 360×288 a livello enhancement, ed infine, una di 720×576 ad alta risoluzione;
- il frame rate massimo è di 25 frame al secondo, ma sono disponibili anche frame rate di $12,5$ e di $6,25$ frame al secondo;
- la TS-HVQ è implementata con 3 o 4 stadi in modo da garantire un rapporto di compressione (spaziale) pari a 8 o a 16.

2.8.1 Il prototipo "fedele"

In modalità "intra" (si veda la figura 2.10) il codificatore opera prima due operazioni di filtraggio e decimazione, passando dalla frame X (720×576 pixel) a Y (360×288 pixel) e a Z (280×144 pixel). La frame Z viene compressa spazialmente con la TS-HVQ, e si ottiene la matrice di indirizzi A, che rappresenta il livello base del flusso codificato. Per realizzare la codifica piramidale, la matrice A viene decodificata, ottenendo $\hat{Z}$, che è la frame di livello base disponibile in ricezione. Questa frame viene interpolata, ottenendo la predizione $\hat{Y}$ della frame di livello enhancement. Si calcola poi l'errore di predizione W, che però viene trasmesso solo se significativo, cioè se la sua energia supera una certa soglia. In caso positivo, W viene quantizzata sempre con la TS-HVQ, ma si usano altri codebook, generati conformemente alle statistiche dell'errore di predizione. È necessario inserire nel flusso codificato un bit di sincronizzazione per comunicare se l'errore è stato trasmesso oppure no.

Il funzionamento del decodificatore in modalità intra è semplice: il livello base $\hat{Z}$ viene ricostruito decodificando la matrice di indirizzi A. Per quanto riguarda il livello enhancement, si interpola $\hat{Z}$, ottenendo $\hat{Y}$; a seconda del bit di sincronizzazione l'errore di predizione è nullo oppure deve essere decodificato dal flusso di livello enhancement.

Il funzionamento del codificatore in modalità inter è illustrato in figura 2.11. Dopo il doppio sottocampionamento, si valuta, per ogni macroblocco, l'energia dell'errore di predizione rispetto alla frame di riferimento $\tilde{Z}$. Notiamo che nella figura 2.11, per non aumentare la complessità dello schema, è stato illustrato solo il caso di

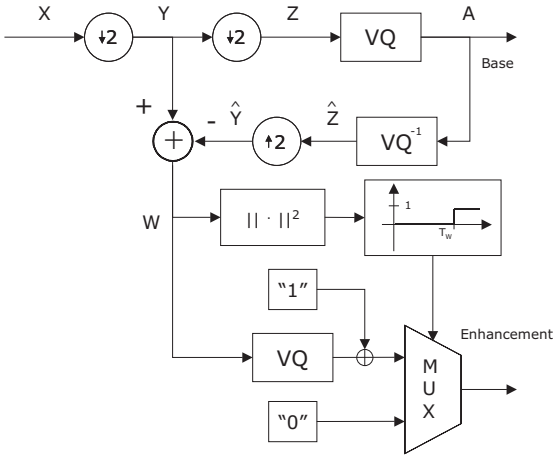

Figura 2.10: Prototipo “fedele”: codificatore in modalità intra

CR unidirezionale, ma l’estensione al caso bidirezionale è concettualmente molto semplice: tutti i confronti del CR vanno fatti due volte (relativamente alla frame precedente e alla successiva), e bisogna cambiare la semantica dei bit di segnalazione in questo modo: “0” indica il successo del CR dalla frame precedente, “10” indica il successo del CR dalla frame successiva, “11” indica l’insuccesso del CR[4]. Un altro elemento da sottolineare è che i riferimenti usati in codifica devono essere gli stessi disponibili anche in decodifica: $\tilde{Z}$ è la frame di riferimento *decodificata*.

In questo prototipo le dimensioni di un macroblocco coincidono con quelle dei vettori della VQ, cioè 2×4 o 4×4 pixel. Si valuta allora l’errore di predizione relativamente al macroblocco di riferimento (ai macroblocchi, nel caso di CR bidirezionale) e, se l’energia del (più piccolo) errore è inferiore alla soglia T_Z, il macroblocco è ritenuto predicibile, per cui si trasmette solo l’informazione di sincronia (o *side information*). Altrimenti il macroblocco va codificato e, oltre al bit di sincronia, bisogna trasmettere l’indirizzo del vettore che lo codifica.

Indipendentemente dall’esito del CR bisogna poi ricostruire la frame $\hat{Z}$ disponibile in ricezione, i cui blocchi di volta in volta coin-

[4]La sequenza di segnalazione più breve è stata assegnata al caso che, nel corso di tutte le simulazioni, si è rivelato il più probabile.

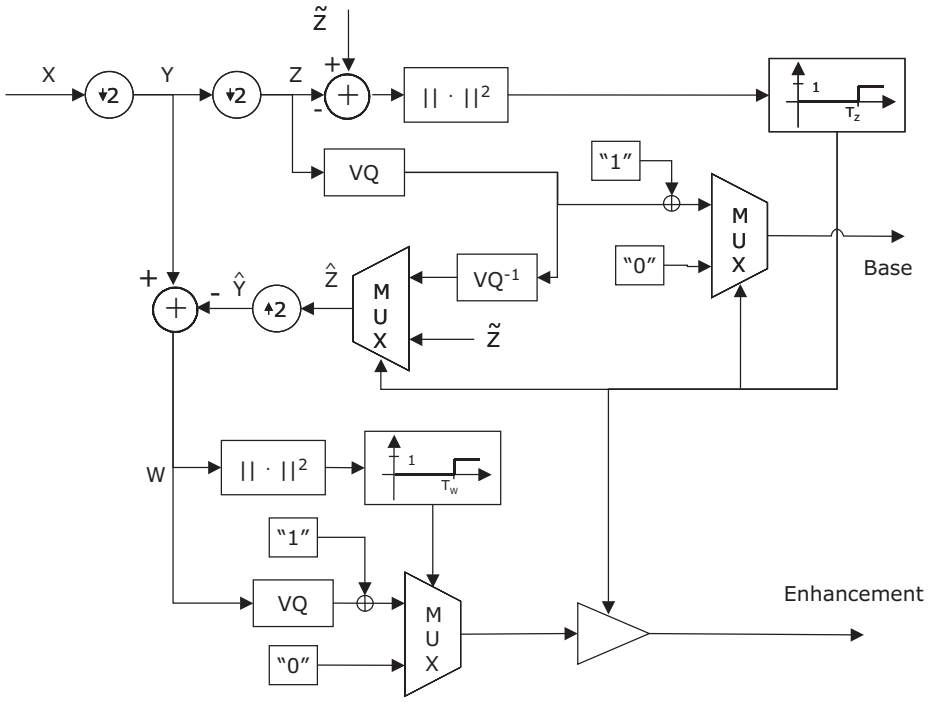

Figura 2.11: Prototipo “fedele”: codificatore in modalità inter

cideranno con quelli quantizzati di Z o di $\tilde{Z}$. La frame ricostruita $\hat{Z}$ viene interpolata per trovare $\hat{Y}$ e poi l'errore di predizione W.

Tuttavia, se il CR al livello base ha avuto successo, i corrispondenti blocchi al livello enhancement non vengono proprio codificati, perché si è ritenuto che un cambiamento significativo di un macroblocco sia rilevabile a tutte le risoluzioni, e che quindi sia inutile fare due volte il test del CR. Questo permette di risparmiare dei calcoli, ma vincola le prestazioni dei due livelli: in altre parole il valore di T_Z che permette di raggiungere il miglior compromesso tra tasso e qualità al livello base può essere diverso da quello che invece ottimizza le prestazioni al livello enhancement. In particolare, le simulazioni hanno evidenziato che al livello base è preferibile usare valori molto elevati della soglia, che permettono di ridurre significativamente il tasso senza penalizzare troppo la qualità; però, per tali valori della soglia, la qualità del livello enhancement non risulta soddisfacente, e quindi non è possibile scegliere un valore della soglia che permetta

far lavorare nel modo migliore il codificatore ad entrambi i livelli.

Se invece il CR al livello base non ha avuto successo, si prosegue con la codifica dell'errore di predizione, che viene fatta allo stesso modo del caso intra, cioè valutando se tale errore ha un'energia tale da giustificarne la trasmissione.

Il decodificatore in modalità inter si occupa innanzitutto del livello base. Se la side information indica il successo del CR, il blocco del livello base viene recuperato dalla frame precedente, e il livello enhancement viene valutato semplicemente interpolando il livello base. Se il CR non ha avuto successo, si decodifica il blocco al livello base e poi si passa al livello enhancement. Il blocco base viene interpolato; se i bit di sincronia del livello enhancement indicano che l'errore di predizione è stato trasmesso, lo si decodifica e lo si aggiunge al blocco interpolato. Altrimenti il blocco al livello enhancement è costituito solo da $\hat{Y}$.

2.8.2 Prestazioni del prototipo "fedele"

Nelle figure 2.12 e 2.13 sono riportate le prestazioni del prototipo ricavate utilizzando come ingresso una sequenza di prova di 61 frame, con un frame rate di 25 fps, per una durata totale di circa 2,5 secondi. Sull'asse delle ascisse è riportato il tasso medio necessario per trasmettere la sequenza in tempo reale; sull'asse delle ordinate la qualità della sequenza decodificata, espressa tramite il PSNR. Nelle figure vengono mostrate le prestazioni in due casi: quello in cui la VQ operi su blocchi 2×4 e quello in cui operi su blocchi 4×4; a questi due casi corrisponde una compressione spaziale con rapporto pari a 8 e a 16 rispettivamente. I grafici sono stati ottenuti facendo variare la soglia T_Z che stabilisce qual è l'errore tollerabile per il CR: ogni punto nel grafico corrisponde ad un diverso valore di T_Z, da cui dipendono il tasso ed il PSNR.

Queste simulazioni evidenziano nettamente che il codificatore funziona meglio quando la VQ opera su blocchi di dimensione 2×4; evidentemente, già al terzo stadio della HVQ la maggior parte della ridondanza spaziale è stata rimossa, e aumentando ulteriormente il rapporto di compressione si ha solo un drastico calo delle presta-

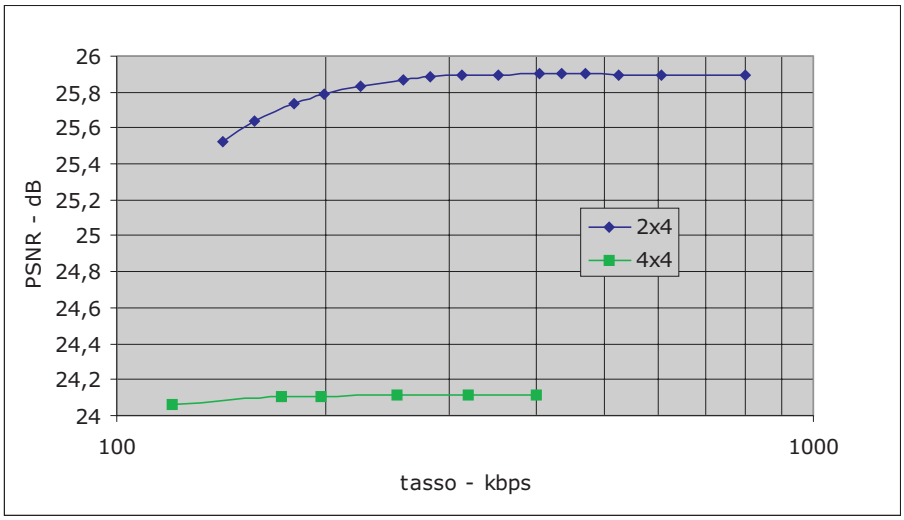

Figura 2.12: Prototipo “fedele”: prestazioni al livello base

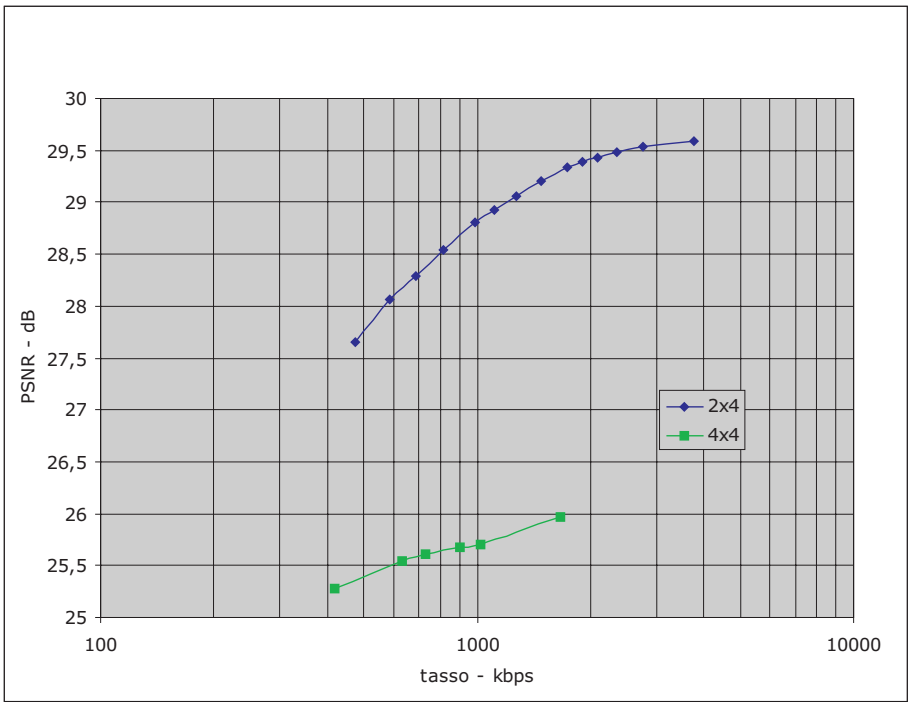

Figura 2.13: Prototipo “fedele”: prestazioni al livello enhancement

zioni senza un'adeguata riduzione del tasso. Si noti invece come, al variare della soglia del CR (cioè dell'entità della compressione temporale), la qualità della sequenza video, espressa in termini di PSNR, vari molto poco. Di contro, la qualità soggettiva della sequenza cala sensibilmente in corrispondenza dei valori più alti della soglia. Questa è una dimostrazione di quanto sia poco opportuno affidarsi solo ad una misura della qualità di tipo analitico.

Da queste prime analisi risulta evidente che:

- si può operare in corrispondenza di valori abbastanza alti della soglia, visto che il PSNR si riduce di poco, ma non in corrispondenza dei valori *più* alti, visto che in corrispondenza di essi la qualità soggettiva della sequenza video non è accettabile;
- è opportuno considerare solo blocchi di dimensione 2×4.

Bisogna ora valutare come incide sulle prestazioni il parametro T_W, che individua la minima energia che deve avere l'errore di predizione della codifica piramidale affinché possa essere ritenuto significativo. Nella figura 2.14 sono illustrate le prestazioni del codificatore con blocchi 2×4 al livello enhancement. Ogni curva è relativa ad un fissato valore di T_W, ed è stata ottenuta facendo variare T_Z. Si nota che piccoli incrementi iniziali di T_W influenzano in modo irrilevante le prestazioni: vengono scartati solo gli errori a bassissima energia, che sono molto pochi, e questo influenza scarsamente sia il tasso, che la qualità.

Al crescere di T_W vengono considerati rilevanti solo gli errori di predizione con energia abbastanza grande. Ciò comporta che sempre più spesso tali errori non verranno codificati, consentendo una riduzione del tasso medio necessario per trasmettere la sequenza, ma causando anche una riduzione della qualità. Il grafico di figura 2.14 mette in luce come, al crescere di T_W, la degradazione della qualità della sequenza non sia adeguatamente compensata dalla riduzione del tasso. Se ne conclude che le prestazioni migliori si hanno per $0 \leq T_W \leq 100$, ma allora tanto vale non eseguire il test sull'energia risparmiando i calcoli relativi, e trasmettere sempre l'errore.

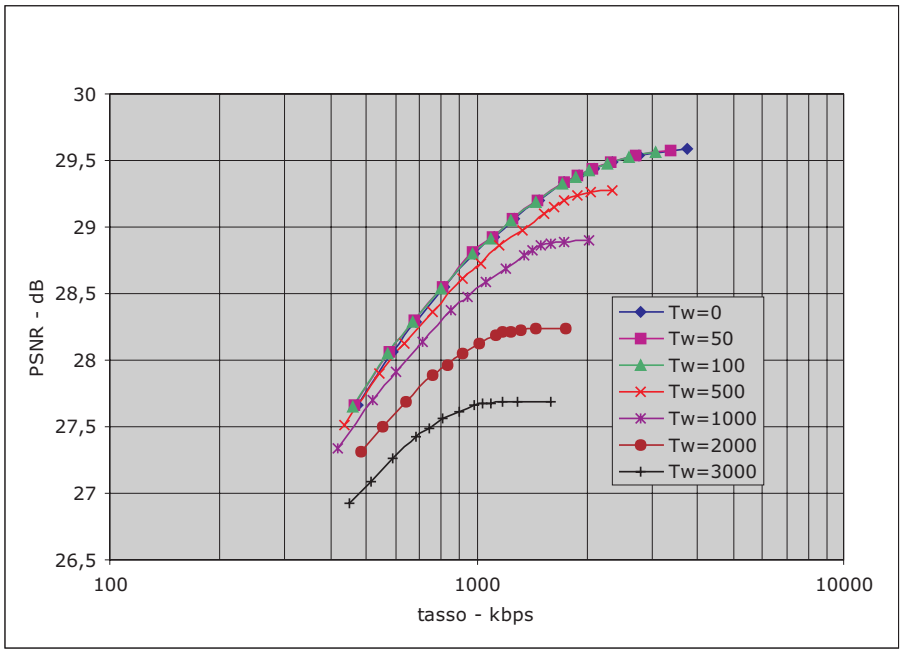

Figura 2.14: Prototipo "fedele", livello enhancement, blocchi 2×4: prestazioni al variare di T_W.

Altri elementi da evidenziare sono i tempi di codifica e decodifica e l'impatto dei bit di sincronia nel flusso dati. I tempi di codifica sono abbastanza elevati: si è rilevato un valore medio di 75 ms [5] per frame in corrispondenza di $T_Z = 0$ e $T_W = 0$. In queste condizioni è possibile arrivare ad un frame rate di circa 13 fps, inferiore all'obiettivo di 25 fps. Per la decodifica è stato rilevato invece un tempo medio di soli 17 ms: questo risultato era prevedibile, visto che l'unica operazione complessa effettuata dal decodificatore è l'interpolazione necessaria per lo schema piramidale. Se poi si considera che il test sull'energia dell'errore può essere evitato, il tempo medio di codifica si riduce fino a circa 55 ms, il che consente un frame rate di circa 18 fps.

