UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Tesi di Laurea in

INGEGNERIA DELL'INFORMAZIONE

Codifica audio lossless

Relatore

Prof. Giancarlo Calvagno

Candidato

Mattia Bonomo

Anno Accademico 2009/2010

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Tesi di Laurea in

INGEGNERIA DELL'INFORMAZIONE

Codifica audio lossless

Relatore

Prof. Giancarlo Calvagno

Candidato

Mattia Bonomo

Anno Accademico 2009/2010

Ai miei nonni.

Indice

Ringraziamenti

Desidero ringraziare dal più profondo del cuore tutta la mia famiglia, che in questi tre anni mi è stata sempre vicino, in tutti i momenti. Perché questo primo frutto dei miei studi è anche un po' vostro, vi sarò eternamente grato per l'incoraggiamento, la dedizione, la disponibilità e l'affetto che mi avete sempre portato.

Grazie Mamma Ale, Papà Mauro, Selene, Katia e Chicco.

Un caro abbraccio va anche ai miei nonni, a cui dedico questa tesi, e senza i quali non sarei mai potuto arrivare a questo primo importante traguardo, perché nei momenti più difficili che mi hanno accompagnato voi siete sempre stati vivi e presenti dentro me.

Grazie Pina,Anna,Fiorenzo e Roberto el conte.

Non posso non pensare a te, che in questi anni mi hai trascinato come nave in un mare in burrasca, facendomi vedere la profondità dell'oceano e portandomi sulle spiagge più belle, delle quali mi sono innamorato.

Grazie Jessica.

Un sorriso anche ai miei coinquilini, per le simpatiche nottate, i pomeriggi in aula studio e per tutte le immonde scelleratezze che la nostra casa ha sentito.

Grazie, Albi, Fede, Enri, Bige.

Ai nazareni per acquisizione, compagni di tante avventure,

Grazie Scar, Vander, Wolver, Luca, Alfredo, Kekko, Toni.

Ultimi, ma non ultimi, tutti gli amici di Roana, perchè con voi ho trascorso moltissimi giorni che porterò nel cuore.

Un sentito e doveroso ringraziamento al Prof. Giancarlo Calvagno che mi ha dato la possibilità di fare questa tesi, che si è sempre dimostrato cordiale e disponibile nei miei confronti e che ho avuto il graditissimo piacere di conoscere.

Capitolo 1

Introduzione

Al giorno d'oggi molto lavoro è stato svolto per quanto concerne la codifica audio e voce con perdita (lossy) mentre ancora relativamente poco è stato pubblicato sulla codifica senza perdita (lossless).
La motivazione principale del perchè è molto semplice, le codifiche lossless consentono di ottenere un rapporto di compressione all'incirca di 3:1 , nemmeno paragonabile al rapporto 12:1, e anche superiore, delle codifiche lossy.
Risulta evidente quindi che la codifica audio lossless non è destinata a diventare la tecnologia dominante, però può sicuramente divenire un utilissima alternativa alla codifica con perdita in molte applicazioni.
Gli algoritmi lossy sono sicuramente l'unica alternativa possibile quando viene richiesto di occupare il minor spazio possibile, a scapito ovviamente della qualità e della fedeltà del segnale compresso. Tuttavia la codifica lossless di un CD audio digitale, campionato a 44.1 kHz e quantizzato a 16 bit può diventare una tecnologia essenziale per quanto riguarda la distribuzione digitale attraverso Internet, perchè molti consumatori vorranno avere a di-sposizione un file audio più fedele possibile all'originale in modo da poter sfruttare al meglio i dipositivi audio di riproduzione.
Ovviamente Internet non è l'unica applicazione possibile, basti pensare che questa tecnologia può essere utilizzata anche per l'archiviazione e il mixing di registrazioni ad alta fedeltà in ambienti professionali. In questo caso la compressione lossless evita il degrado generato durante la continua codifica e decodifica del file originale.
La codifica audio lossless è attualmente lo standard per i dispositivi DVD ad alta fedeltà. Si evince da queste considerazioni come l'oggetto in questione di questa tesina sia di attualità, motivo in più per il quale merita di essere approfondito.

Struttura della tesi

- **Capitolo 2**: in questo capitolo saranno introdotti i concetti principali della codifica lossless tramite un'analisi approfondita di ogni singolo passaggio attraverso il quale il segnale audio originale diviene un segnale compresso lossless. In particolare vengono analizzate la parti del coder quali: blocking, inter ed intra channel correlation, entropy coding e framing.

- **Capitolo 3**: in questo capitolo sarà esaminato lo stato dell'arte dei principali codec lossless. Una tabella comparativa consentirà di capire come operano i diversi codec e metterà in evidenza pregi e difetti di ognuno di essi, al fine di poter scegliere il più adatto alle proprie esigenze. La seconda parte introduce allo standard MPEG4, che ha sviluppato due diverse tipologie di codec lossless, una che si basa sulla predizione e un'altra che si basa sulle trasformate.

- **Capitolo 4**: in questo capitolo vi sarà spazio per un'analisi approfondita dello standard MPEG4-ALS. Saranno analizzate tutte le fasi della codifica, ed in particolare sarà approfondita in particolare la parte riguardante la predizione lineare, l'algoritmo RLS-LMS è una peculiarità di questo codec che lo differenzia profondamente dagli altri in commercio.

- **Capitolo 5**: in questo capitolo vi sarà spazio per un'analisi approfondita dello standard MPEG4-SLS. L'approcio del codec in questo caso è molto diverso dal precedente, in quanto esso sfrutta trasformate intere nel dominio della frequenza per ottenere la compressione. Saranno analizzate tutte le fasi della codifica, prestando particolare attenzione per la trasformazione intera e la codifica entropica, punti di forza del codec in questione.

- **Capitolo 6**: in questo capitolo sono svolti numerosi test su campioni audio del codec MPEG4-ALS, utilizzando le numerose opzioni di codifica che lo standard offre. Le simulazioni sono fatte per comprendere come, per il risultato finale, siano determinanti: l'algoritmo RLS-LMS, la codifica adattiva, e la codifica inter-channel. Tutti i risultati ottenuti sono riportati in tabelle e grafici.

- **Capitolo 7**: in questo capitolo si introduce lo sviluppo in ambiente Matlab di un software per la codifica lossless. Tale software è stato implementato sulle basi delle conoscenze ottenute durante la stesura dei capitoli precedenti. In

particolare è stato scelto di implementare un codec a predizione lineare che sfrutta l'algortimo adattivo MEQP (minimo errore quadratico pesato), il cui errore viene quindi codificato tramite il metodo di Golomb Rice. A seguito dello sviluppo sono anche stati effettuati test con diverse tracce audio, e test comparativi con codec lossless commerciali. Tutti i risultati ottenuti sono riportati in grafici e tabelle.