Per quanto riguarda invece i bit di overhead, le simulazioni hanno mostrato che, al crescere della soglia per il CR, essi cominciano

[5] Tutti i tempi riportati sono stati valutati su di un personal computer con processore a 800MHz, 256 MB di RAM, e sistema operativo Linux.

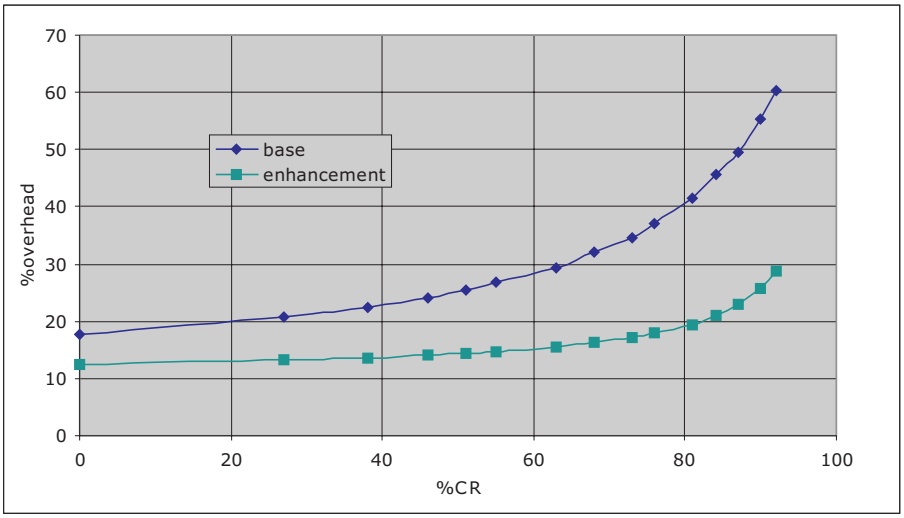

Figura 2.15: Prototipo “fedele”: percentuale di bit di overhead.

a diventare una frazione preponderante del flusso dati (si veda la figura 2.15). La spiegazione di questo fatto è semplice: noi utilizziamo, al livello base, un bit di overhead per ogni blocco di 2×4 pixel; se la soglia del CR è alta, per molti blocchi avremo solo il bit di overhead, e solo per alcuni anche gli 8 bit che individuano il vettore codice. Per il livello enhancement vale lo stesso discorso, relativamente però alla soglia T_W. Si tenga conto inoltre che qui abbiamo un bit di overhead ogni 4 blocchi di 2×4 pixel, per cui l'overhead è inferiore a quello del livello base.

2.8.3 Il prototipo “veloce”

In modalità intra, il funzionamento di questo prototipo è molto simile all'altro. L'unica differenza è che l'errore di predizione viene sempre trasmesso, per evitare di doverne valutare ogni volta l'energia. Lo schema è riportato in figura 2.16.

Il funzionamento del decodificatore in modalità intra è ovvio: la matrice di indirizzi A è decodificata, fornendo $\hat{Z}$, che, interpolata, da $\hat{Y}$, a cui si somma l'errore di predizione decodificato.

Il funzionamento in modalità inter è invece diverso rispetto al prototipo “fedele” (si veda la figura 2.17). Si noti che anche in que-

sto caso la figura fa riferimento al caso di CR unidirezionale: valgono quindi le stesse considerazioni fatte per l'altro prototipo. Dopo il doppio sottocampionamento (e filtraggio), la frame di livello base viene comunque quantizzata: si ottiene così il blocco A di indirizzi. Questi vengono raggruppati in macroblocchi di dimensione 3×3; si calcola poi la differenza tra questi macroblocchi e quelli nella stessa posizione nella frame di riferimento quantizzata $\tilde{A}$, ed infine l'energia di tale differenza. In questo prototipo si usa quindi la tecnica di CR sugli indirizzi, che, come evidenziato nel paragrafo 2.7, consente una riduzione della complessità al prezzo di un minor controllo sull'effettiva distorsione introdotta dal CR. Si noti che la matrice A è 8 volte più piccola di Z è quindi il test per il CR richiede 8 volte meno calcoli; inoltre si usa un bit di sincronizzazione ogni $3 \times 3 \times 2 \times 4 = 72$ pixel, per cui è ragionevole aspettarsi che in questo caso l'incidenza dell'overhead sul tasso totale sia piuttosto ridotta.

Il test sul CR viene completato confrontando l'energia dell'errore sui macroblocchi con la soglia T_Z: se è inferiore, il CR ha successo, e non si trasmette il nuovo macroblocco, ma solo l'informazione di sincronia. Sulla base di questo test si stabilisce quindi qual è il macroblocco disponibile in ricezione per la decodifica; questo viene quindi decodificato, permettendo di ottenere la frame $\hat{Z}$, che viene interpolata e sottratta da Y, in modo da ottenere l'errore di predizione W. Se, per il macroblocco corrente, il CR al livello base aveva ottenuto successo, tale errore non si trasmette (si manda solo l'informazione di sincronia); altrimenti W viene quantizzato e trasmesso.

Per la decodifica, al livello base si valuta la side information: se indica il successo del CR, il blocco corrente è recuperato dalla frame di riferimento, sia per il livello base che per il livello enhancement; altrimenti è prelevato dal flusso d'ingresso. Una volta ottenuti i macroblocchi, questi vengono decodificati, ottenendo $\hat{Z}$ e W; poi il primo viene interpolato e sommato al secondo, formando il blocco per il livello enhancement.

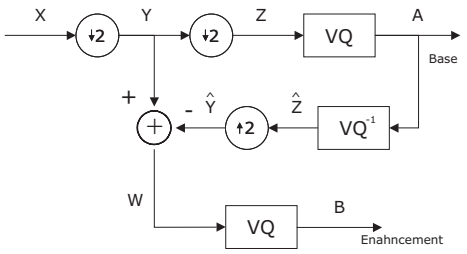

Figura 2.16: Prototipo “veloce”: codificatore in modalità intra

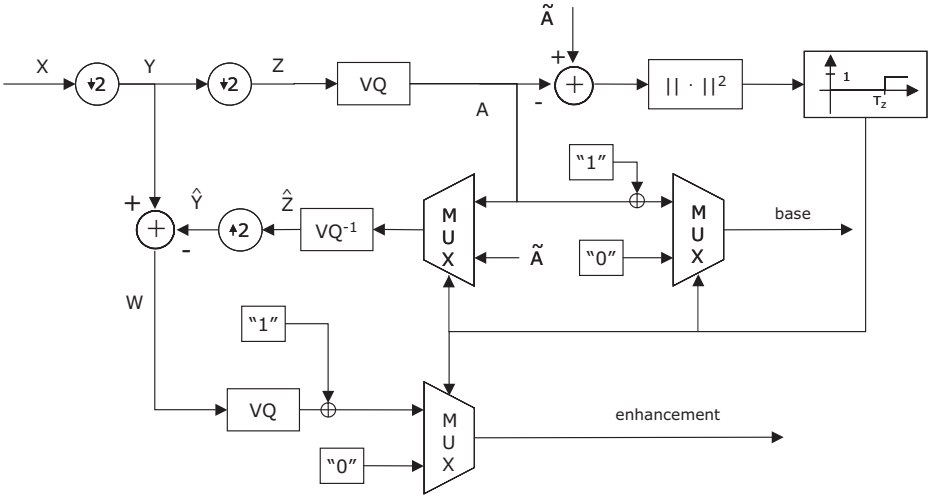

Figura 2.17: Prototipo “veloce”: codificatore in modalità inter

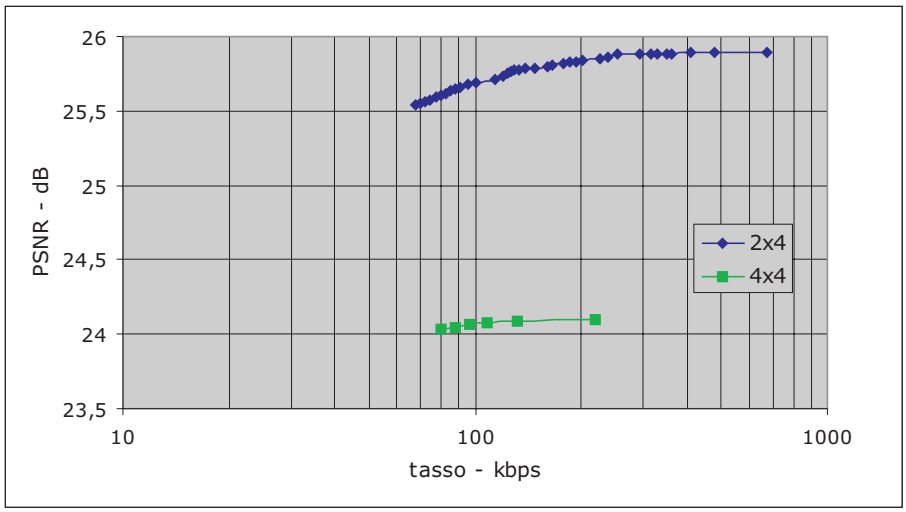

Figura 2.18: Prototipo “veloce”: prestazioni al livello base

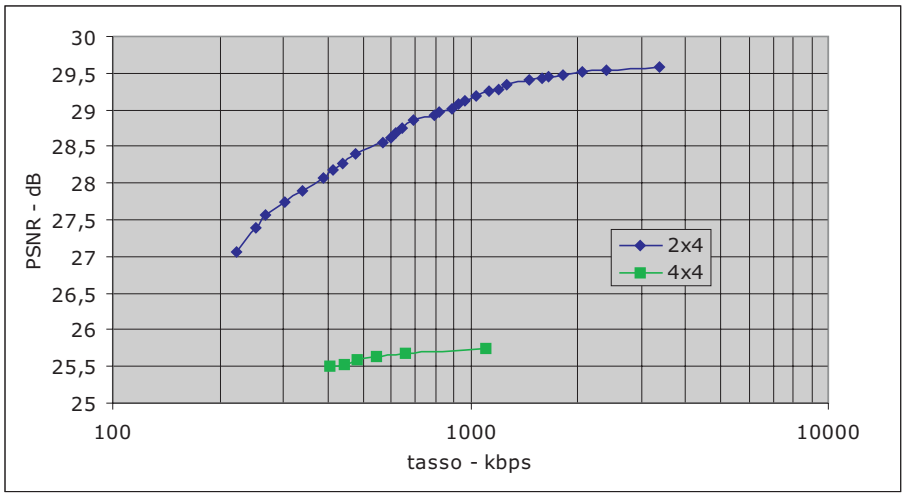

Figura 2.19: Prototipo “veloce”: prestazioni al livello enhancement

2.8.4 Le prestazioni del prototipo “veloce”

Nelle figure 2.18 e 2.19 sono illustrate le prestazioni del prototipo “veloce”, ottenute usando sempre la stessa sequenza di prova. Si nota, anche in questo caso, l’opportunità di usare blocchi di dimensioni 2×4 piuttosto che 4×4. Un’altra somiglianza nel comportamento dei due prototipi riguarda l’effetto del valore della soglia del CR sulle prestazioni. Anche in questo caso, piccoli incrementi iniziali della soglia consentono una forte riduzione del tasso senza grosse perdite di qualità: in questa fase stiamo eliminando gran parte della ridondanza temporale. Al crescere ulteriore della soglia, la riduzione del tasso si fa sempre più esigua, mentre il peggioramento delle qualità comincia ad essere sempre più significativo, non tanto in relazione al PSNR, che comunque varia in totale di frazioni di dB, quanto invece riguardo alla qualità soggettiva. Dagli esperimenti appare comunque chiaro che, per il livello base si può operare con i valori più alti della soglia con prestazioni ancora accettabili; per il livello enhancement invece non è opportuno usare i valori più alti della soglia, perché si ha un degrado troppo alto delle prestazioni. Il valore effettivamente usato della soglia deve allora essere il frutto di un compromesso tra queste due esigenze.

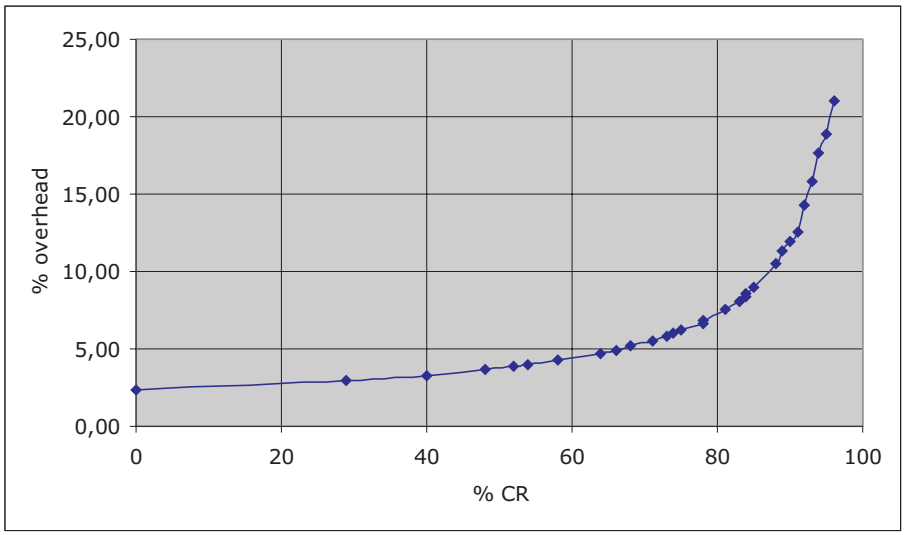

Figura 2.20: Prototipo “veloce”: percentuale di bit di overhead

Per quanto riguarda l'impatto dei bit di sincronizzazione sul tasso, dalla figura 2.20 si deduce che la gestione del Conditional Replenishment per blocchi di codeword ha consentito di ridurre a valori accettabili la percentuale di bit di overhead rispetto al totale. Si noti inoltre che tale percentuale è la stessa al livello base e al livello enhancement.

Un altro punto interessante che è stato analizzato riguarda l'effettiva capacità dell'implementazione di trarre vantaggio dalla ridotta complessità computazionale. Gli esperimenti hanno messo in luce una riduzione dei tempi di calcolo per frame di circa il 50%[6] rispetto al prototipo "fedele". Si arriva cioè ad un tempo di codifica per frame di circa 37 ms mentre la decodifica richiede non più di 17 ms. Questi valori sono compatibili con il funzionamento in tempo reale.

Si è poi analizzato l'effetto sulle prestazioni delle dimensioni dei macroblocchi usati per il CR. Nelle figure 2.21 e 2.22 sono stati riportati i risultati delle simulazioni al variare delle dimensioni del macroblocco, espresse in termini di numero di vettori codice. Le combinazioni scelte per le simulazioni sono state quelle che permettono di coprire tutta la frame base di 144×180 pixel (cioè 72×45 vettori codice) con macroblocchi delle stesse dimensioni. Usare macroblocchi molto piccoli significa valutare con maggiore precisione le aree dove c'è attività; tuttavia bisogna tenere conto che nelle sequenze video il movimento è ben localizzato, quindi è inutile avere macroblocchi di dimensioni troppo ridotte. Inoltre, con macroblocchi grandi, la frazione di bit di sincronizzazione sul flusso complessivo diventa sempre più piccola.

Le figure mettono in luce che la scelta migliore è quella di usare macroblocchi formati da 3×3 vettori codice, corrispondenti a 6×12 pixel. Questa scelta garantisce infatti, per livello enhancement sempre le prestazioni migliori, e per il livello base le prestazioni migliori (o confrontabili con le migliori) per i valori d'interesse del tasso (cioè al di sotto dei 150 kbps). D'ora in poi le dimensioni dei macroblocchi si sottintendono sempre pari a 3×3.

[6] Sempre con la stessa configurazione di cui alla nota 5 di pagina 78.

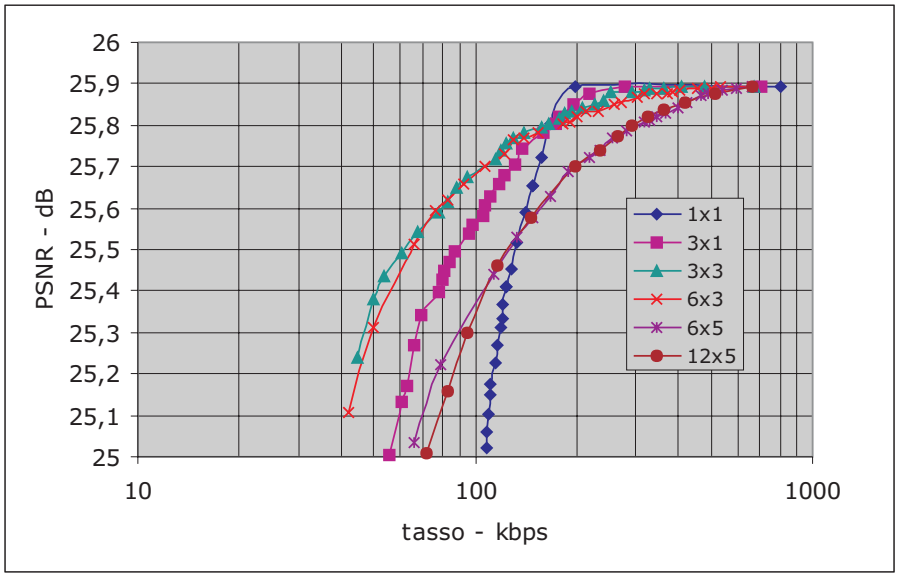

Figura 2.21: Prestazioni del prototipo "veloce" al variare delle dimensioni dei macroblocchi : livello base

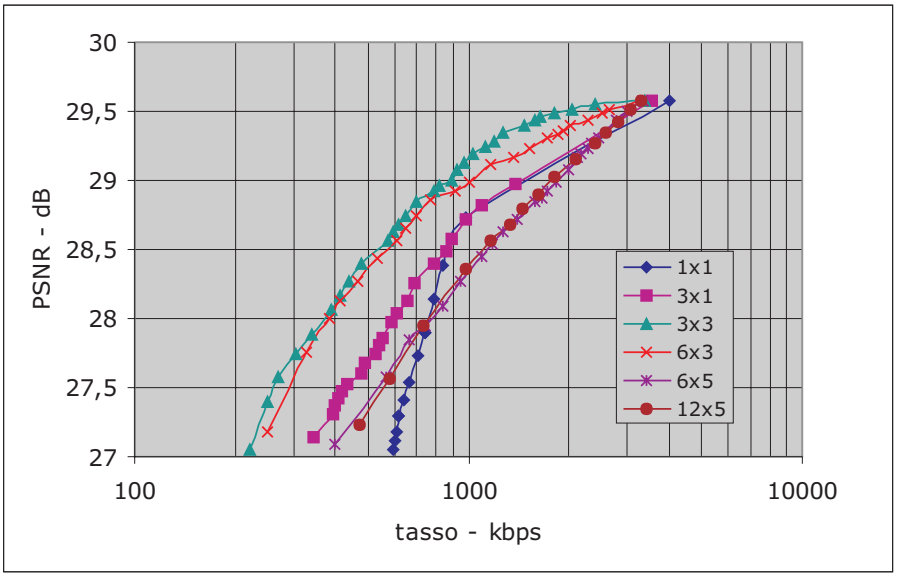

Figura 2.22: Prestazioni del prototipo "veloce" al variare delle dimensioni dei macroblocchi : livello enhancement

2.8.5 Il confronto tra i due prototipi

Nelle figure 2.23 e 2.24 sono confrontate le prestazioni dei due prototipi, relativamente alla sequenza di prova, usando per la VQ vettori di dimensioni 2×4, sia per il livello base, sia per il livello enhancement. Si nota allora che il temuto calo delle prestazioni del prototipo veloce rispetto al fedele è praticamente inesistente, anzi, quasi sempre è il prototipo veloce ad avere le prestazioni migliori. Ciò può essere spiegato con il fatto che il secondo prototipo gestisce meglio i bit di sincronizzazione, e così, a parità di tasso, possono essere spesi più bit per la codifica dei macroblocchi, e questo finisce col compensare la tecnica semplificata di CR.