- **Capitolo 8**: Conclusioni.

Capitolo 2

Principi della compressione lossless

Le tecniche studiate per la codifica lossless hanno lo scopo di ridurre al minimo la ridondanza dal segnale e successivamente cofidicare il segnale in maniera più efficiente possibile, in modo da ottenere un segnale compresso con le stesse informazioni dell'originale.
In linea di principio la maggior parte degli algoritmi segue uno schema del tipo: blocking -interchannel decorrelation - intrachannel decorrelation - entropy coding - framing, come illustrato in Figura 2.1. Di seguito verranno approfonditi in dettaglio tutti i passaggi che consentono di arrivare ad un segnale audio codificato senza perdite partendo da un segnale audio in formato wave.

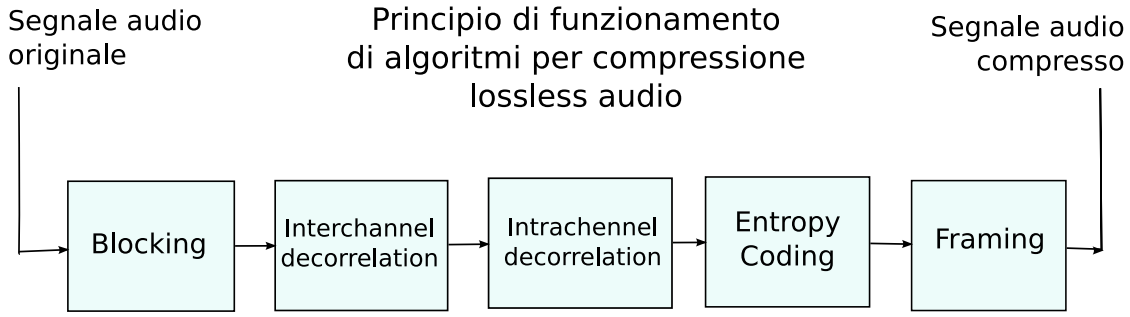


Fig. 2.1: Operazioni base nella maggior parte degli algoritmi lossless.

2.1 Blocking e Framing

Quando ci si appresta a codificare un segnale audio è importante avere chiaro già dal principio in che modo andremo a elaborare tale segnale. Dal momento che elaborare un'intera traccia audio risulta essere un'operazione complessa, ed inoltre non consente di raggiungere risultati ottimi, si è reso necessario trovare dei metodi efficienti per scomporre il segnale originale, codificarlo e quindi garantire la perfetta ricostruzione e riproduzione.
Nello specifico, la maggior parte degli algoritmi procede in questo modo: una prima parte, *blocking* divide il segnale audio originale in blocchi di dimensione fissa o variabile, più facilmente elaborabili, mentre una seconda parte, posta solitamente alla fine del processo di codifica, chiamata *framing* oltre ad aggiungere informazioni utili per la decodifica del segnale compresso, provvede ad unire i frame di segnale codificato.

Per spiegare meglio il seguente paragrafo faremo riferimento al codec FLAC [1], introducendo alcune utili definizioni:

- *Blocco*: uno o più campioni audio che comprendono più di un canale
- *Sottoblocco*: uno o più campioni audio all'interno di un singolo canale. Quindi un blocco contiene un sottoblocco per ogni canale.
- *Dimensioni blocco*: numero di campioni contenuti in ogni sottoblocco. Per esempio un secondo di segnale audio campionato a 44.1 KHz produce un blocco di dimensione 44100.
- *Frame*: contiene un frame di intestazione più i subframe.
- *Subframe*: contiene un subframe di intestazione e uno, o più, campioni codificati da un determinato canale. Tutti i subframe all'interno di un frame contengono lo stesso numero di campioni.

E' importante osservare che parliamo di blocco e sottoblocco per il segnale che deve ancora essere codificato, mentre di frame e subframe per il segnale già codificato.

[1] FLAC è l'acronimo di: Free Audio Lossless Codec. Generalmente la maggior parte degli algoritmi esegue operazioni simili se non uguali a quelle illustrate.

Blocking Per blocking, si intende l'operazione con la quale si va a suddividere il segnale audio di ingresso in blocchi, di durata finita o variabile a seconda dell'algoritmo utilizzato.
Il motivo per cui viene svolta questa operazione è sostanzialmente quello di rendere meno complesse le operazioni di codifica e decodifica. La restrizione applicata al segnale audio consente all'algoritmo di lavorare all'interno di un dominio che è più facile da elaborare. La dimensione utilizzata per il blocking è molto importante. Se infatti vengono presi dei blocchi troppo piccoli, avremo uno spreco inutile di bit nei metadati di intestazione (che approfondiremo in seguito), mentre se prendiamo dei blocchi troppo grandi sarà difficile ottenere un risultato ottimo, a causa della difficile adattività temporale dei predittori.
Per quanto riguarda lo standard FLAC, è previsto un range di dimensioni predefinito del blocco: da 16 a 65536 campioni.

Framing Questa operazione, spesso chiamata anche format, prevede di condensare in un unico frame tutti i subframe di segnale codificato, aggiungendo informazioni utili per la riproduzione e la decodifica del segnale audio codificato.

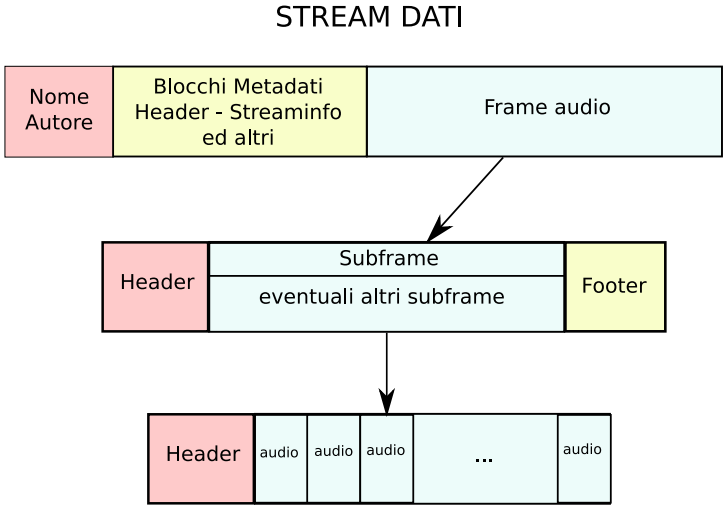


Fig. 2.2: Suddivisione dello stream dati.