In conclusione, tenendo conto del vantaggio computazionale del prototipo veloce, della primaria importanza del requisito di bassa complessità, dello scarso o inesistente peggioramento delle prestazioni del prototipo veloce rispetto a quello fedele, nello sviluppo del progetto si è optato per il prototipo veloce, che presenta ancora dei margini nel miglioramento delle prestazioni.

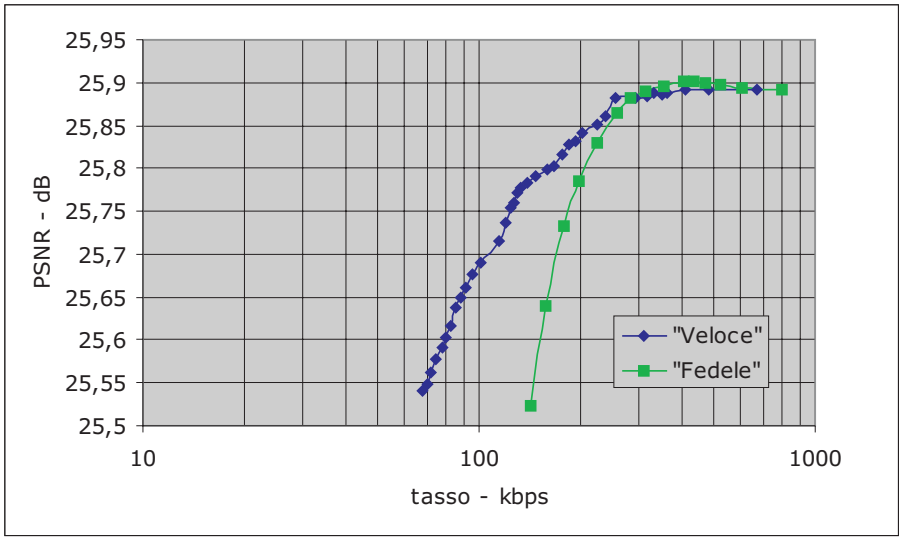

Figura 2.23: Confronto tra le prestazioni dei prototipi: livello base

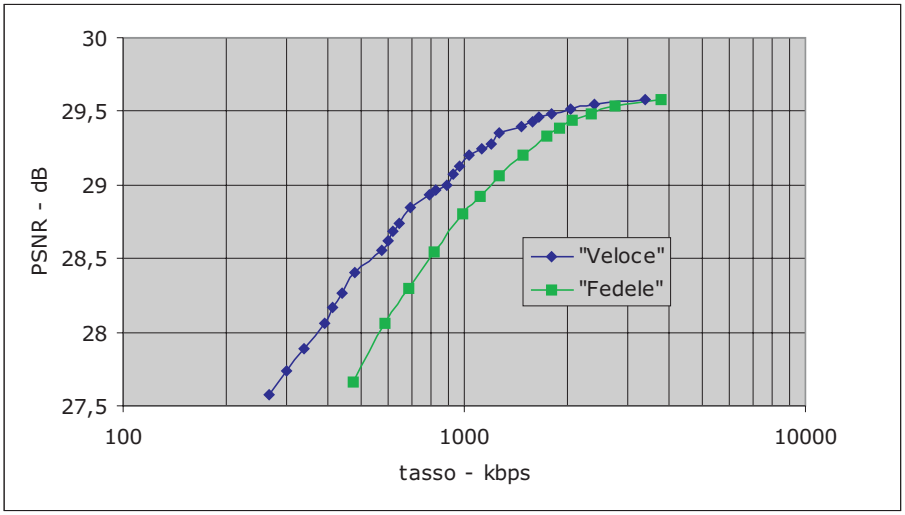

Figura 2.24: Confronto tra le prestazioni dei prototipi: livello enhancement

Capitolo 3

L'ottimizzazione del codificatore

Il prototipo "veloce" descritto nel capitolo precedente soddisfa i requisiti di bassa complessità e di scalabilità e, contemporaneamente, consente di produrre una sequenza video di qualità accettabile a basso bit rate.

Questo codificatore può essere migliorato ulteriormente sotto diversi punti di vista: da un lato si può pensare di apportare delle modifiche all'algoritmo di compressione, dall'altro c'è la possibilità di compattare il flusso di dati prodotto dal codificatore. A questo proposito osserviamo che tale flusso è costituito da due parti con caratteristiche distinte, vale a dire i bit di sincronizzazione e i bit che rappresentano gli indirizzi delle codeword; entrambi questi sotto-flussi possono essere compressi, e ciò avviene con tecniche diverse.

Nel seguito del capitolo vengono descritte le varie modifiche che sono state introdotte al codificatore per migliorarne le prestazioni. Lo schema di partenza è quello del prototipo "veloce", usato con vettori di dimensione 2×4 pixel e macroblocchi formati da 3×3 vettori, cioè 6×12 pixel.

3.1 Conditional Replenishment indipendente a livello enhancement

La tecnica di CR usata nel codificatore prevede di decidere se un blocco debba essere trasmesso o meno considerando solo quello che succede al livello base, ed estendendo poi la decisione al livello superiore. Questa strategia è giustificata da una considerazione di natura sperimentale: se un macroblocco del livello base non varia significativamente tra una frame e l'altra, probabilmente anche i corrispondenti quattro macroblocchi del livello enhancement rimangono quasi immutati. Il vantaggio è che si evita di ripetere il test del CR al livello enhancement per il quale però questa tecnica risulta meno precisa. Inoltre si rendono tra loro dipendenti i fattori di compressione dei due livelli, entrambi legati alla soglia T_Z, con la conseguenza che, se si vuole ottenere la massima compressione possibile a livello base, si deve rinunciare alla possibilità di ottenere una sequenza di buona qualità al livello enhancement.

Risulta allora interessante capire come cambia il comportamento del codec se si rende la decisione sul CR al livello enhancement indipendente da quello che succede al livello base. Più precisamente, i macroblocchi del livello enhancement subiscono la stessa elaborazione di quelli del livello base: si calcola la differenza con i macroblocchi di riferimento e poi l'energia dell'errore, e il risultato viene confronto con una soglia T_W indipendente da T_Z. Lo schema risultante del codificatore è presentato nella figura 3.1, mentre il decodificatore rimane invariato, in quanto la semantica del flusso codificato non è cambiata[1]. Da questa modifica ci si può attendere un'aumento dei tempi di calcolo, ma anche un miglioramento delle prestazioni, visto che le decisioni al livello enhancement sono prese in modo più corretto; inoltre si guadagna un grado di libertà nella configurazione del codec, nel senso che le prestazioni a livello base dipendono *solo* da T_Z, mentre quelle del livello enhancement dipendono *soprattutto* da T_W, e quindi in pratica i due livelli possono essere configurati indipendentemente.

[1] Infatti nel prototipo iniziale i bit di segnalazione del livello enhancement sono comunque presenti .

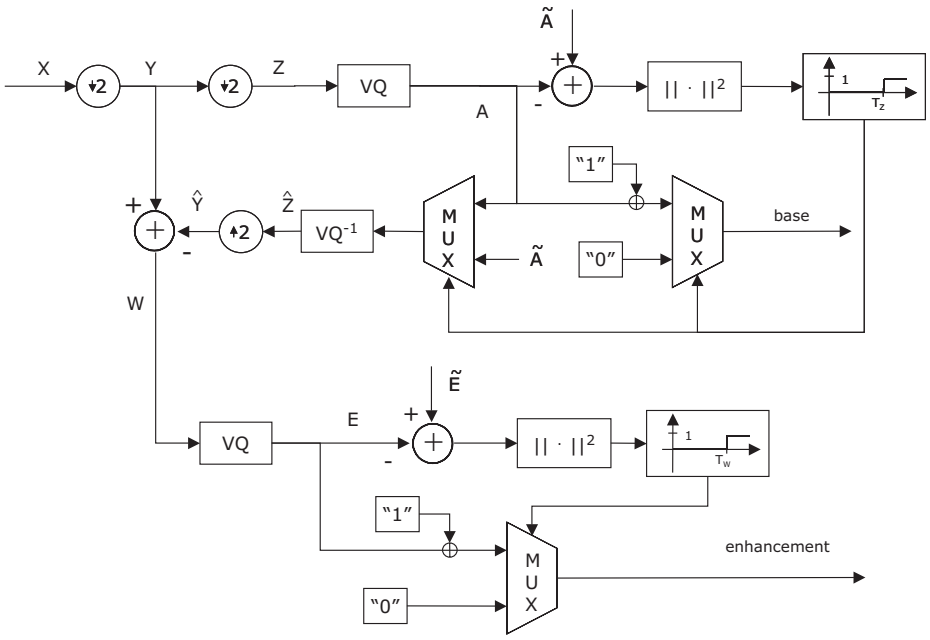

Figura 3.1: CR indipendente al livello enhancement: schema del codificatore in modalità inter.

Nella figura 3.2 sono riportati i risultati delle simulazioni del codificatore che usa due soglie, confrontate con quelle del prototipo iniziale, che usa una singola soglia. Ogni curva è stata ricavata al variare di T_Z per un fissato valore di T_W. Si nota che, soprattutto in corrispondenza dei valori più bassi del tasso, la tecnica di CR indipendente a livello enhancement è in grado di migliorare significativamente le prestazioni, consentendo riduzioni del tasso anche del 20% a parità di PSNR.

Per quanto riguarda i tempi di calcolo, le simulazioni hanno messo in luce incrementi inferiori al 20%: si passa infatti da 37 a 44 ms per frame. L'aumento dei tempi di codifica è stato tutto sommato contenuto perché la tecnica di CR sugli indirizzi è molto meno onerosa di quella sui pixel.

In conclusione, la scelta di fare un ulteriore test per il CR a livello enhancement porta due vantaggi: un miglioramento delle prestazioni, e la possibilità di sfruttare il trade-off tra tasso e qualità ad ognuno dei due livelli di risoluzione indipendentemente dall'al-

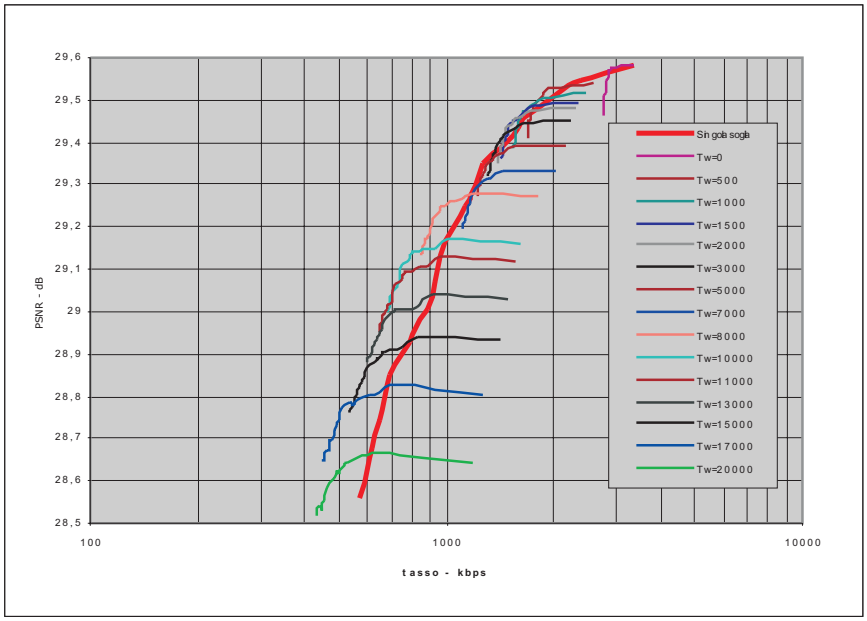

Figura 3.2: CR indipendente al livello enhancement: impatto sulle prestazioni.

tro. Il prezzo da pagare è un aumento dei tempi di codifica, che però è compatibile con il funzionamento in tempo reale. Inoltre, nel seguito, verranno illustrate delle modifiche alla tecnica di CR sugli indirizzi che rendono ancora meno oneroso il test, e quindi del tutto accettabile il sovraccarico legato al fatto di doverlo ripetere al livello enhancement.

3.2 Compattazione dei bit di side information

I bit di side information sono quelli che indicano, per ogni macroblocco, se il CR ha avuto successo oppure no. Se si considerano i bit di side information di macroblocchi vicini, sulla base del principio della *località del movimento*, possiamo affermare che questi sono tipicamente molto correlati tra loro. Infatti, in una sequenza video si possono individuare per ogni frame delle aree dove c'è attività ed altre che invece costituiscono lo "sfondo", e queste aree sono sempre

ben distinte tra loro. Possiamo quindi immaginare di raggruppare le informazioni di sincronizzazione relative ad un insieme formato da $n \times m$ macroblocchi adiacenti; tale insieme è indicato col nome di *superblocco*. È probabile allora che in una regione di sfondo la maggior parte dei bit di side information siano uguali: in tal caso, invece di trasmettere $n \times m$ bit tutti uguali tra loro, possiamo trasmetterne uno solo che indica che per tutti i macroblocchi del superblocco il CR ha avuto successo. Se invece il CR non avuto successo anche solo in un macroblocco, bisogna mandare un bit di sincronizzazione per il superblocco e tutti gli $n \times m$ bit relativi ai macroblocchi. L'idea può essere estesa anche a strutture più grandi, formate ad esempio da insiemi di superblocchi: si delinea così una struttura gerarchica nella gestione della side information. Osserviamo poi che, dal punto di vista della complessità computazionale, l'uso di questa tecnica è praticamente ininfluente, perché sostanzialmente essa modifica l'ordine di scansione dei macroblocchi e introduce un confronto in più per ogni macroblocco ed uno per ogni superblocco: il peso di queste operazioni è trascurabile rispetto ai calcoli necessari per il CR.

I parametri di questa tecnica sono il numero di livelli della gerarchia e le dimensioni dei blocchi di ogni livello. Usare molti livelli della gerarchia consente (teoricamente) di mandare pochissimi bit di sincronizzazione. In realtà, più aumentano le dimensioni del blocco, meno probabile diventa il caso in cui per tutti i macroblocchi il CR abbia successo; inoltre ogni livello della gerarchia comporta sicuramente un aumento del numero massimo[2] di bit di side information per frame. Si può fare un analogo discorso per quanto riguarda le dimensioni dei blocchi: più sono grandi, minore è la probabilità di successo del CR su tutto il blocco, ma maggiore è la compressione in caso di successo.

Passando alle simulazioni, queste hanno subito evidenziato che l'uso di più di due livelli di gerarchia è del tutto inutile, perché la probabilità di successo su un insieme di macroblocchi troppo grande diventa insignificante.

[2] È il caso in cui tutti i macroblocchi sono cambiati, e quindi bisogna trasmettere tuti i bit di side information di tutti i livelli della gerarchia.

Dimensioni	unità
144×180	pixel
72×45	codeword = 2×4 pixel
24×15	macroblocco = 3×3 codeword
$a)$ 12×5	superblocco = 2×3 macroblocchi
$b)$ 8×5	superblocco = 3×3 macroblocchi

Tabella 3.1: Dimensioni frame base ai vari livelli della gerarchia.

La struttura che è stata presa in considerazione è composta allora solo da due livelli: macroblocchi composti da $n_1 \times m_1$ vettori codice e superblocchi composti da $n_2 \times m_2$ macroblocchi. Sebbene si possa pensare di ottimizzare congiuntamente i parametri n_1,m_1,n_2,m_2, si è preferito operare separatamente per i due livelli, visto che per il primo i valori ottimi n_1^* e m_1^* sono stati già trovati (si veda il paragrafo 2.8.4):

$$\begin{cases} n_1^* &= 3 \\ m_1^* &= 3 \end{cases} \tag{3.1}$$

Per quanto riguarda le dimensioni dei superblocchi, sono state valutate due possibili configurazioni, nelle quali ognuno di essi è formato da 2×3 (configurazione a) oppure da 3×3 macroblocchi (configurazione b). Nella tabella 3.1 sono illustrate le dimensioni di una frame di livello base espressa in termini di pixel, codeword, macroblocchi e superblocchi.

Consideriamo il caso di CR unidirezionale in cui l'overhead per macroblocco è sempre di un bit. Una frame è allora composta da 360 macroblocchi, per cui, se non usiamo la gestione gerarchica della side information, avremo sempre 360 bit di overhead per ogni frame. Se invece usiamo la nuova tecnica, bisogna tenere conto che bisogna inserire un bit di side information per ogni superblocco, quindi altri 60 o 40 bit rispettivamente per il caso a e il caso b. Se il CR non ha mai successo (cioé per $T_Z = 0$) i bit di side information diventano 420 o 400 per frame, ma al crescere di T_Z sarà possibile compattarli, in quanto sempre più spesso ci saranno interi superblocchi in cui il CR ha successo, e per i quali è sufficiente quindi un solo bit

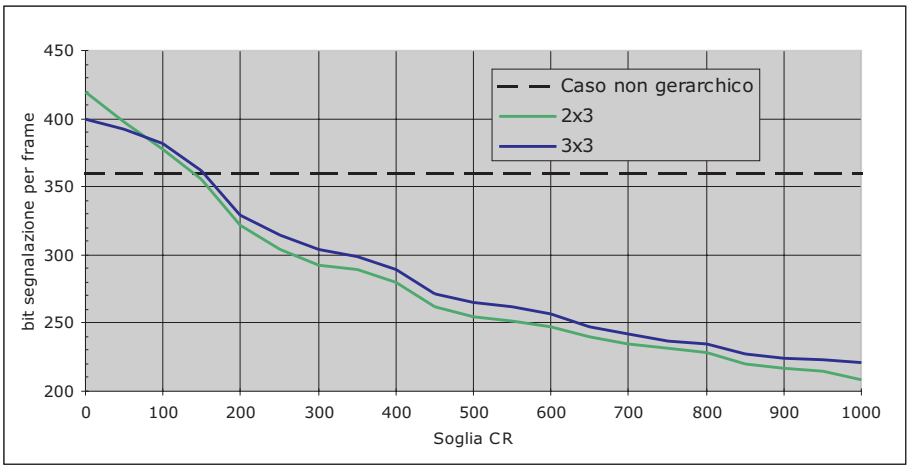

Figura 3.3: Confronto del numero di bit di overhead per frame nel caso gerarchico e nel caso non gerarchico.

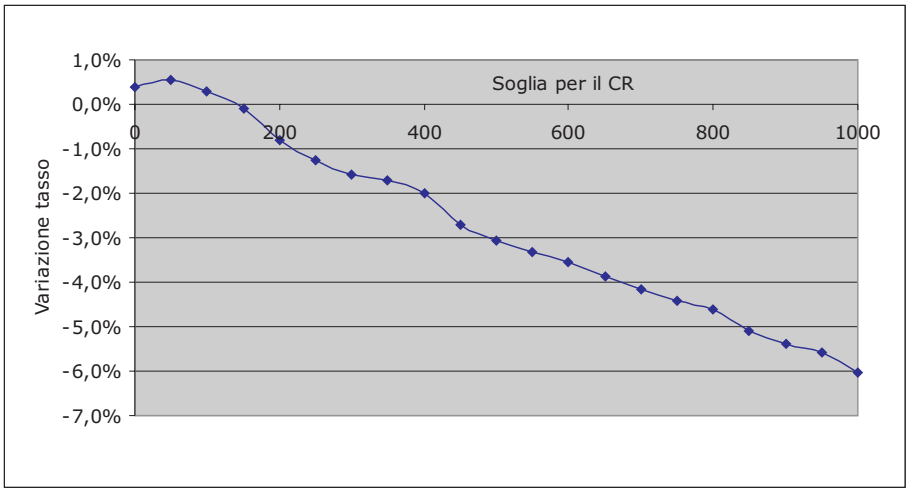

Figura 3.4: Effetto della gestione gerarchica della side information sul tasso al livello base.

di sincronizzazione invece che $9+1$ o $6+1$. Tramite le simulazioni si può poi capire in quali condizioni la tecnica gerarchica diventa conveniente, e qual è il suo effetto complessivo sul tasso di codifica.

Nella figura 3.3 è illustrato il numero di bit di overhead per frame per i tre casi in esame, al variare della soglia del CR nell'intervallo corrispondente alle condizioni di funzionamento di interesse del codec. Si nota allora che per i valori più alti della soglia la gestione gerarchica consente di ridurre consistentemente il numero dei bit di side information per frame, soprattutto nel caso in cui si scelgono superblocchi di dimensione 2×3. Nella figura 3.4 è illustrato poi l'effetto della gestione gerarchica della side information sul tasso complessivo per quanto riguarda il livello base. Come si vede questa riduzione non è molto grande, e ciò è dovuto al fatto che i bit di side information costituiscono comunque una frazione ridotta del flusso totale.

3.3 Riordinamento dei codebook esistenti

Per poter utilizzare con successo la tecnica di CR sugli indirizzi, è necessario che il codebook utilizzato presenti adeguate caratteristiche di *ordinamento*, termine con il quale indichiamo il fatto che i vettori-codice con indirizzi vicini sono (quasi sempre) simili tra loro. Questa proprietà è un "effetto collaterale" della progettazione del codice con la TSVQ, ma risulta di primaria importanza nel nostro codec. È opportuno allora valutare la possibilità di modificare o progettare di nuovo i codebook in modo da enfatizzarne le caratteristiche di ordinamento.

Il livello di ordinamento (detto anche *grado di auto-organizzazione*) di un codebook $\mathcal{C} = \{\mathbf{y}(A) \in \mathbb{R}^k\}_{A=1,\ldots,N}$ può essere descritto, come suggerito in [14], introducendo il funzionale:

$$D(n) \triangleq \frac{1}{N-n} \sum_{A=1}^{N-n} \frac{1}{k} \|\mathbf{y}(A) - \mathbf{y}(A+n)\|^2 \tag{3.2}$$

che rappresenta l'errore quadratico medio tra due vettori codice il cui indirizzo differisce di n, al variare di n. Si può anche introdurre

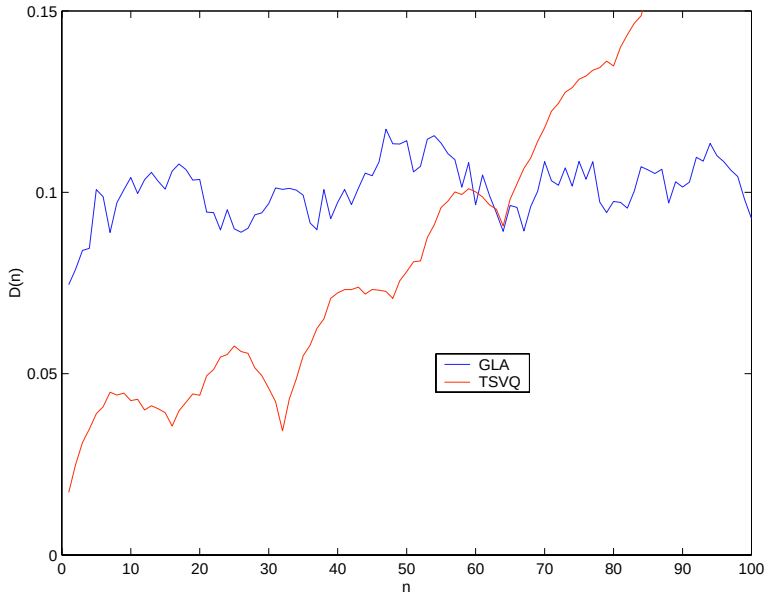

Figura 3.5: Grado di auto-organizzazione di un codebook TSVQ confrontato con quello di un codebook generato con il GLA

una misura più sintetica, data dalla media pesata di $D(n)$

$$D_W \triangleq \sum_{n=1}^{N-1} W(n)D(n) \tag{3.3}$$

nella quale si possono usare dei pesi che decrescono esponenzialmente: $W(n) = 2^{-n}$. Questa misura tiene conto del fatto che i valori più significativi di $D(n)$ sono quelli relativi alle codeword più vicine. Un altro parametro d'interesse è invece l'errore *massimo* tra due vettori codice il cui indirizzo differisce di n:

$$M(n) \triangleq \max_A \frac{1}{k} |||\mathbf{y}(A) - \mathbf{y}(A+n)|||^2 \tag{3.4}$$

Per valutare il livello di ordinamento del codebook tree-structured di cui disponiamo, abbiamo confrontato l'andamento di $D(n)$ per tale codebook con quello relativo ad un codebook ottenuto usando il GLA; i risultati sono illustrati nella figura 3.5.

Dai grafici si può osservare come il codebook strutturato ad albero presenti un certo grado di organizzazione: infatti $D(n)$ è ini-

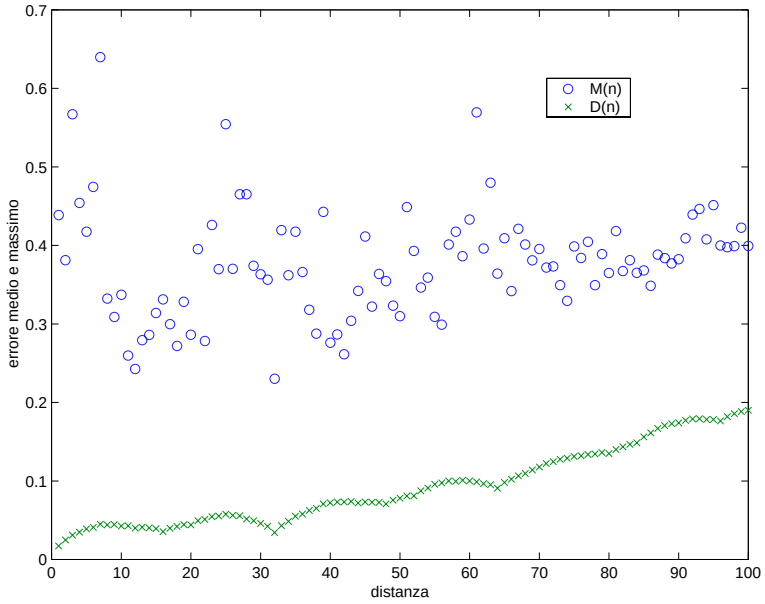

Figura 3.6: Errore medio e massimo tra due codeword per il codebook TSVQ al livello base

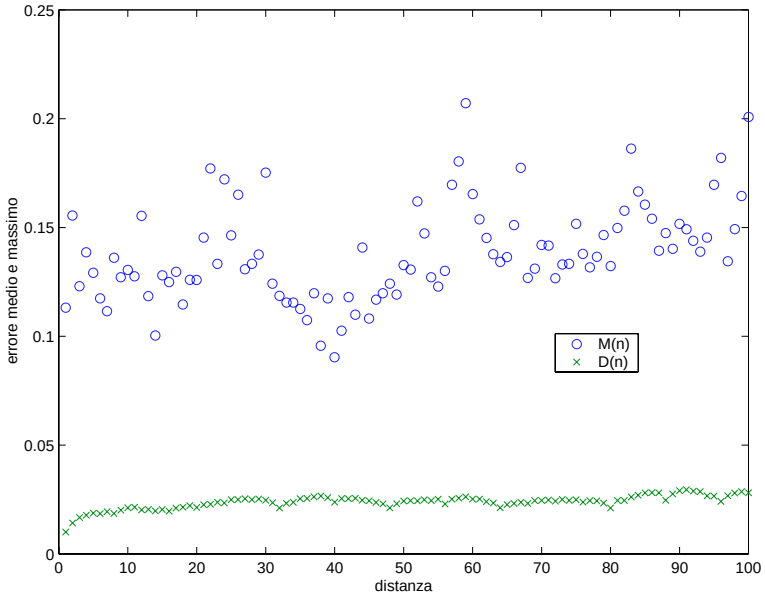

Figura 3.7: Errore medio e massimo tra due codeword per il codebook TSVQ al livello enhancement

zialmente molto piccola ed ha un andamento crescente, mentre per il codebook ottenuto applicando il GLA $D(n)$ ha un andamento praticamente casuale.

Anche se l'andamento di $D(n)$ sembra accettabile per le nostre esigenze, è opportuno valutare anche $M(n)$, che permette di stabilire nel caso peggiore quanto sono diverse due codeword con indirizzi vicini. Abbiamo allora valutato i funzionali (3.2) e (3.4) per i codebook tree-structured sia al livello base che al livello enhancement. I risultati sono riportati nelle figure 3.6 e 3.7.

Si può notare che, mentre il funzionale $D(n)$ presenta un andamento abbastanza "dolce", $M(n)$ é molto più irregolare, e, in particolare, ha dei valori abbastanza grandi anche per n relativamente piccolo. In altri termini, nella maggior parte dei casi, due parole codice con indirizzo vicino sono molto simili, ma nel caso peggiore possono essere molto diverse. Si é dunque ritenuto opportuno analizzare la possibilità di migliorare le caratteristiche di ordinamento del codebook a nostra disposizione, e a questo fine, sono state individuate due possibili strade: generare ex-novo dei codebook con il preciso obiettivo di enfatizzare la proprietà di ordinamento, oppure cercare di migliorare quelli già disponibili usando degli algoritmi di riordinamento che ne conservino la struttura ad albero. Inizialmente si è deciso di sperimentare quest'ultima soluzione, usando la tecnica descritta in [15]. In tale lavoro si affronta il problema della trasmissione di immagini compresse con la VQ su di un canale rumoroso: per migliorare la qualità dell'immagine ricevuta, si utilizza un codebook ordinato che garantisce una elevata correlazione tra i bit più significativi delle parole codice che codificano blocchi di pixel adiacenti. Tale correlazione viene sfruttata in ricezione per implementare un algoritmo di correzione degli errori.

L'ordinamento del codebook è ottenuto in due passi. Il primo consiste nel generare un codebook già relativamente ordinato con un algoritmo detto Principal Component Partitioning (PCP), che fornisce un codebook strutturato ad albero, analogamente al nostro caso. Il secondo passo consiste nel riordinare tale codebook utilizzando opportuni algoritmi che preservino la struttura ad albero del codebook. Il concetto su cui si basano tali algoritmi è quello

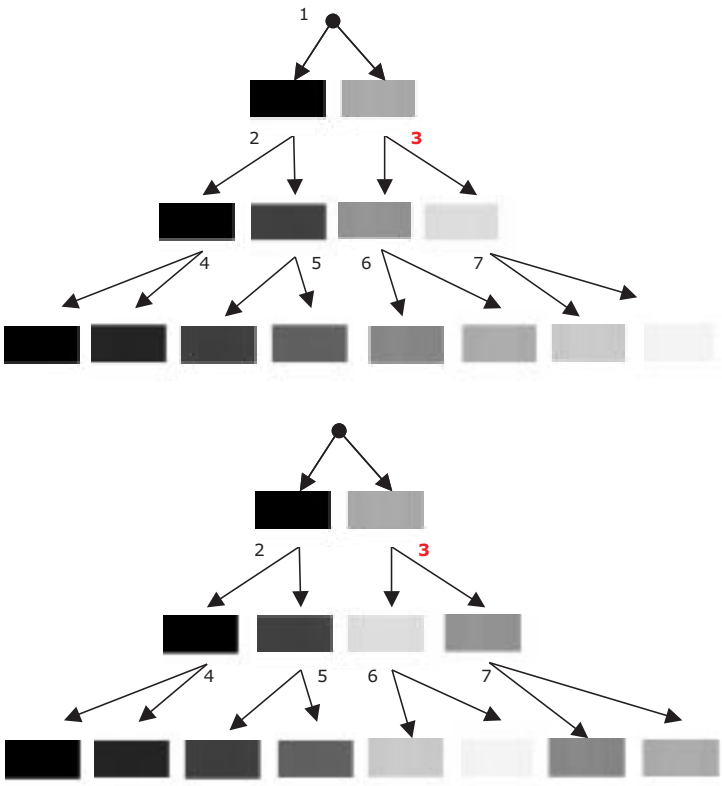

Figura 3.8: Effetto del local switch su di un codebook con struttura ad albero.

di *local switch* (LS). Un'operazione di LS consiste nello scambiare i due sotto-alberi relativi ad un nodo, in modo che le codeword corrispondenti alle loro foglie cambino il loro indirizzo. In figura 3.8 è mostrato l'effetto di un LS effettuato al nodo 3 di un codebook tree-structured a tre livelli. Come si vede, un LS può dare luogo tanto ad un miglioramento quanto ad un peggioramento delle caratteristiche di ordinamento del codebook, ma non modifica le relazioni gerarchiche della struttura ad albero: in altre parole l'insieme dei discendenti di qualsiasi nodo rimane immutato dopo un LS. Per valutare quantitativamente l'effetto di un LS abbiamo usato come misura di distorsione la D_W definita nella (3.3), perché nel nostro caso, a causa del CR sugli indirizzi, è possibile che una parola codice con indirizzo A venga sostituita con un'altra con indirizzo $A + n$.

Invece in [15] si utilizza una misura di distorsione che tiene conto del fatto che un errore sul k-esimo bit b_k (a causa della trasmissione sul canale rumoroso) trasforma un indirizzo A nell'indirizzo $A + (-1)^{b_k} 2^k$. Tuttavia, a parte l'uso di due diverse misure di distorsione, non ci sono altre significative differenze tra i due problemi di riordinamento.

Assegnato un codebook con struttura ad albero ed una misura di distorsione D_W, è evidente che esiste (almeno) una sequenza di LS ottima, cioè tale da portare il codebook al valore minimo di D_W. Tuttavia individuare tale sequenza è un problema computazionalmente molto complesso, per cui si ricorre a tecniche subottime, ma con un peso computazionale accettabile, che permettono comunque di individuare delle configurazioni del codebook localmente ottime. Le tecniche proposte in [15] sono le seguenti.

Simple greedy heuristic (SGH). Si effettua una scansione dell'albero, visitando un nodo alla volta, e per ognuno di essi si effettua un LS. Si valuta la variazione di D_W risultante e si riporta l'albero nella situazione iniziale con un altro LS. Terminata la scansione dell'albero, si effettua il LS che ha causato la massima diminuzione di D_W, e si ricomincia daccapo. L'algoritmo termina quando ogni singolo LS applicato ad un nodo dell'albero fa aumentare la distorsione, cioè al raggiungimento di un ottimo locale.

SGH with simulated annealing. L'algoritmo SGH viene ripetuto k volte, con l'obiettivo di svincolarsi dagli ottimi locali. Ogni volta l'albero di partenza è diverso, e viene ottenuto effettuando m local switch su altrettanti nodi dell'albero iniziale scelti in modo casuale. Il valore suggerito di k per un codebook di 256 elementi è 100. Invece, visto che il codebook di partenza è già abbastanza ordinato, si suggerisce di scegliere m pari al 5% dei nodi dell'albero.

Top-down greedy heuristic (TDGH). L'algoritmo è simile al SGH, l'unica differenza è la tecnica di scansione dell'albero: visto che le maggiori variazioni di D_W si hanno in corrispondenza dei nodi prossimi alla radice (perché i LS spostano molte co-

deword, si veda la figura 3.8), la scansione viene effettuata livello per livello.

TDGH with simulated annealing. L'algoritmo TDGH viene ripetuto k volte, partendo da alberi ottenuti applicando LS casuali a m nodi.

In [15] gli autori sottolineano come i vari algoritmi diano risultati praticamente equivalenti in termini di distorsione, mentre il tempo di esecuzione delle tecniche con il *simulated annealing* è nettamente maggiore degli altri. Per questo motivo abbiamo scelto di implementare solo l'algoritmo Simple Greedy Heuristic. I risultati sono presentati nelle figure da 3.9 a 3.16.

Gli esperimenti hanno messo in luce che l'algoritmo è effettivamente in grado di migliorare l'ordinamento del codebook. In particolare si può notare nelle figure 3.9 e 3.10 come alcune parole codice abbastanza diverse, con indirizzi vicini perché all'estremità dei rispettivi sotto-alberi, siano state spostate opportunamente. Si tenga presente che in queste figure le codeword sono ordinate per colonne, per cui quelle evidenziate con lo stesso colore hanno indirizzi consecutivi.