Facendo riferimento alla Figura 2.2, anche in questo paragrafo sarà utile introdurre alcune utili definizioni:

- *Stream*: Lo stream è il flusso di dati costituito da: il nome dell'algoritmo o dell'autore, un blocco di metadati obbligatiorio chiamato *Streaminfo*, dei

blocchi di metadati aggiuntivi opzionali e uno o più frame audio. Ogni blocco metadati è preceduto da un header che specifica se esso è l'ultimo prima dei frame audio e ne definisce il tipo. Infatti a seconda dell'algoritmo possono essere infatti previsti diversi tipi di blocchi metadati, ad esempio: tabelle di ricerca, per ridurre il ritardo, vorbis-comment per aggiungere commenti alla traccia audio, picture per allegare delle immagini e altro.

- *Streaminfo*: è il blocco metadati più importante per la corretta ricostruzione del segnale audio: è composto da: 16 bit che definiscono la dimensione minima del blocco, 16 bit che definiscono la dimensione massima del blocco, 24 bit che definiscono la dimensione minima del frame, 24 bit che definiscono la dimensione massima del frame, 20 bit che definiscono la frequenza di campionamento, 3 bit che definiscono il numero di canali.

- *Frame*: è composto da

  – Frame header: inizia con il codice di sincronizzazione e contiene informazioni riguardanti tipo di blocco (dimensione fissa o variabile), campioni per blocco, frequenza campionamento, numero bit per campione, genera un CRC (Controllo a Ridondanza Ciclica) a 8 bit.

  – Subframe: a sua volta composto da un header contente le informazioni minime per la decodifica e un numero definito di altri subframe contenenti campioni del segnale codificato.

  – Frame footer: è il blocco conclusivo, genera un CRC a 16 bit per rilevare eventuali errori. In tal caso viene prodotto un blocco contenente tutti zeri.

Dal momento che il decoder potrebbe iniziare la decodifica in un punto qualsiasi dello stream, è necessario che sia presente un metodo per determinare tutte le informazioni indispensabili per la corretta riproduzione del segnale. Un codice di sincronia a 14 bit è posto all'inizio di ogni frame, tale codice appare nel frame header e, ovviamente, negli header di ogni substream.
Oltre alla lettura del frame header, peraltro non sempre sicura, il decoder può verificare l'integrità dei dati letti confrontando il CRC a 8 bit generato partendo dal frame header con quello contenuto alla fine del suddetto frame.
Si spiega quindi il motivo principale per cui vi è una ridonzanza delle informazioni all'interno del segnale codificato. Non è possibile garantire sempre l'accesso al

metadato streaminfo dal ricevitore, quindi le informazioni base sono riportate periodicamente nel segnale. Ovviamente i frame e subframe header sono di dimensioni molto ridotte, tali da non pregiudicare la bontà della compressione ottenuta.

2.2 Interchannel decorrelation

Un segnale audio contenuto in un cd audio ha due canali separati left e right. Il proposito di questo blocco è quello di rimuovere la ridondanza che esiste tra essi. Il metodo più comune per svolgere questo tipo di operazione è chiamato *mid / side encoding*.
Negli ultimi anni sono stati proposti pochi altri metodi efficienti, tra i quali però vale la pena di citare: *Lossless matrix* usata dall'algoritmo MLP, e *Joint stereo prediction* usato ad esempio da MPEG4 - ALS.

Mid / Side encoding E' probabilmente il metodo più conosciuto e anche il più semplice. Esso si propone di ridurre la correlazione trasformando i due canali, left e right, in un segnale somma (mid) ed in un segnali differenza (side). La maggior parte degli algoritmi sfrutta questa tecnica anche per segnali più complessi, ad esempio la codifica FLAC di un file in formato dolby 5.1 prevede di codificare i canali accorpandoli in segnali stereo e mono.

Lossless Matrix Il codec MLP (Meridian Lossless Packaging) utilizza una matrice, vedi Figura 2.3, che consente all'encoder di ridurre la correlazione concentrando i segnali con ampiezza maggiore in un numero minore di canali. Le matrici convenzionali non sono lossless, una matrice inversa convenzionale infatti ricostruisce il segnale con degli arrotondamenti che implicano degli errori.
Il codec MLP scompone la matrice generale in una cascata di trasformazioni affini. Ogni trasformazione affine modifica solamente un canale, addizionando una combinazione lineare quantizzata degli altri canali. A questo punto avremo dei canali ottimizzati, la cui codifica entropica porterà a risultati migliori. Il codec MLP è utilizzato come standard di compressione per i DVD-AUDIO.

Joint Stereo Prediction Come si evince dalla Figura 2.4 questo metodo unisce inter ed intra channel correlation.

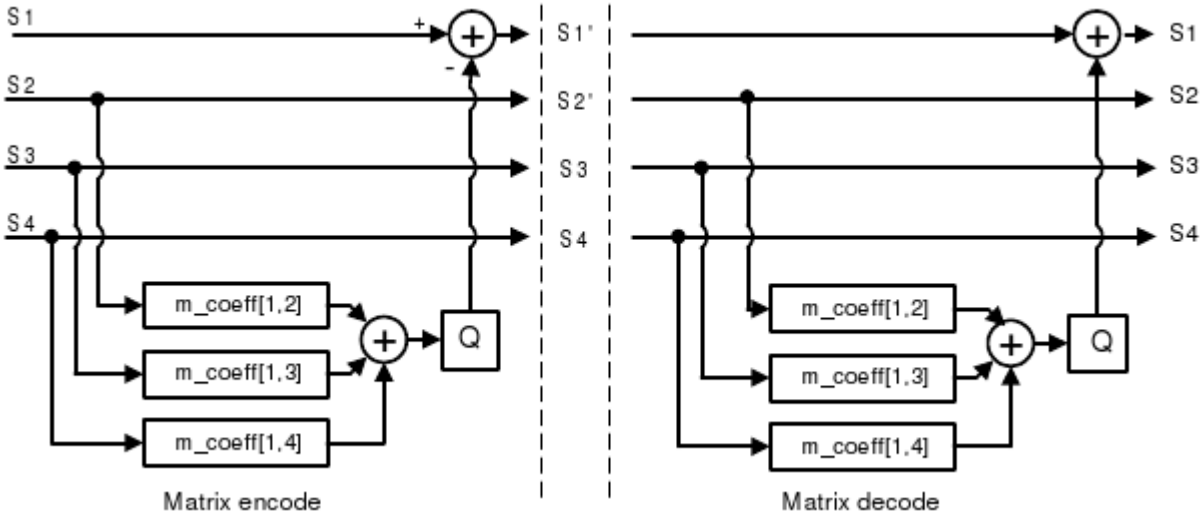


Fig. 2.3: Funzionamento matrice lossless codifica e decodifica.

Il predittore intrachannel a_L genera una stima del canale left mentre allo stesso tempo il predittore interchannel b_L genera una stima usando i campioni del canale right. La stima del predittore globale per il canale left è ottenuta come somma delle due precedenti stime.
Siano M_a e M_b l'ordine del predittore a_L e b_L rispettivamente, allora la stima per il canale left è data da

$$y_L(n) = \sum_{m=1}^{M_a} a_{L,m} x_L(n-m) + \sum_{m=1}^{M_b} b_{L,m} x_R(n-m)$$

ove $a_{L,m}$ e $b_{L,m}$ sono i coefficienti dei predittori a_L e b_L rispettivamente.
In modo simile si ricava che

$$y_R(n) = \sum_{m=1}^{M_a} a_{R,m} x_R(n-m) + \sum_{m=0}^{M_b-1} b_{R,m} x_L(n-m)$$

dove però notiamo che il secondo termine della sommatoria parte da 0 anzichè da 1. Il motivo è che, nei decoder ASL, i campioni audio sono ricostruiti in questo ordine:

$$\{\ldots, x_L(n-1), x_R(n-1), x_L(n), x_R(n), x_L(n+1), x_R(n+1), \ldots\}$$

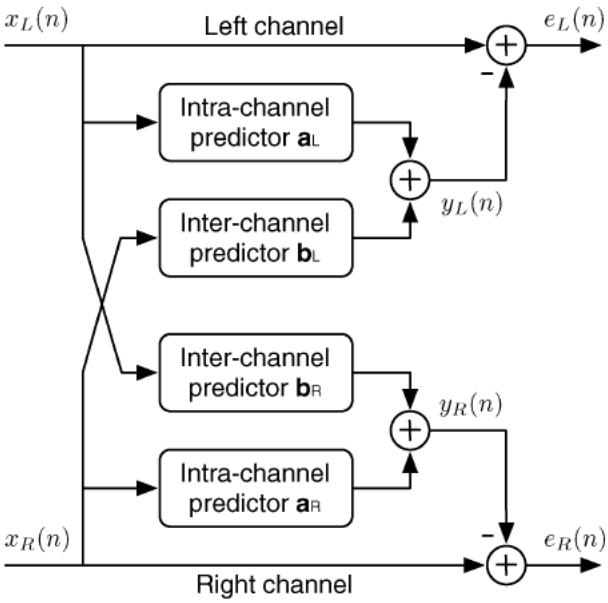


Fig. 2.4: Predizione Stereo Joint.

Questo ordine garantisce che i campioni del canale left siano decodificati prima dei campioni del canale right, il che spiega perchè si usano i campioni del canale left per decodificare il canale right.
L'algoritmo MPEG4-ALS utilizza come predittori una cascata ricorsiva di filtri RLS (recursive least square) e LMS (least mean square). Questa particolare tecnica sfrutta sia la correlazione intrachannel che interchannel, e consente un miglioramento delle prestazioni quantificabile in un 3 percento rispetto alla sola codifica intrachannel.
L'algoritmo MPEG4-ALS (del quale parleremo approfonditamente in seguito) offre anche una versione multicanale, MCC, che può essere utilizzata in sostituzione o in aggiunta a questa.

2.3 Intrachannel decorrelation

Lo scopo di questo terzo blocco è quello di decorrelare il segnale in ingresso all'interno di ogni singolo blocco quindi di ridurre la sua energia prima di codificarlo. La maggior parte degli algoritmi (Monkey's audio, DVD, Oggsquish, Philips...) usa sostanzialmente due tipi di approccio, che rappresentano due linee di pensiero molto differenti per raggiungere lo stesso risultato.

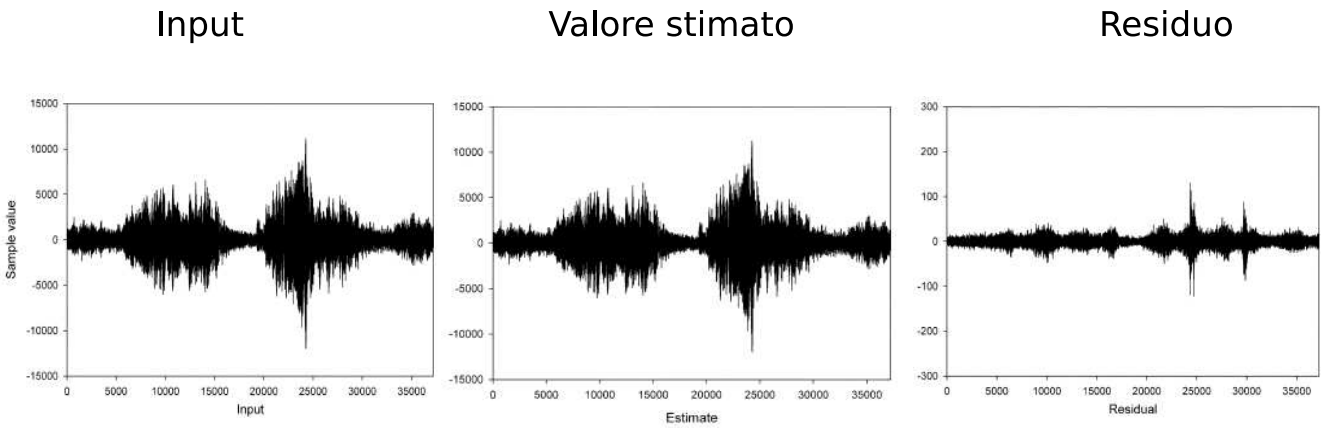
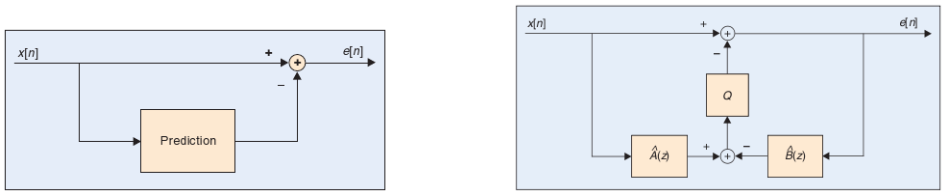


Fig. 2.5: Grafici di: segnale di ingresso, valori stimati, differenza residua.