Il riordinamento si riflette sull'andamento dei funzionali $D(n)$ e $M(n)$, come illustrato nelle figure da 3.11 a 3.14. Si nota che ora $D(n)$ ha un andamento più regolare (come auspicabile) soprattutto al livello base, dove è praticamente monotòno. Inoltre assume valori più bassi in corrispondenza di $n < 10$, che è poi la regione di interesse, perché due codeword il cui indirizzo dista più di 10 cominciano ad essere molto diverse e quindi è difficile che l'una possa essere usata al posto dell'altra nel CR. In altre parole non è tanto importante che $D(n)$ sia aumentato per valori molto elevati di n, quanto che per valori più bassi sia diminuito.

Anche osservando l'andamento di $M(n)$ si può verificare che l'ordinamento è migliorato: infatti per i valori di interesse di n, l'errore massimo tra due codeword $\mathbf{y}(A)$ e $\mathbf{y}(A+n)$ si è ridotto, soprattutto al livello base.

Il riordinamento del codebook, tuttavia, non incide significativamente sulle prestazioni del codec, come chiarito dalle figure 3.15 e

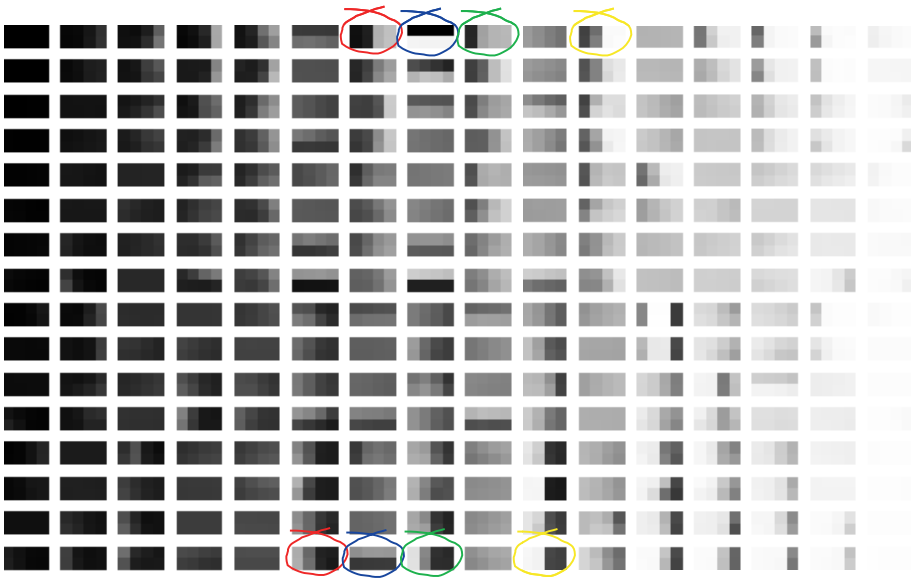

Figura 3.9: Codebook di livello base con struttura ad albero: sono evidenziate con lo stesso colore coppie di codeword consecutive ma significativamente diverse tra loro.

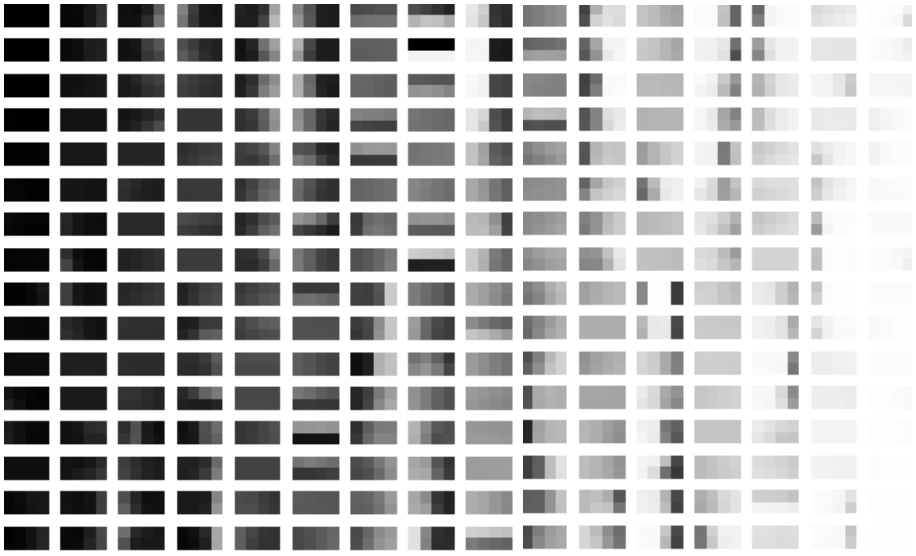

Figura 3.10: Il codebook della figura precedente dopo il riordino.

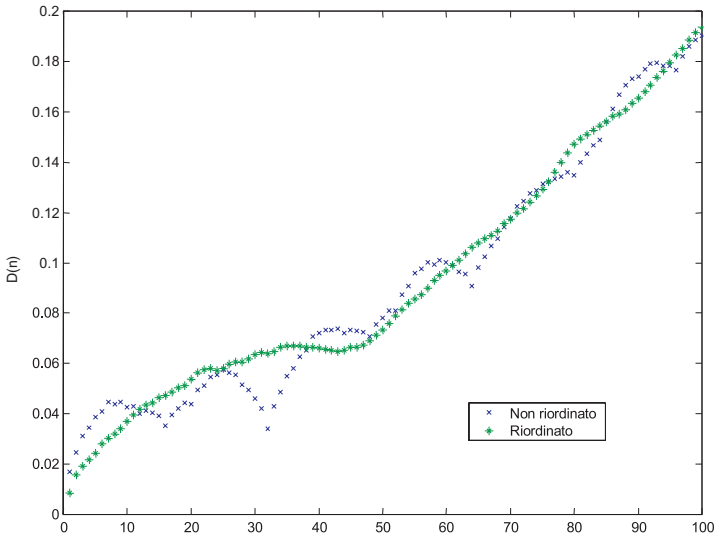

Figura 3.11: Confronto dell'andamento di $D(n)$ prima e dopo il riordinamento per il codebook di livello base.

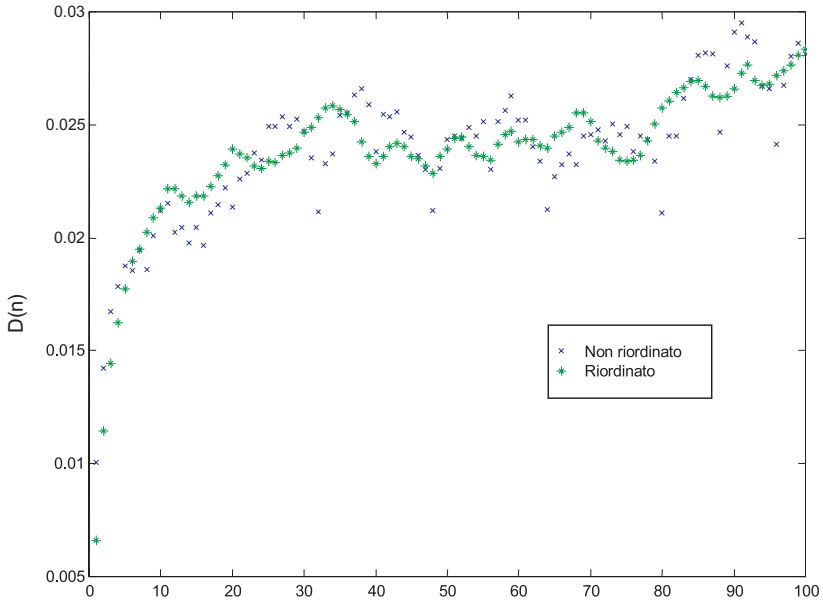

Figura 3.12: Confronto dell'andamento di $D(n)$ prima e dopo il riordinamento per il codebook di livello enhancement.

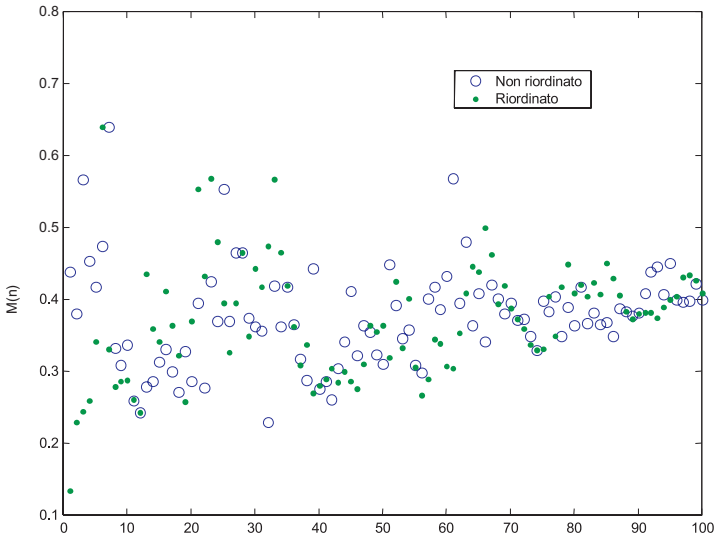

Figura 3.13: Confronto dell'andamento di $M(n)$ prima e dopo il riordinamento per il codebook di livello base.

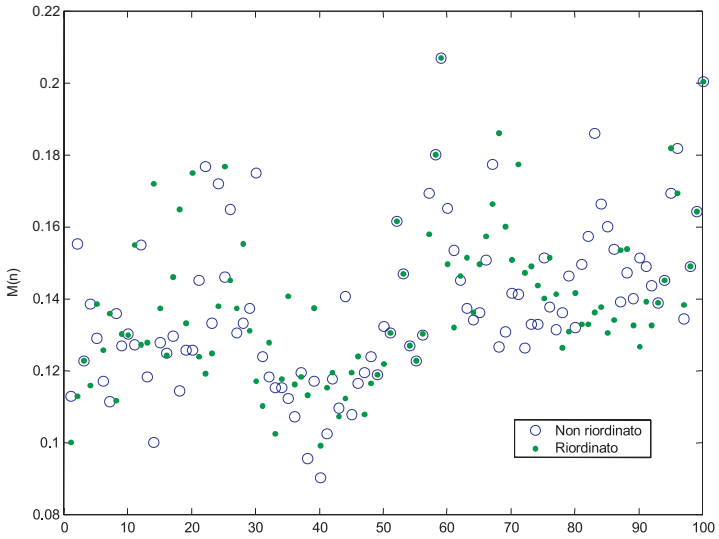

Figura 3.14: Confronto dell'andamento di $M(n)$ prima e dopo il riordinamento per il codebook di livello enhancement.

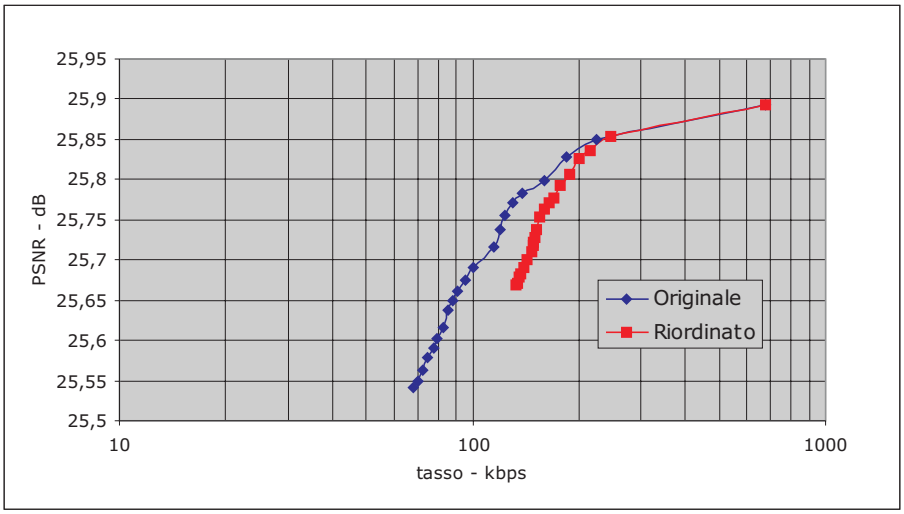

Figura 3.15: Prestazioni del codec al livello base valutate usando il codebook originario e quello riordinato.

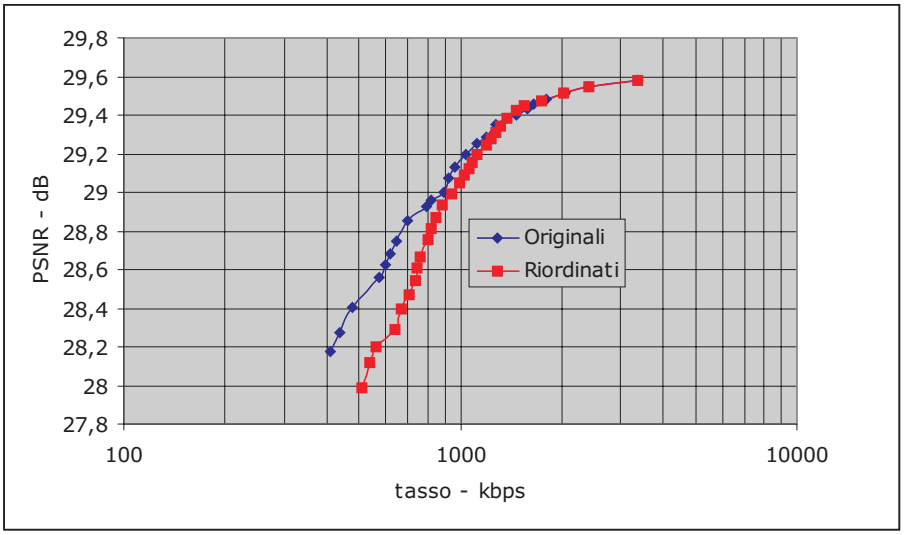

Figura 3.16: Prestazioni del codec al livello enhancement valutate usando i codebook originari e quelli riordinati.

3.16, dove invece appare che l'uso di codebook riordinati comporta quasi sempre un peggioramento delle prestazioni. Questo risultato, un po' inatteso viste le buone caratteristiche di ordinamento dell'algoritmo SGH, è stato attribuito alla portata ridotta dei nostri test, e al fatto che la tecnica di CR prevede la sostituzione di blocchi interi di indirizzi, per i quali la distanza è valutata con una metrica quadratica, che, tutto sommato, è stata scelta in modo arbitrario, e quindi non è detto che sia la più corretta per sfruttare le proprietà di ordinamento.

3.4 Tecniche di filtraggio

L'algoritmo di codifica prevede prima di ogni operazione di sottocampionamento un filtraggio anti-aliasing dell'immagine. Questa è un'operazione costosa in termini computazionali, ma che permette di migliorare notevolmente le prestazioni del codec. Nel prototipo iniziale si usa una tecnica di filtraggio molto semplice: per ogni blocco di 2×2 pixel della frame d'ingresso si calcola la media e la si usa come campione della frame a risoluzione inferiore. Questo coincide con l'uso di un filtro bidimensionale con risposta impulsiva:

$$h(n,m) = \begin{cases} 1/4 & \text{per } n,m \in \{0,1\} \\ 0 & \text{altrimenti} \end{cases} \tag{3.5}$$

Questa operazione di filtraggio bidimensionale equivale a filtrare due volte (prima sulle righe e poi sulle colonne) l'immagine con un filtro monodimensionale di risposta impulsiva:

$$h(n) = \begin{cases} 1/2 & \text{per } n \in \{0,1\} \\ 0 & \text{altrimenti} \end{cases} \tag{3.6}$$

detto anche filtro di Haar. Questa tecnica di filtraggio è molto semplice, ma non di meno rappresenta una dell'operazioni più costose (dal punto di vista computazionale) tra quelle che il codificatore deve realizzare. Comunque gli esperimenti hanno dimostrato un netto miglioramento delle prestazioni quando si usa questo semplice filtraggio rispetto al caso in cui non se ne effettua nessuno, per

Tipo di filtro	Lunghezza	Tempo di codifica [ms]	PSNR [dB]
nessuno	-	35	28.231
Haar	2	44	29.582
Daubechies	4	60	30.014
Daubechies	6	65	30.176
Daubechies	8	75	30.180

Tabella 3.2: Effetto dei diversi tipi di filtraggio sui tempi di codifica e sulle prestazioni al livello enhancement, con $T_Z = 0$ e $T_W = 0$.

cui è stato deciso di includere questa operazione nell'algoritmo di codifica.

Si è voluto allora esplorare ulteriormente il trade-off tra prestazioni e complessità conseguente all'uso di filtri più lunghi, e l'attenzione è stata spostata sui filtri "wavelet" in vista della possibilità di implementazioni veloci con la tecnica del "lifting scheme" [16]; in particolare sono stati usati i filtri proposti da Daubechies [17].

Nella tabella 3.2 sono riassunti i principali risultati dalle simulazioni, mentre nella figura 3.17 si confrontano le prestazioni (al livello enhancement) che si possono ottenere con l'uso dei filtri Haar 2 e Daubechies 4. Come si vede, l'uso di filtri più lunghi del semplice Haar 2 comporta un netto aumento dei tempi di codifica, ed un miglioramento delle prestazioni non sempre adeguato. Tenendo conto però della possibilità di ridurre i tempi di codifica con l'uso di tecniche veloci (lifting scheme o tecniche tabellari) e della crescente potenza di calcolo dei microprocessori di cui sono dotati i PC, sembra plausibile per il futuro l'uso del filtro Daubechies 4 per elaborazioni in tempo reale.