L



(a) `Modello_predizione.` (b) `Struttura_predittore_lineare.`

Fig. 2.6: modelli di predizione

Il primo tipo rimuove la ridondanza tramite un particolare blocco chiamato *predittore lineare*.
In questo approcio al predittore lineare è applicato il segnale di ingresso campionato. In uscita esso genera una sequenza di campioni di errore di predizione, mentre i parametri del predittore rappresentano la ridondanza rimossa dal segnale. Pertanto il segnale originale potrà essere ottenuto come somma tra i parametri codificati e gli errori di predizione. La bontà del predittore è espressa come rapporto tra la potenza del segnale originale e la potenza del residuo.

Il secondo tipo di approcio, chiamato *codifica lossless con trasformate* è quello di ottenere una rappresentazione lossy del segnale, e quindi successivamente comprimere senza perdite di informazione la differenza tra la versione lossy e quella originale del segnale. Questo tipo di codifica ibrida opera nel dominio delle frequenze, e offre l'indubbio vantaggio di poter ottenere sia la versione lossy che

quella lossless del segnale.
Generalmente questo tipo di approcio prevede l'uso di trasformate intere, come *IntWT, IntDCT* oppure *IntWHT*.
Andiamo ora ad analizzare con attenzione queste due diverse metodologie di approcio.

Predittori lineari Analizziamo innanzitutto il comportamento del predittore nel dominio della frequenza.

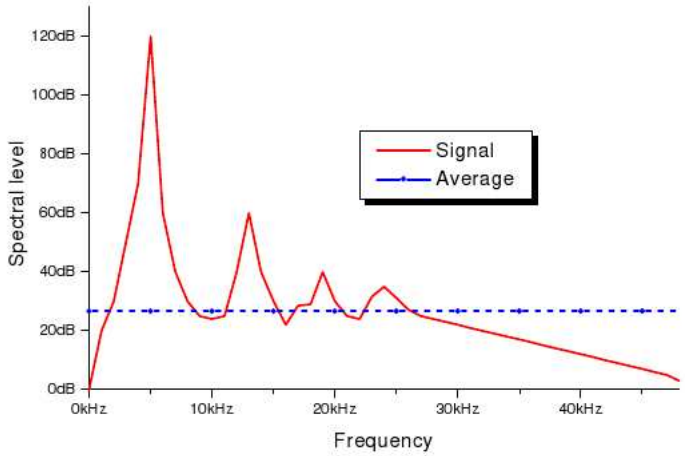


Fig. 2.7: Spettro e media del segnale.

La Figura 2.7 rappresenta lo spettro di uno spezzone di segnale.
Idealmente un predittore dovrebbe produrre un segnale residuo a spettro piatto, essenzialmente come rumore bianco. Il teorema di *Gerzon - Craven* mostra come il livello ottimo del segnale decorrelato sia dato dalla media dello spettro del segnale originale, illustrato come in Figura 2.7. Come si evince facilmente dal grafico, la media ha energia minore e quindi richiede una quantità minore di bit per essere rappresentata. In pratica il grado con cui è possibile sbiancare un file audio dipende dalla sua complessità, di conseguenza la complessità del predittore. Nel cercare di creare un predittore che riesca ad appiattire il più possibile il segnale bisogna comunque cercare di renderlo relativamente semplice, in quanto i suoi coefficienti dovranno poi essere inviati.
Il principio base di questo blocco, come già accennato, è quello di predire i valori del campione $x[n]$ usando i precedenti campioni $x[n-1]$, $x[n-2]$, etc.
La Figura 2.6(a) rappresenta uno schema a blocchi del modello con il predittore. Se

il predittore è ben progettato, gli errori di predizione $e[n]$ saranno incorrelati quindi avremo una risposta in frequenza piatta . Allo stesso modo $e[n]$ sarà mediamente minore di $x[n]$ e quindi la sua rappresentazione digitale richiederà meno bit.
In Figura 2.6(b) si può osservare nel dettaglio la struttura del predittore. Dallo schema a blocchi si ricava facilmente

$$e[n] = x[n] - Q\left\{\sum_{k=1}^{M} \hat{a}_k x[n-k] - \sum_{k=1}^{M} \hat{b}_k e[n-k]\right\}$$

dove Q indica la quantizzazione della stessa lunghezza della parola del segnale originale $x[n]$.
Le due sommatorie $\sum_{k=1}^{M} \hat{a}_k x[n-k]$ e $\sum_{k=1}^{M} \hat{b}_k e[n-k]$ stanno ad indicare rispettivamente la trasformata polinomiale z della preazione $\hat{A}(z)$ e della retroazione $\hat{B}(z)$.
Si noti che per $\hat{B}(z) = 0$ si possono omettere i termini di retroazione, ottenendo quindi un filtro di predizione di tipo FIR (finite impulse response). In caso contrario il nostro filtro sarà di tipo IIR (infinite impulse response).
Le operazioni di quantizzazione rendono il predittore un dispositivo non lineare, ma, dato che la quantizzazione è a 16 bit di precisione, è ragionevole trascurarla per comprendere gli effetti del predittore e per lo sviluppo di metodi per la stima dei parametri del predittore.
La ricostruzione del segnale originale $x[n]$ partendo da $e[n]$ può essere fatta ricorrendo alla relazione

$$x[n] = e[n] + Q\left\{\sum_{k=1}^{M} \hat{a}_k x[n-k] - \sum_{k=1}^{M} \hat{b}_k e[n-k]\right\}$$

Come già accennato sopra, ad ogni finestra temporale viene determinato un nuovo set di coefficienti del predittore, in modo che esso possa essere il più adatto possibile al segnale in ingresso.
La scelta dei coefficienti offre generalmente due possibilità: scegliere dei coefficienti che minimizzino il più possibile l'errore quadratico medio sulla predizione (in accordo con il *criterio di autocorrelazione*) oppure la scelta di un predittore con diversi coefficienti da una libreria fissata, in questo caso viene di molto ridotto il costo computazionale.
Il secondo approcio è generalmente usato quando sia i termini di retroazione che quelli di preazione compaiono nel predittore, questo perchè cercare un predittore che minimizzi l'errore quadratico risulta un'operazione più complessa dal punto di vista e progettuale e computazionale. Ovviamente il risultato di questo tipo di

operazione non consente di ottenere predizioni ottime.
Nella compressione lossless comunque, come dimostrato da Craven, i predittori IIR hanno le potenzialità per operare molto meglio dei predittori FIR, in quanto essi possono usufruire di un più ampio bacino di possibili forme spettrali a parità di numero di coefficienti e consentono il controllo del data-rate di picco.
Riportiamo per chiarezza la Figura 2.8 che rappresenta l'approssimazione di un segnale generata da un filtro FIR di ordine 8, e da un filtro IIR di ordine 4.

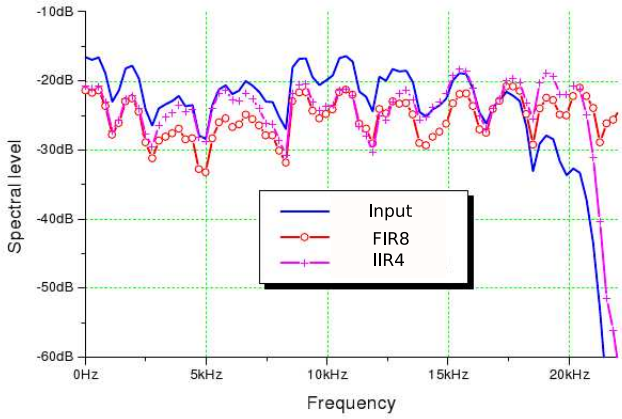


Fig. 2.8: Spettro di un estratto del segnale audio, sovrapposto alle approssimazioni di dei predittori IIR4 e FIR8.

Si noti come il filtro FIR abbia un comportamento molto buono, simile a quello del filtro IIR, sino ai 20 KHz dove però si evince come il filtro IIR lavora molto meglio con segnali la cui ampiezza cambia repentinamente.
Nel grafico in esempio i miglioramenti introdotti dal filtro IIR sono esigui, in quanto esso sfrutta le sue potenzialità solamente in una banda di 2 KHz circa, ma si può intuire come esso possa lavorare molto meglio di un filtro FIR anche di ordine molto superiore.
In generale i predittori a scarto quadratico medio minimo coinvolgono coefficienti frazionari, che nella rappresentazione a precisione finita possono subire approssimazioni non ottimali, pertanto se possibile è consigliabile utilizzare coefficienti interi, che garantiscono semplicità di calcolo maggiore e hannno la stessa rappresentazione su ogni tipo di architettura.
Recentemente algoritmi come *Monkey's Audio* o *The True Audio* hanno perfezionato l'uso di modelli IIR, ottenendo risultati molto positivi.

modello FIR	modello IIR
Shorten	OggSquish
Sonarc	DVD
WA	TTA
MusiCompress	Monkey's audio

Tabella 2.1: Modelli di predizione usati da diversi algoritmi.

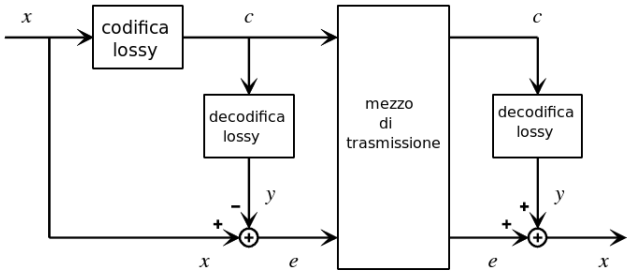


Fig. 2.9: Codifica lossless schematizzata come combinazione di blocchi lossy.

Codifica lossless con trasformate La codifica lossless con trasformate può essere considerata come una combinazione della convenzionale codifica lossy a cui va aggiunta la codifica degli errori. L'idea che sta alla base di questo metodo è illustrata in figura 2.9.
Il segnale codificato *c* è derivato dal segnale di ingresso *x* usando un algoritmo di compressione lossy. Per produrre un segnale lossless è necessaria anche la codifica dell'errore commesso, a tale scopo il segnale *e* è ottenuto come differenza tra il segnale *x* e il segnale ricostruito localmente *y*. Successivamente i due segnali sono codificati entropicamente e quindi spediti attraverso il mezzo trasmissivo.
La ricostruzione avviene facilmente conoscendo la regola di compressione lossy, semplicemente decodificando *c* e quindi sommando l'errore *e*.
Va da se che la scelta dell'algoritmo lossy che si utilizzerà andrà ad influire in maniera drastica sull'errore che si commetterà. La scelta di un fattore di compressione troppo alto, ad esempio, produce un errore grande al punto che esso diviene comparabile con il segnale di ingresso, a questo punto dovremmo de-correllare tale segnale e filtrarlo prima di inviarlo per ottenere una compressione decente, quindi la codifica non avrebbe alcun senso.
Nel caso opposto, la scelta di un fattore di compressione troppo piccolo porta ad avere un errore praticamente nullo, ma senza aver rimosso ridondanza.

Quindi l'obiettivo principale della codifica lossless con le trasformate è quello di ottenere il miglior compromesso tra una compressione elevata nel ramo lossy, e un segnale facilmente codificabile nel ramo correttivo.

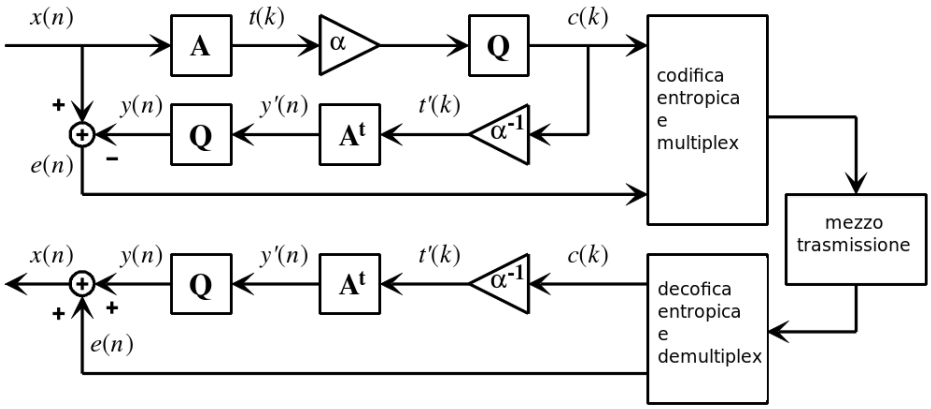


Fig. 2.10: Schema a blocchi della codifica lossless con trasformata.