3.5 Compattazione degli indirizzi

Come già sottolineato in precedenza, l'operazione di quantizzazione vettoriale può essere considerata come la successione di un'operazione di codifica e una di decodifica. In particolare il codificatore associa ad ogni vettore in ingresso l'indice del corrispondente vettore del codebook. Nel caso di quantizzazione di immagini i simboli emessi dal codificatore saranno dipendenti in seguito alla correla-

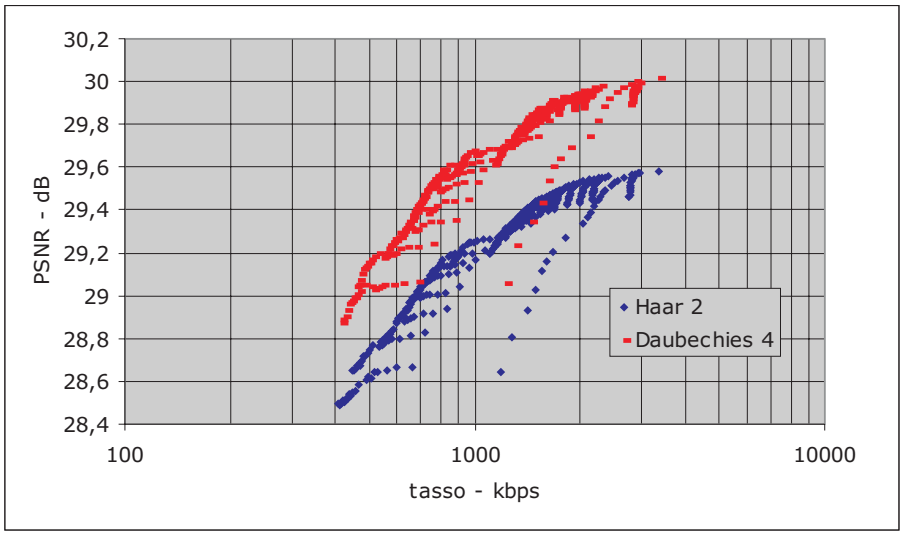

Figura 3.17: Confronto delle prestazioni al livello enhancement con diversi tipi di filtraggio.

zione esistente tra i vettori d'ingresso. Questa dipendenza può essere sfruttata per ottenere ancora una compressione, tuttavia, a causa della non linearità dell'operazione di codifica, gli indirizzi non risultano correlati e l'unico modo per compattare i dati rimane, in generale, quello di raggruppare tali indirizzi in blocchi ed effettuare una codifica entropica su di essi. Questa tecnica, nota col nome di Address Vector Quantization (AVQ), presenta notevoli problemi di complessità computazionale, anche per vettori piccoli e codebook di cardinalità ridotta, in quanto la codifica entropica deve essere fatto su dei simboli il cui alfabeto ha una cardinalità molto grande[3]. Se però si riuscisse a fare in modo che i simboli di uscita del codificatore, invece di essere statisticamente dipendenti in modo non lineare, fossero semplicemente *correlati*, si potrebbero usare a valle delle tecniche di codifica più semplici, come ad esempio la codifica predittiva. Per ottenere questo risultato è sufficiente usare un codebook ordinato: se ad indirizzi vicini corrispondono vettori codice

[3]Il "muro computazionale" della VQ, che era stato evitato usando vettori di dimensioni ridotte, si ripresenta quando si cerca di raggruppare gli indici che li codificano per sfruttarne la dipendenza.

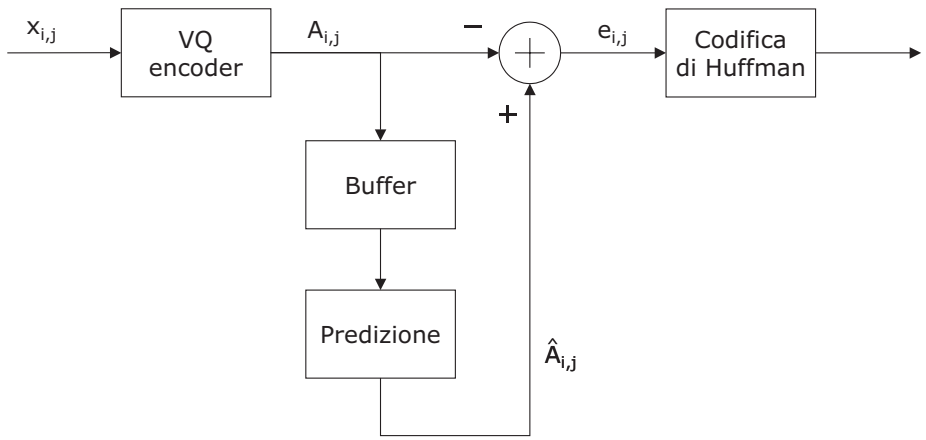

Figura 3.18: Schema di codifica dell'APVQ.

simili, gli indici risultanti dalla codifica di zone grosso modo uniformi saranno dei numeri vicini tra loro o, in altre parole, il codificatore emetterà una sequenza di valori fortemente correlati.

3.5.1 Codifica predittiva per indirizzi (APVQ)

L'utilizzo di codebook ordinati consente di conservare la correlazione all'uscita del codificatore, e quindi di usare tecniche di codifica di tipo predittivo. Questo tipo di codifica, indicato come Address Predictive Vector Quantization (APVQ) è stato introdotto in [14], e lo schema di base è illustrato nella figura 3.18.

Sia $\mathbf{x}_{i,j}$ il vettore codice d'ingresso (i e j rappresentano l'indice di riga e di colonna); indichiamo con $A_{i,j}$ l'indirizzo della codeword che codifica $\mathbf{x}_{i,j}$. L'indirizzo $A_{i,j}$ viene memorizzato (servirà per le successive predizioni), e confrontato con il valore predetto $\hat{A}_{i,j}$, che viene determinato sulla base del buffer, cioè dei blocchi già quantizzati. L'algoritmo di predizione è abbastanza semplice: il valore previsto per $A_{i,j}$ è o l'elemento precedente sulla stessa riga, o quello sulla stessa colonna. La scelta è fatta stimando qual'è la direzione migliore per la predizione, e tale stima viene fatta valutando gli

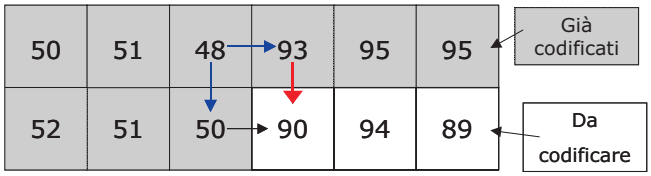

Figura 3.19: Esempio di funzionamento dell'algoritmo adattativo di predizione.

errori di riga e di colonna:

$$e_C = |A_{i-1,j-1} - A_{i,j-1}| \tag{3.7}$$

$$e_R = |A_{i-1,j-1} - A_{i-1,j}| \tag{3.8}$$

Se $e_C < e_R$, la direzione preferenziale è quella verticale, e quindi $A_{i,j}$ è predetto con $A_{i-1,j}$; altrimenti con $A_{i,j-1}$. Si noti che questo algoritmo di predizione impiega elementi che sono noti anche in ricezione, quindi non richiede che venga trasmessa della side information, e inoltre permette di codificare bene sia i bordi orizzontali che quelli verticali. Il funzionamento dell'algoritmo è esemplificato nella figura 3.19, in cui è rappresentata una parte di un'immagine (codificata con la VQ e con un codebook ordinato) dove è presente un bordo verticale. La predizione della prossima codeword (90) viene fatta valutando l'errore orizzontale, pari a 45 (=93-48) e quello verticale, pari a 2 (=50-48). Si decide allora che la direzione migliore per la predizione è quella verticale, e quindi la predizione è 93 e l'errore 3.

Infine l'errore di predizione viene codificato con un opportuno codice di Huffman, generato in precedenza valutando le statistiche dell'errore di predizione.

3.5.2 Prestazioni dell'APVQ

La tecnica proposta produce un miglioramento delle prestazioni se effettivamente l'errore di predizione può essere codificato mediamente con meno degli otto bit necessari per trasmettere normalmente un indirizzo. Le prime simulazioni che abbiamo effettuato riguardano proprio la stima dell'entropia del primo ordine dell'erro-

re di predizione: esse hanno in effetti evidenziato che al livello base l'errore di predizione ha un entropia che si mantiene sui sei bit per i valori d'interesse della soglia (si veda la figura 3.20), mentre al livello enhancement si hanno valori più alti, anche prossimi agli otto bit (figura 3.21). Per questo si è deciso di usare la tecnica di APVQ solo al livello base.

La figura 3.22 mostra l'effetto dell'APVQ sul tasso di trasmissione: si nota una riduzione dell'ordine del 20%, che è sicuramente interessante per gli utenti del livello base, che sono presumibilmente quelli per i quali la larghezza di banda è una risorsa scarsa.

Nella figura 3.23 la riduzione effettivamente conseguita del tasso è confrontata con quella massima teorica, calcolata come la riduzione del tasso che si avrebbe se tutti gli indirizzi potessero essere codificati con un numero medio di bit pari all'entropia dell'errore di predizione. Si nota che la riduzione che si riesce ad ottenere grazie alla codifica di Huffman è abbastanza vicina a quella massima teorica.

3.6 Progetto di un codebook ordinato

Come abbiamo visto, la proprietà di ordinamento del codebook è di centrale importanza per il nostro codec, visto che rende possibili le tecniche di CR sugli indirizzi e di APVQ. Si è allora voluto investigare più approfonditamente sulla possibilità di ottenere dei codebook ordinati. Finora abbiamo considerato solo codebook con struttura ad albero, per i quali la proprietà d'ordinamento è una conseguenza dell'algoritmo usato per la generazione del codebook. Adesso invece vogliamo considerare la possibiltà di progettare dei codebook con il principale obiettivo dell'ordinamento, magari rinunciando alla struttura ad albero. Questo significa che i nuovi codebook non supportano la scalabilità in qualità; tuttavia questa perdita è tollerabile, visto che le simulazioni hanno evidenziato che il trade-off tra tasso e prestazioni garantito da questo tipo di scalabilità non è molto favorevole. In altre parole, la riduzione del tasso che si ottiene rinunciando a usare 8 bit per ogni parola codice non compensa adeguatamente il degrado della qualità della sequenza deco-

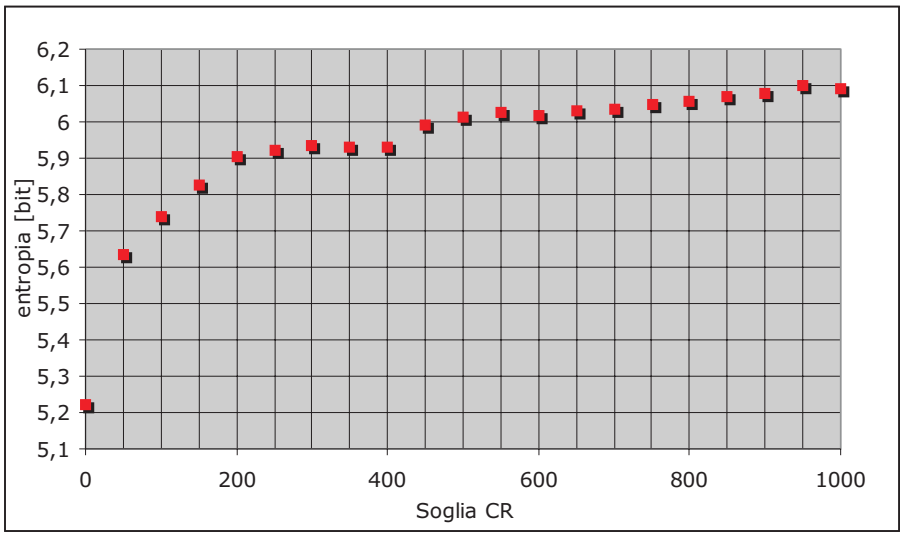

Figura 3.20: Entropia dell'errore di predizione, livello base.

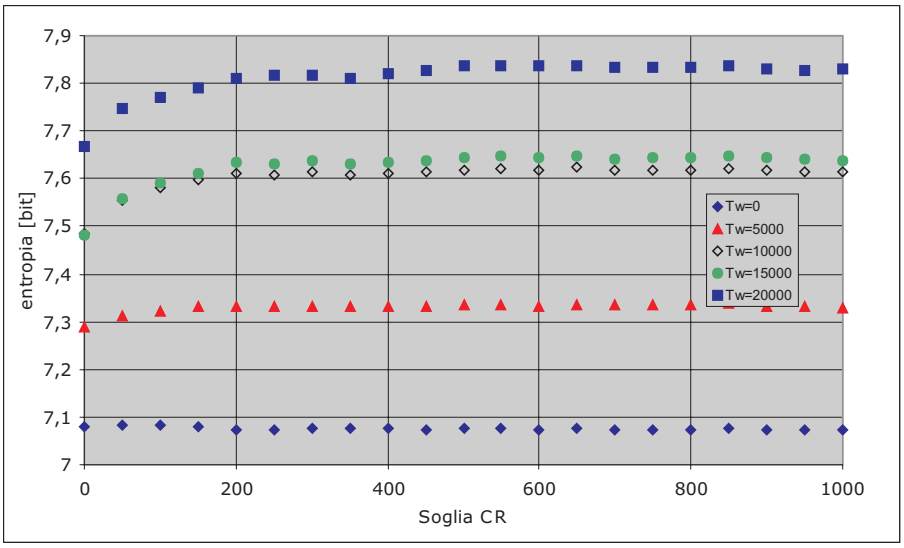

Figura 3.21: Entropia dell'errore di predizione, livello enhancement.

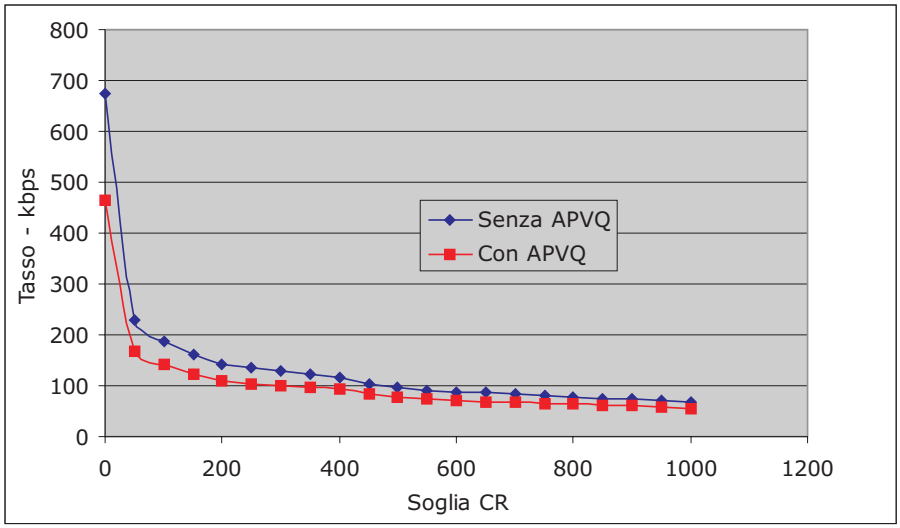

Figura 3.22: Tasso di trasmissione al livello base.

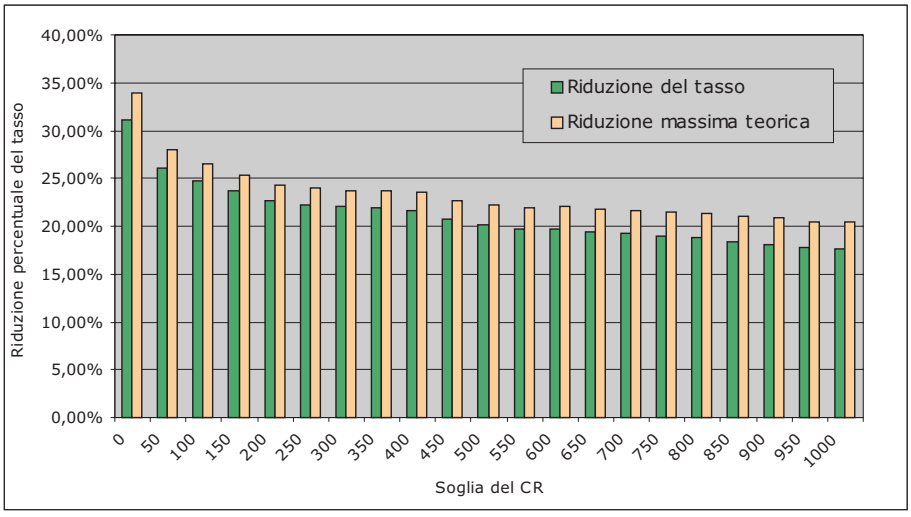

Figura 3.23: Riduzione del tasso dovuta alla APVQ.

dificata, mentre la scalabilità può essere più efficacemente sfruttata riducendo la risoluzione o il frame-rate.

Un modo per ottenere un codebook che presenti le desiderate caratteristiche di ordinamento è quello di usare le cosiddette mappe auto-organizzanti (Self Organizing Feature Map - SOFM), originariamente proposte da Kohonen [18] nell'ambito delle reti neurali, ed introdotte in [14] per l'APVQ. Il codebook che è stato creato con questa tecnica non ha una struttura ad albero, per cui non è utilizzabile per generare un flusso di bit *embedded*.

L'algoritmo di Kohonen consente di generare un codebook con le desiderate caratteristiche di ordinamento a partire da uno iniziale scelto arbitrariamente, e da un training set, indicato con $X = \{\mathbf{x}(s),\ s = 1, \ldots, S\}$. L'algoritmo è iterativo: al generico passo k, si scandisce l'intero training set, e per ogni $\mathbf{x}(s) \in X$, si individua il vettore del codebook che meglio lo rappresenta, sia questo $\mathbf{y}(A_s)$. Tale vettore viene spostato in direzione di $\mathbf{x}(s)$, e poiché ciò avviene per ogni vettore codice di cui $\mathbf{y}(A_s)$ è la codifica, alla fine di un'iterazione, l'effetto di questi spostamenti è quello di portare tale vettore codice verso il centroide della relativa cella in cui è ripartito il training set. A questa traslazione se ne sovrappone una verso i centroidi delle partizioni associate ai vettori codice con indirizzi vicini: tali spostamenti sono però ridotti da un fattore di scala che dipende dalla distanza tra le parole codice (intesa come differenza tra gli indirizzi) e dal passo di iterazione. Inoltre il numero di vettori codice influenzati da uno stesso vettore di training dipende dal passo k di iterazione. Questo algoritmo di aggiornamento determina allora, per ogni iterazione, degli spostamenti simili per parole codice con indirizzi vicini: così si può ottenere un codebook ordinato.

In sintesi la formula di aggiornamento del codebook alla k-esima scansione dell'elemento $\mathbf{x}(s)$ del training set, è:

$$\begin{aligned} \mathbf{y}(A)^{(k)} &= \mathbf{y}(A)^{(k-1)} + \alpha(k) r(A - A_s)(\mathbf{x}(s) - \mathbf{y}(A)^{(k-1)}) \\ &\quad \forall A : |A - A_s| \leq N(k) \end{aligned} \tag{3.9}$$

La funzione $\alpha(\cdot)$ è il tasso di adattamento al training set; per essa

si sceglie tipicamente una legge esponenziale, del tipo:

$$\alpha(s) = \alpha_0 e^{(-s/S_\alpha)} \tag{3.10}$$

Il motivo è che tale funzione deve essere piuttosto grande all'inizio, per garantire che i vettori codice seguano la distribuzione del training set, ma deve poi anche permettere il raggiungimento di una situazione stazionaria.

La funzione $r(\cdot)$ è una funzione di peso, ad esempio il coseno rialzato.