Consideriamo ora la Figura 2.10.
Dal segnale quantizzato $x(n)$ è calcolato un insieme di coefficienti $t(k)$ usando un'arbitraria trasformazione ortonormale A con lunghezza della parola pari a N. I coefficienti sono quindi ridotti di un fattore α e quantizzati con passo $\Delta = 1$, che conduce al valore intero $c(k)$, pronto per essere codificato entropicamente e quindi trasmesso. Usando una trasformazione adatta si ottengono molti coefficienti piccoli o addirittura uguali a zero, che costituiscono una sorgente incorrelata, facilmente elaborabile senza dover tener conto delle probabilità congiunte.
Come risultato della quantizzazione, equivalente ad un arrotondamento all'intero, la decodifica di $c(k)$ non garantisce una perfetta ricostruzione del segnale, quindi si rende necessario un metodo per determinare ogni possibile errore di decodifica, se necessario.
Dopo aver ripristinato il valore originale tramite la funzione α^{-1} e aver anti-trasformato il segnale tramite[2] A^t, indicante la matrice trasposta di A, otteniamo il segnale $y^{'}(n)$, che a causa degli errori commessi non è uguale all'originale $x(n)$. Dal momento che consideriamo segnali interi, non c'è bisogno di ricostruire esattamente il segnale di ingresso da un segnale a valori reali. Se $|y^{'}(n) - x(n)| < 0.5$ allora il troncamento di $y^{'}(n)$ porta a un segnale ricostruito $y(n)$ perfettamente identico all'originale, altrimenti sarà presente un errore $e(n) \neq 0$. Tale errore sarà certamente

[2] In generale si usa di norma la matrice inversa, in questo caso specifico può essere usata la matrice trasposta dal momento che le due coincidono.

intero, in quanto ottenuto come $e(n) = x(n) - y(n)$ entrambi a valori interi.
Il segnale originale è ottenuto quindi addizionando $e(n)$ a $y(n)$, infatti:

$$y(n) + e(n) = y(n) + [x(n) - y(n)] = x(n)$$

2.4 Entropy Coding

La codifica entropica ha lo scopo di rimuovere definitivamente la ridondanza residua dal segnale decorrelato, quindi di ottimizzare il codice del segnale in ingresso prima che questo venga spedito sul mezzo trasmissivo. La codifica entropica ha sostanzialmente lo scopo di assegnare ad ogni parola un codice specifico a seconda della sua probabilità di occorrenza. Pertanto valori molto probabili avranno bisogno di pochi bit per essere rappresentati, mentre valori poco probabili necessiteranno di codifiche molto più lunghe.
I principali algoritmi che analizzeremo sono:

- *RLE*
- *Huffman*
- *codici aritmetici*
- *codice di Rice*
- *LZ77, LZ78*
- *DEFLATE*

RLE L'algoritmo RLE (run-length encoding) è storicamente il primo algoritmo utilizzato per la compressione delle immagini. L'algoritmo RLE cerca nei dati da comprimere una serie di elementi uguali e la sostituisce con un solo elemento, un carattere speciale, che rappresenta il numero di ripetizioni e il carattere sostituito.

Huffman Per Codifica di Huffman si intende un algoritmo di codifica entropica usato per la compressione di dati, basato sul principio di trovare il sistema ottimale per codificare stringhe sfruttando la frequenza relativa di ciascun carattere. La codifica di Huffman usa un metodo specifico per scegliere la rappresentazione di

ciascun simbolo, producendo un codice senza prefissi (cioè in cui nessuna stringa binaria di nessun simbolo è il prefisso della stringa binaria di nessun altro simbolo) che esprime il carattere più frequente nella maniera più breve possibile. È stato dimostrato che la codifica di Huffman è il più efficiente sistema di compressione di questo tipo: nessun'altra mappatura di simboli in stringhe binarie può produrre un risultato più breve nel caso in cui le frequenze di simboli effettive corrispondono a quelle usate per creare il codice.
E' però importante notare che il codice di Huffman è estremamente vulnerabile, nel senso che un errore su un bit può comportare l'errata decodifica dell'intero messaggio. Questo fatto costringe a modificare leggermente il codice, introducendo bit aggiuntivi per il check degli errori. Il procedimento di questo algoritmo può essere descritto come segue (ove N è il numero di parole di ingresso) :

- Se $N = 2$ allora banalmente una parola sarà codificata con il bit 0, mentre l'altra con il bit 1.
- se $N > 2$ allora:
    1. Ordinare le N probabilità in modo decrescente.
    2. Le parole meno probabili hanno codifica più lunga. Posto che p_N e p_{N-1} sono le probabilità più basse, assegnamo alla maggiore tra le due il bit 0, alla minore il bit 1, quindi si esegue la somma tra le due ottenendo la nuova probabilità $p'_{N-1} = p_N + p_{N-1}$.
    3. A questo punto itero il procedimento, considerando questa volta le probabilità p_{N-2} e p'_{N-1}.
    4. All'ultimo passo assegno il bit 1 (o il bit 0) alla $p(1)$, e quindi leggo a ritroso le codifiche assegnate ad ogni nodo interno dell'albero binario fino ad arrivare alle foglie.

La variante più nota dell'algoritmo di Huffman è la *Codifica di Huffman adattiva*: la particolarità di questa variante è che genera una codifica di Huffman non conoscendo la distribuzione dei simboli della sorgente. Il vantaggio è che la codifica può essere fatta in real-time, d'altra parte l'algoritmo diventa più sensibile agli errori. Una tra le implementazioni più note di questa tecnica è stata realizzata da *Vitter*, che sfrutta una implementazione ad albero flottante.

Codici aritmetici A differenza delle codifiche tradizionali in questo caso la parola di codice rappresenta un sottointervallo aperto di $[0,1)$. Ogni sottointervallo

è rappresentato da un numero binario di precisione opportuna, tale da differenziarlo da tutti gli altri. Chiaramente, a codeword corte corrispondo sottointervalli di ampiezza maggiore, associati quindi ai simboli più probabili. In pratica l'ampiezza dei sottointervalli è ottenuta suddividendo progressivamente il segmento $[0, 1)$ in modo che le ampiezze di ogni partizione siano proporzionali alla probabilità dei singoli eventi dell'alfabeto di ingresso.
Mediamente i codificatori aritmetici raggiungono prestazioni migliori dei codificatori tradizionali. Una delle caratteristiche più interessanti è la possibilità di ottenere meno di un bit di informazione codificata per ciascun simbolo in ingresso. Il prezzo da pagare è una maggiore complessità implementativa dal momento che si devono manipolare numeri binari di precisione arbitraria.
Il processo di codifica necessita di conoscere la distribuzione di probabilità dei simboli da comprimere. Tanto maggiore è la precisione con cui è nota questa distribuzione, tanto migliore sarà la compressione ottenibile.
Esistono tre tipi di codificatori aritmetici:

- *non adattivi*. La distribuzione di probabilità è assegnata a priori, scelta opportunamente in base ai dati. Le prestazioni in genere sono molto modeste, come pure la complessità realizzativa.