La funzione $N(k)$ rappresenta invece l'ampiezza dell'insieme di codeword influenzate da ogni vettore di training $\mathbf{x}(s)$ alla k-esima iterazione. Tale insieme risulta allora formato da tutti i vettori codice il cui indirizzo differisce da A_s per meno di $N(k)$, dove ricordiamo che:

$$A_s = \arg\min_A \|\mathbf{y}(A) - \mathbf{x}(s)\|^2 \tag{3.11}$$

Le caratteristiche di ordinamento del codebook dipendono essenzialmente dalla funzione $N(\cdot)$. Essa deve essere abbastanza grande per le prime iterazioni, in modo da garantire che inizialmente si raggiunga abbastanza presto un certo grado di ordinamento, ma deve poi decrescere affinché le codeword siano adeguatamente diversificate. Si assume anche per tale funzione un andamento esponenziale:

$$N(k) = N_0 e^{-s/S_n} + N_\infty \tag{3.12}$$

Il parametro critico è S_n, che determina quanto rapidamente si restringe il vicinato. Se tale parametro è troppo alto, il vicinato si restringe lentamente, dando luogo ad un codebook molto ordinato, ma privo di parole codice adeguatamente differenziate, e quindi un codebook con una distorsione relativamente elevata. Se invece S_n è troppo piccolo, il vicinato si restringe rapidamente, sicché la distorsione è minore, ma il codebook è anche meno ordinato. In sintesi la scelta del valore di S_n permette di scambiare ordinamento per distorsione.

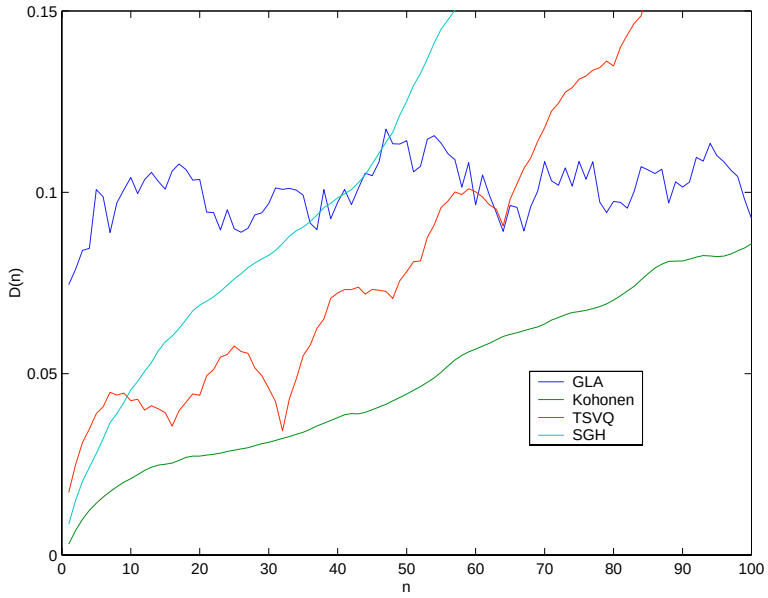

Figura 3.24: Confronto dell'auto-organizzazione dei codebook di livello base

Nella figura 3.24 viene confrontato l'andamento di $D(n)$ per il codebook tree-structured iniziale, il codebook riordinato con l'algoritmo SGH, il codebook generato con l'algoritmo di Kohonen, ed un codebook generato con il GLA, mentre nelle figure 3.25 e 3.26 viene mostrato l'andamento di $D(n)$ e $M(n)$ per i codebook di Kohonen ai livelli base ed enhancement. Da questi grafici appare chiaro che questi codebook hanno le migliori caratteristiche di ordinamento.

Le prestazioni sono illustrate nelle figure 3.27 e 3.28. Si nota che al livello base, l'uso di un codebook di Kohonen permette di ottenere un significativo incremento del PSNR; tuttavia, in corrispondenza dei valori più alti per la soglia T_Z, si nota un netto calo della qualità soggettiva, a cui non fa riscontro un adeguata diminuzione del tasso: in pratica per tassi al di sotto dei 150 kbps la sequenza decodificata appare scadente ad un osservatore umano, mentre, nel caso tree-structured, si ha una qualità soggettiva accettabile intorno a 100 kbps.

A livello enhancement l'uso dei codebook di Kohonen non riesce a migliorare significativamente le prestazioni, anzi per i valori d'interesse del tasso, le prestazioni migliori si ottengono con il co-

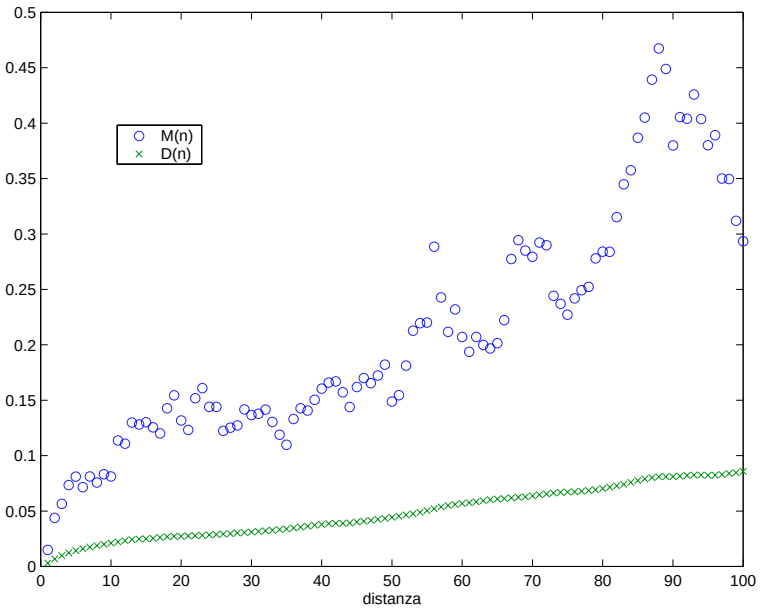

Figura 3.25: Andamento di $D(n)$ e di $M(n)$ per il codebook di livello base ottenuto con l'algoritmo di Kohonen.

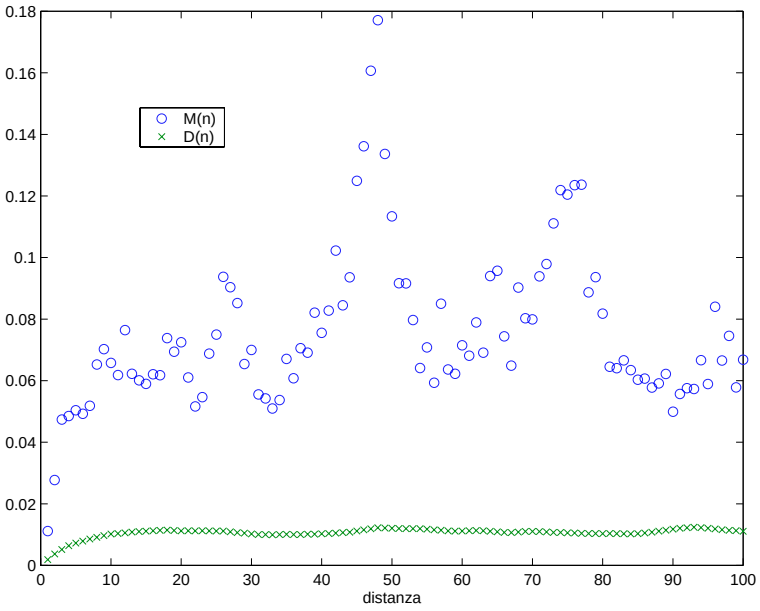

Figura 3.26: Andamento di $D(n)$ e di $M(n)$ per il codebook di livello enhancement ottenuto con l'algoritmo di Kohonen.

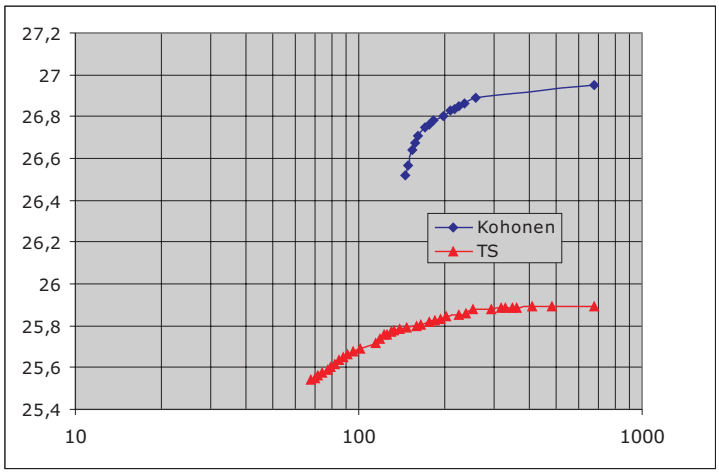

Figura 3.27: Prestazioni del codec al livello base con i codebook ottenuti con l'algoritmo di Kohonen.

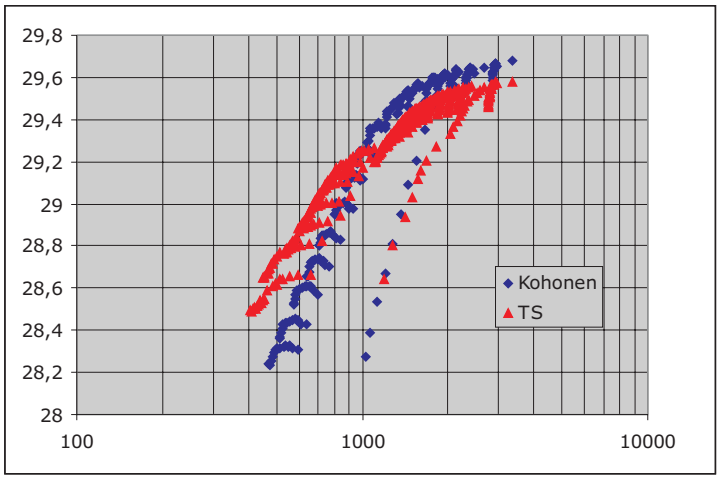

Figura 3.28: Prestazioni del codec al livello enhancement con i codebook ottenuti con l'algoritmo di Kohonen.

debook tree-structured. Questi risultati giungono inattesi, alla luce delle ottime proprietà di ordinamento dimostrate dal codebook di Kohonen: ciò ha fornito lo spunto per riconsiderare nel suo insieme l'algoritmo di codifica.

3.7 Il problema della metrica per il CR

Una delle tecniche che caratterizzano maggiormente il nostro codificatore rispetto ad altre soluzioni proposte per la codifica video a bassa complessità [4] [5] è la tecnica di CR sugli indirizzi. Ricordiamone brevemente il funzionamento: un blocco di 6×12 pixel viene quantizzato, dando luogo a una matrice $\mathbf{A}$ di 3×3 indirizzi da 0 a 255. Dalla frame di riferimento quantizzata si estrae la matrice $\hat{\mathbf{A}}$ sempre 3×3, che si trova nella stessa posizione di $\mathbf{A}$. La distanza tra le due matrici è calcolata come energia della matrice differenza:

$$d(\mathbf{A}, \hat{\mathbf{A}}) = \sum_{\substack{i=0,\ldots,2 \\ j=0,\ldots,2}} (A_{i,j} - \hat{A}_{i,j})^2 \tag{3.13}$$

dove $A_{i,j}$ $[\hat{A}_{i,j}]$ è l'indirizzo di posto i, j nella matrice $\mathbf{A}$ $[\hat{\mathbf{A}}]$. Il test per il CR ha esito positivo (cioè stabilisce che i due blocchi sono sufficientemente simili) se la distanza calcolata con la 3.13 è minore di una soglia opportunamente fissata.

Questa metrica è stata direttamente mutuata dal caso di CR sui pixel, dove i blocchi di $n \times m$ pixel $\mathbf{Z}$ e $\hat{\mathbf{Z}}$ hanno distanza euclidea:

$$d(\mathbf{Z}, \hat{\mathbf{Z}}) = \sum_{\substack{i=0,\ldots,n \\ j=0,\ldots,m}} (Z_{i,j} - \hat{Z}_{i,j})^2 \tag{3.14}$$

In questo caso tale metrica assume un significato chiaro: è l'energia dell'errore fatto sulle luminanze dei pixel.

Invece il significato della metrica definita nella (3.13) non è altrettanto chiaro, in quanto gli elementi delle matrici $\mathbf{A}$ e $\hat{\mathbf{A}}$ sono degli indirizzi. Non si può dire che questa distanza sia necessa-

riamente più adatta di un'altra a giudicare la somiglianza tra due macroblocchi; possiamo allora definire un'altra metrica:

$$d^*(\mathbf{A}, \hat{\mathbf{A}}) = \sum_{\substack{i=0,\ldots,2 \\ j=0,\ldots,2}} |A_{i,j} - \hat{A}_{i,j}| \tag{3.15}$$

secondo la quale la distanza tra due matrici di indirizzi è data dalla somma dei moduli delle differenze di indirizzo.

Nelle figure da 3.29 a 3.32 sono confrontate le prestazioni ottenute con l'uso di questa nuova metrica con quelle che si ottengono usando la vecchia: la prima è stata indicata con "abs" la seconda con "norma". I grafici chiariscono che la nuova metrica garantisce le prestazioni migliori praticamente in tutte le condizioni di funzionamento. A questo si deve aggiungere che la nuova metrica richiede meno operazioni per essere calcolata, per cui i tempi medi di codifica per frame si riducono di circa il 30%, passando dai 44 ms a 30 ms. In entrambi i casi si è adottata la strategia del test sul CR indipendente al livello enhancement.

Tutto questo è già sufficiente a far propendere per questa nuova metrica; ma c'è un altro aspetto molto interessante, evidenziato nelle figure 3.33 e 3.34: con questa nuova metrica si riescono a sfruttare meglio le caratteristiche di ordinamento dei codebook, infatti adesso i codebook generati con l'algoritmo di Kohonen presentano in quasi tutte le condizioni delle prestazioni migliori di quelle dei codebook TS. In altre parole, sembra che la nuova metrica permetta di individuare con più precisione i macroblocchi simili, e quindi di far trasparire, anche al livello di prestazioni. la superiorità del codebook di Kohonen.

Bisogna però dire che in alcune condizioni di funzionamento, le prestazioni dei codebook tree-structured si mantengono ancora superiori, in contrasto con le aspettative. Questo però potrebbe anche essere legato ai dati di prova che abbiamo a disposizione: probabilmente, ampliando il campo delle simulazioni, si possono ottenere risultati più consistenti con le previsioni teoriche.

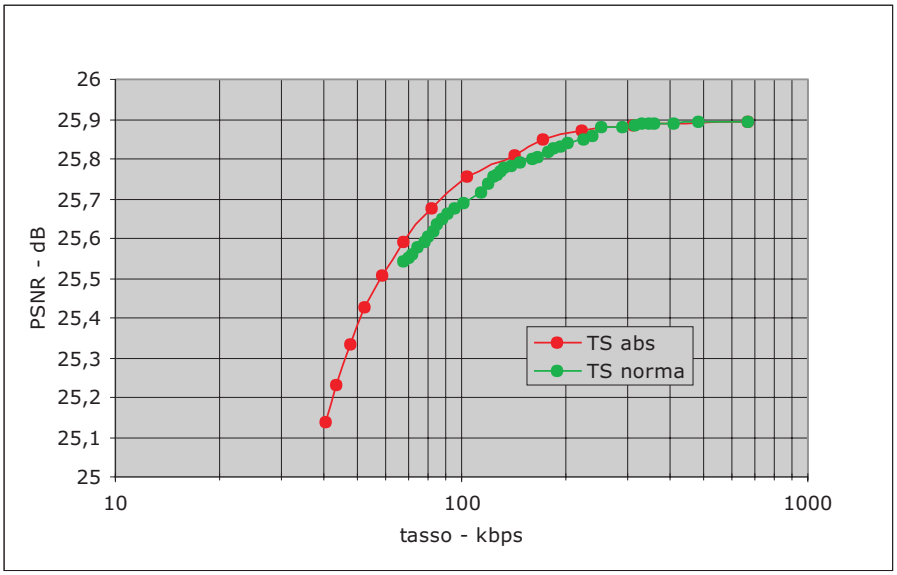

Figura 3.29: Metriche a confronto con il codebook tree-structured al livello base.

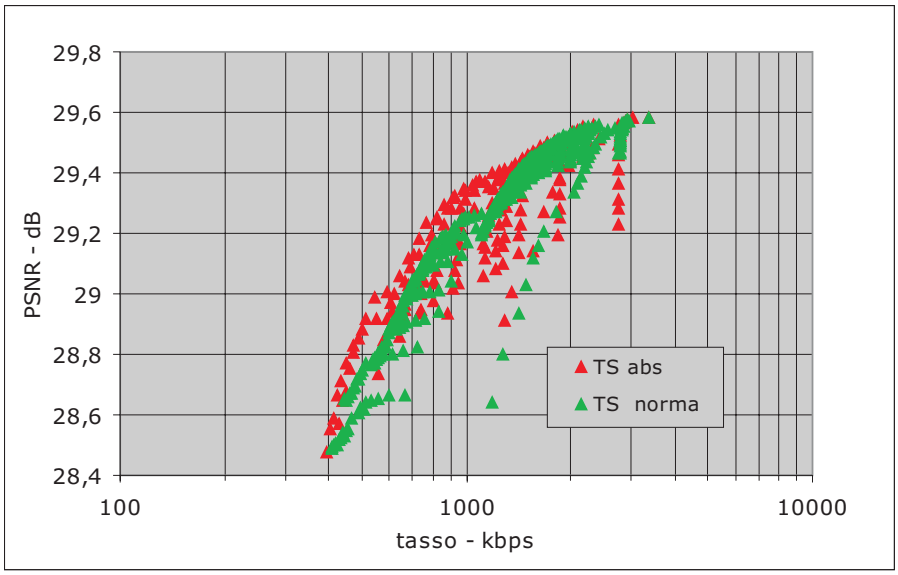

Figura 3.30: Metriche a confronto con il codebook tree-structured al livello enhancement.

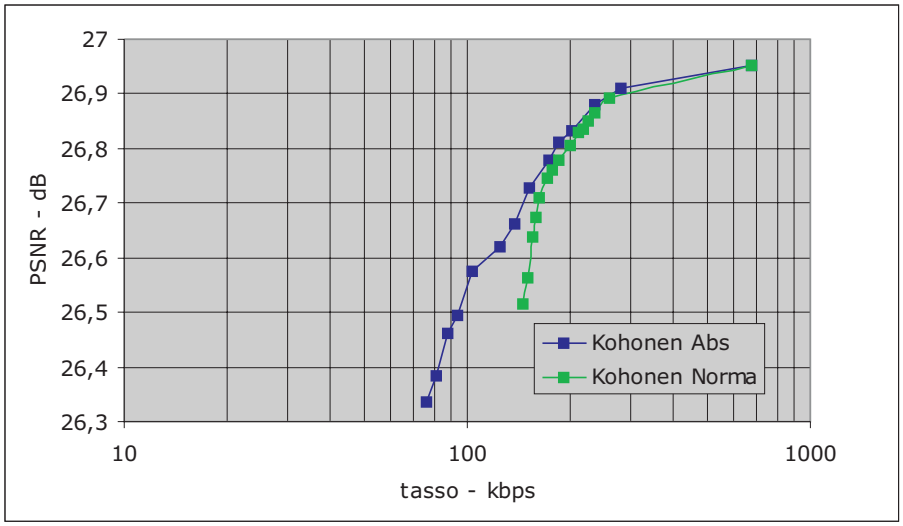

Figura 3.31: Metriche a confronto con il codebook di Kohonen a livello base.

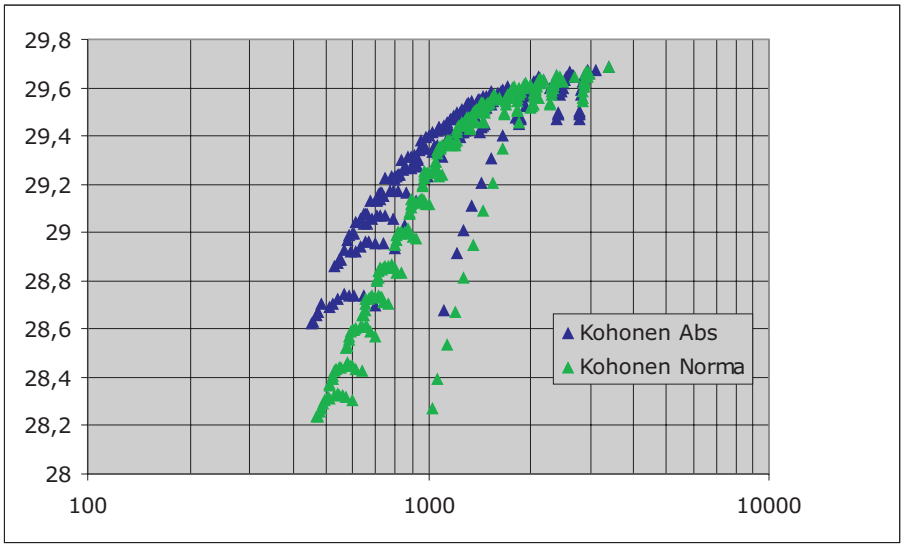

Figura 3.32: Metriche a confronto con il codebook di Kohonen a livello enhancement.

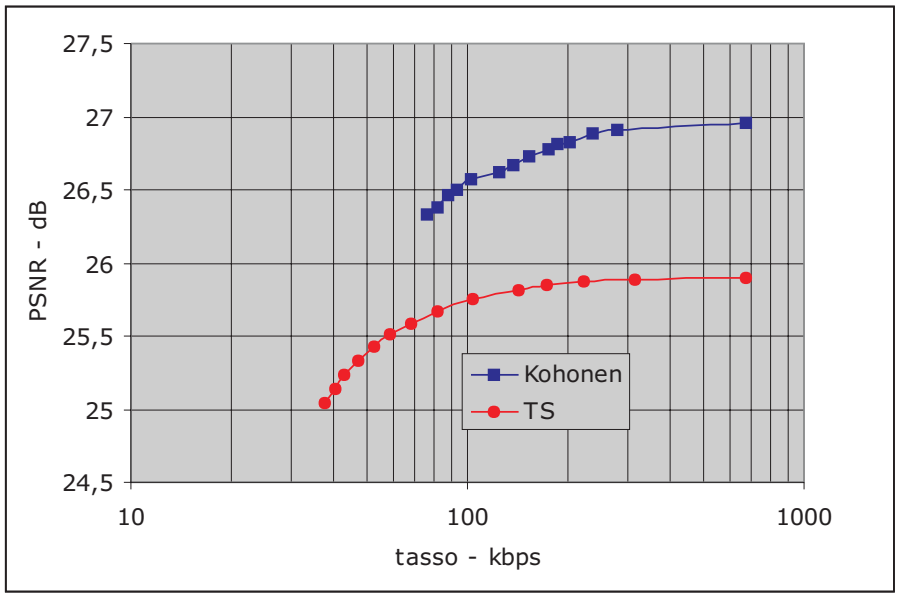

Figura 3.33: Confronto tra i codebook TS e di Kohonen con la nuova metrica, livello base.

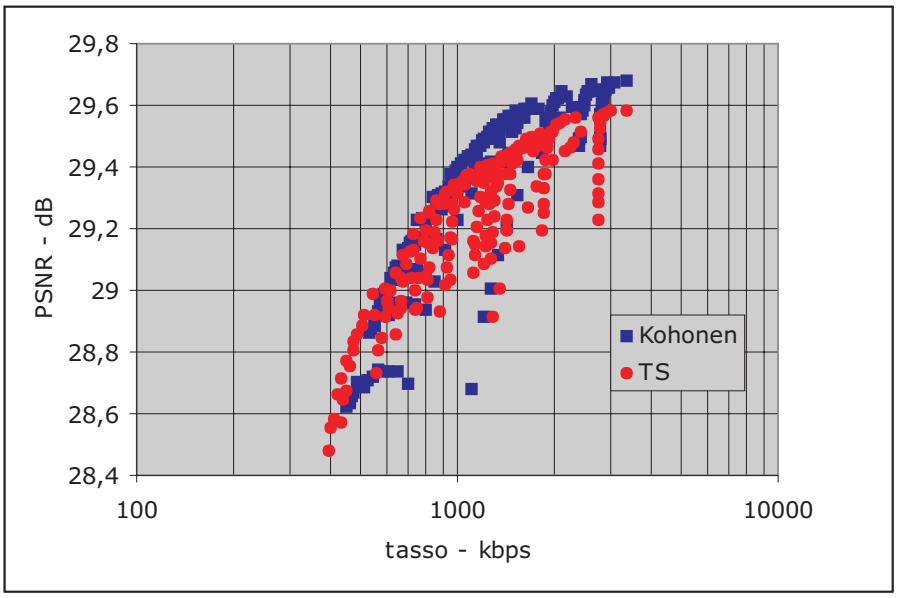

Figura 3.34: Confronto tra i codebook TS e di Kohonen con la nuova metrica, livello enhancement.

3.8 Conclusioni

In questo lavoro di tesi sono stati analizzati alcuni prototipi di codec video a bassa complessità e scalabili precedentemente realizzati, e poi si è cercato di migliorare quello che si è verificato più adatto ai nostri obiettivi. Nelle figure 3.35 e 3.36 si confrontano le prestazioni del prototipo dal quale siamo partiti con quelle di una versione modificata, in cui si fa uso delle varie tecniche illustrate nel capitolo:

- si è usato il test del CR indipendente al livello enhancement;
- si è ricorso al filtraggio antialiasing Daubechies 4;
- la compattazione degli indirizzi è stata ottenuta con l'APVQ.
- come codebook sono state usate le mappe di Kohonen;
- la metrica del CR è quella "a valore assoluto";

Queste tecniche consentono un netto miglioramento delle prestazioni, al prezzo di un aumento del tempo di codifica (che passa da 37 a 47 ms per frame) dovuto alla più complessa tecnica di filtraggio, e solo in parte compensato dalla maggiore semplicità della metrica per il CR.

Nelle figure da 3.37 a 3.38 sono mostrate le prestazioni di un prototipo che invece usa tutte le tecniche citate eccettuato il fatto che filtraggio anti-aliasing è fatto con un filtro di Haar, come nei prototipi iniziali. Si nota allora che, a fronte di un guadagno sulle prestazioni più limitato rispetto al caso precedente (almeno a livello enhancement), si riscontrano tempi di codifica inferiori anche rispetto al prototipo "veloce" di partenza, perché si beneficia della diversa metrica usata per il test del CR: con questo schema si riesce infatti a codificare un frame ogni 30 ms.

In questo lavoro sono state analizzate le potenzialità dello schema di codifica video basato su TS-HVQ e CR in modo abbastanza ampio, anche se rimangono alcuni aspetti da esplorare:

- lo sviluppo di tecniche di filtraggio veloce, basate ad esempio sui lifting scheme;

- lo studio della possibilità di usare la codifica a sottobande al posto di quella piramidale per gestire la scalabilità in risoluzione;
- l'eliminazione delle frame intra e la codifica dei blocchi di sfondo distribuita su più frame, analogamente a quanto previsto nello standard H.261.

Comunque gli esperimenti condotti permettono di concludere che, grazie all'uso delle tecniche di CR sugli indirizzi e di APVQ, è possibile implementare un algoritmo computazionalmente molto semplice e capace di generare un flusso video di qualità accettabile.

Questo però non è l'unico possibile *framework* per lo sviluppo di un codec video a scalabile a bassa complessità: ad esempio in [4], viene proposto uno schema basato sulla DCT, la codifica a sottobande ed il conditional replenishment. Si tratta di un'impostazione completamente alternativa alla nostra, e che si presenta molto interessante sia dal punto di vista delle tecniche usate che da quello delle prestazioni, per cui un'altra possibile strada che potrebbe essere seguita nel progetto potrebbe proprio essere quella di implementare questo schema alternativo, in modo da fornire anche un valido banco di prova alle tecniche fin qui implementate.

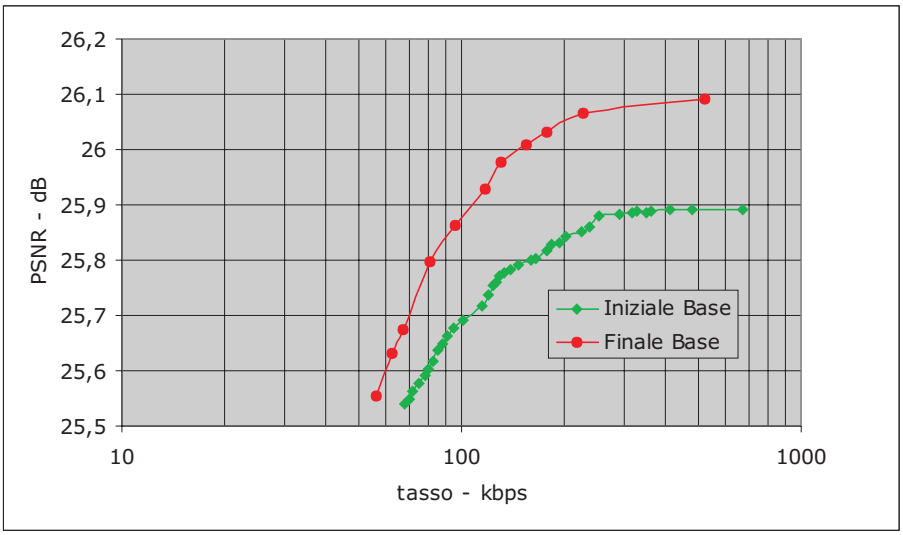

Figura 3.35: Confronto tra le prestazione al livello base del prototipo iniziale e quello che utilizza tutte le tecniche illustrate.

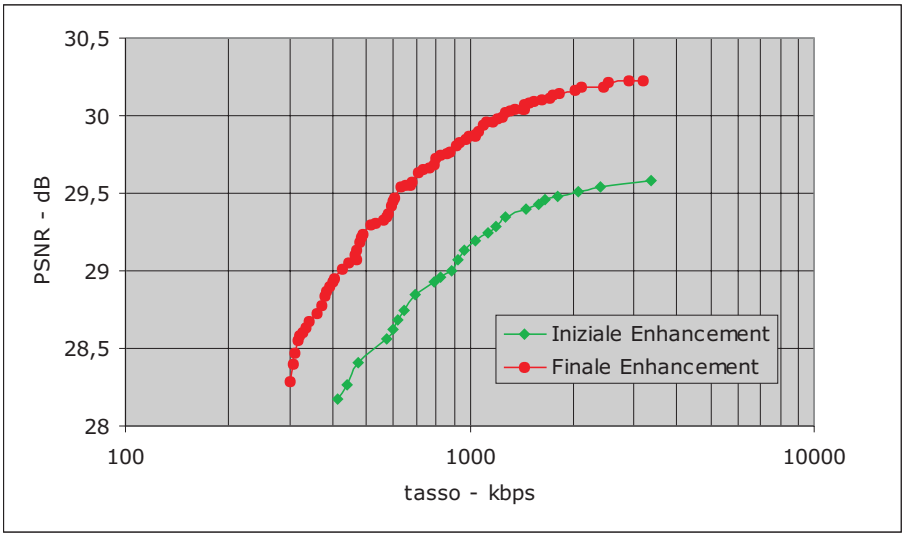

Figura 3.36: Confronto tra le prestazione al livello enhancement del prototipo iniziale e quello che utilizza tutte le tecniche illustrate.

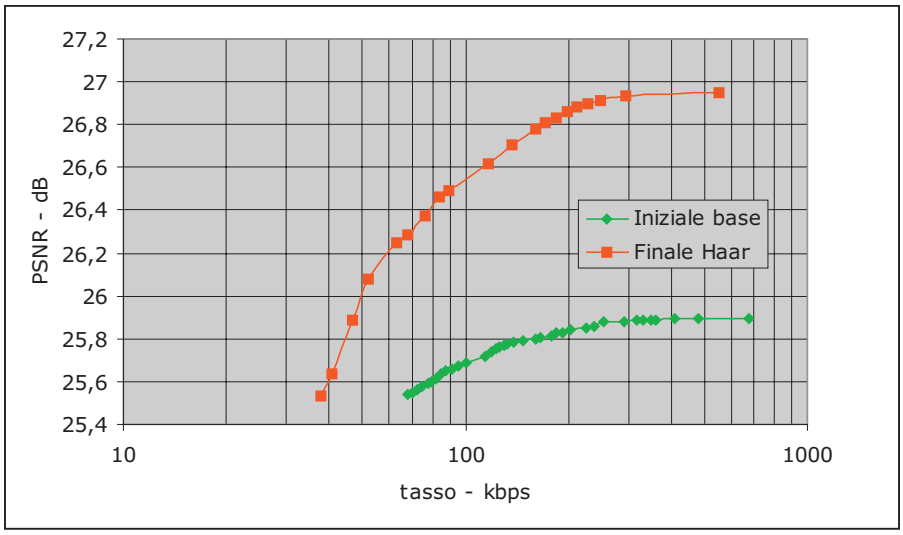

Figura 3.37: Confronto tra le prestazione al livello base del prototipo iniziale e quello che utilizza tutte le tecniche illustrate tranne il filtraggio di Daubechies.

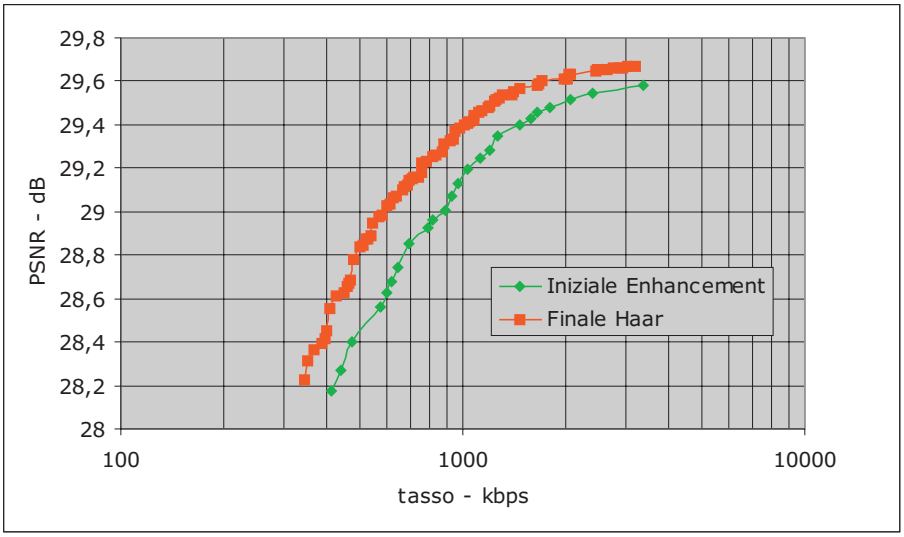

Figura 3.38: Confronto tra le prestazione al livello enhancement del prototipo iniziale e quello che utilizza tutte le tecniche illustrate tranne il filtraggio di Daubechies.

Bibliografia

[1] G.J. Conklin, G.S. Greenbaum, K.O Lillevold, A.F. Lippman, and Y.A. Reznik, "Video Coding for Streming Media Delivery on the Internet", *IEEE Trans. on Circuits and System for Video Technology*, vol. 11, n. 3, pp. 269-281, march 2001.

[2] D. Wu, Y.T. Hou, W.Zhu, Y. Zhang, and J.M. Peha, "Streaming Video over the Internet: Approach and Directions", *IEEE Trans. on Circuits and System for Video Technology*, vol. 11, n. 3, pp. 282-300, march 2001.

[3] A. S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice Hall International Editions, 1996.

[4] S. McCanne, M. Vetterli, and V. Jacobson, "Low-Complexity Video Coding for Receiver-Driven Layered Multicast", *IEEE Journal Select. Areas Commun.*, vol. 15 pp.983-1000, August 1997.

[5] N. Chaddha, and A. Gupta, "A Frame-work for Live Multicast of Video Streams over the Internet", *International Conference on Image Processing*, 1996.

[6] P. Parlato, *Codifica video a bassa complessità per la trasmissione in tempo reale su reti eterogenee*, Università Federico II di Napoli, Tesi di laurea in Ingegneria delle Telecomunicazioni, maggio 2001.

[7] N. Chaddha, M. Vishvanath, and P. A. Chou, "Hierarchical Vector Quantization of Perceptually Weighted Block Transforms", *Proc. of data Compression Conference*, March 1995

[8] N. Chaddha, P. A. Chou, and R. M. Gray, “Constrained and recursive Hierarchical Table-Lookup Vector Quantization”, *Proc. of data Compression Conference*, April 1996

[9] G. Franceschetti e I. Ghidini, *La televisione da satellite*, Ed. CUEN, 1999

[10] T. M. Cover, and J. A. Thomas, *Element of Information Theory*, John Wiley and sons, 1991.

[11] A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[12] A. Bovik, *Handbook of image and video compression*, Academic Press, 2000.

[13] T. Sikora, “MPEG Digital Video-Coding Standard”, *IEEE Signal Processing Magazine*, september 1997.

[14] G. Poggi, “Address-Predictive Vector Quantization of Images by Topology-Preserving Codebook Ordering”, *Euro. Trans. Telecommun.*, vol. 4, pp. 423-434, July-August 1993.

[15] R.-Y. Wang, E. A. Riskin, and R. Ladner, “Codebook Organization to enhance Maximum A Posteriori Detection of Progressive Trasmission of Vector Quantized Images over Noisy Channels”, *IEEE trans. on Image Processing*, vol.5, n.1, pp. 37-48, january 1996.

[16] W. Sweldens and P. Schröder, “Building your own wavelet at home”, in *Wavelets in Computer Graphics*, ACM SIGGRAPH Course notes, pp.15-87, 1996.

[17] K. Sayood, *Introduction to data compression*, 2nd ed., Morgan Kauffmann, 2000.

[18] T. Kohonen. *Self-Organization and Associative Memory*, 2nd ed., Springer Verlaf, 1988.

Indice analitico