- *semi-adattivi*. La codifica avviene in due stadi. Durante il primo si determina la distribuzione di probabilità mediante opportune stime. La codifica vera e propria è effettuata nel secondo stadio. Le prestazioni sono molto buone, lo svantaggio è la necessità di dover trasmettere il modello probabilistico insieme ai dati compressi, deteriorando il rapporto di compressione.

- *adattivi*. Si effettua una stima dinamica dei simboli in ingresso, apprendendo progressivamente la statistica dei dati. Il processo di compressione può avvenire, in questo caso, in un singolo stadio. Per il decodificatore sarà necessario ripetere a ritroso le stime effettuate in codifica, adattando progressivamente il proprio modello di probabilità. Lo svantaggio in questo caso consiste nel costo dell'apprendimento: all'inizio la conoscenza statistica dei dati è poco accurata, inefficace ad una adeguata compressione, durante il transitorio di adattamento iniziale, i simboli codificati non saranno ottimi dal punto di vista dei bit compressi per simbolo.

Codice di Rice La codifica di Rice sfrutta un sottoinsieme della famiglia dei codici di Golomb per produrre un codice sub-ottimo a prefisso. Rice utilizza una codifica adattiva. Considerando che i codici di Golomb hanno un parametro libero, che può essere qualunque numero maggiore di zero, i codici di Rice hanno la particolarità che quel parametro è una potenza di due. Questo rende i codici di Rice convenienti per l'uso su elaboratore, dal momento che operazioni come moltiplicazione e divisione sono svolte in modo molto efficace con l'aritmetica binaria.
I codici di Golomb utilizzano un parametro libero m, il quale serve a dividere il valore di input in due parti: q, il risultato della divisione per m, ed r, il resto. Il valore di m è scelto in modo che $p^m \approx 1/2$.
Il quoziente è codificato in modo unario: se z è il numero da codificare, la sua codifica sarà una sequenza di $n-1$ bit 1, seguiti da un bit 0 (alternativamente vale anche il complementare).
Il funzionamento della variante proposta da Rice è il seguente: consideriamo una sorgente con distribuzione di massa geometrica, cioè del tipo $p_y(n) = (1-p)p^n$, con $0 < p < 1$. L'idea è che ponendo $m = 2^k$ il quoziente $q = \lfloor n/2^k \rfloor$ si ottiene dal codice binario di n con uno shift a destra di k posizioni e il resto r mascherandone tutti i bit ad eccezione dei k meno significativi. Il codice è quindi ottenuto giustapponendo alla rappresentazione unaria di q la rappresentazione binaria su k bit di r.
Per la sua semplicità il codice di Rice è utilizzato anche per sorgenti non strettamente geometriche, ma con probabilità dei simboli velocemente decrescente.
E' interessante anche osservare che, dato un alfabeto con due simboli, 0 e 1 che hanno probabilità rispettivamente p e $(1-p)$, la codifica di Golomb può essere utilizzata per codificare corse di zeri. In questo caso è sufficiente considerare come valore del parametro m l'intero più vicino di $\frac{-1}{\log_2 p}$.
Numerosi codificatori audio lossless, come *Shorten, FLAC, Apple lossless* e *MPEG-4 ALS* utilizzano un codificatore di Rice dopo il filtro predittivo.

LZ77, LZ78 sono i nomi di due algoritmi di compressione a dizionario creati da Ziv-Lempel nei quali il dizionario non è noto a priori, ma viene costruito al codificatore e al decodificatore analizzando la sequenza dei dati che verranno compressi. Questo tipo di metodologia appartiene alla classe degli algoritmi definiti *universali*, nel senso che non necessitano di nessuna conoscenza a priori sulla sorgente e, tuttavia, assicurano una lunghezza media per simbolo che, asintoticamente, tende all'entropia della sorgente.
LZ77. Il codificatore esamina la sequenza in ingresso attraverso una finestra che

scorre lungo la sequenza da codificare. La finestra è formata da due parti distinte: una parte, detta sottofinestra di ricerca, che contiene gli ultimi caratteri della sequenza codificata ed è inizialmente vuota, la seconda parte che contiene la prossima porzione di codice da codificare, detta quindi sottofinestra di codifica.
Si cerca quindi la stringa più lunga che inizia nella sottofinestra di ricerca ed è presente anche nella sottofinestra di codifica. Una volta trovata, il codificatore sostituisce alla seconda stringa un puntatore alla prima, generando anche una tripletta di valori (p,l,C), ove p rappresenta il puntatore, l rappresenta la lunghezza della stringa e C è il codice del simbolo successivo (che serve in caso non si trovi nessuna corrispondenza). Successivamente viene aggiornata la posizione della finestra scorrevole e si ripete l'algoritmo. Tra le numerose varianti di questo algoritmo, molte propongono una ulteriore codifica, solitamente con un codice a lunghezza variabile, della tripletta di valori in uscita. Il problema più grosso che presenta questo algoritmo è che il dizionario utilizzato per i confronti è relativamente piccolo e si riferisce a un passato della stringa recente. Se consideriamo il caso peggiore, avremo che la nostra stringa di ingresso è periodica, ma di periodo superiore alla finestra temporale considerata, e quindi il nostro algoritmo sarà assolutamente inutile.
LZ78. Questo algoritmo è stato creato per risolvere il problema sopraccitato che affligge l'algoritmo LZ77. A tal scopo, la sequenza da codificare $a^N = a_1 a_2 \cdots a_N$ viene suddivisa in frasi distinte del tipo $a^N = w_1, w_2, \cdots, w_t$, dove $w_i \neq w_j$ e la virgola indica la concatenazione. L'algoritmo analizza iterativamente la sequenza di ingresso, suddividendola in frasi che non sono ancora state trovate. Alla iterazione $i \geq 1$ la parte di sequenza non ancora analizzata viene scandita, cominciando dalla lettera che segue la $(i-1)$ esima frase, e si cerca nel vocabolario V_i la frase più lunga che non sia ancora stata trovata nelle iterazioni precedenti, che indicheremo con w_i. Per costruzione, quindi, in precedenza sono stati già incontrati tutti i prefissi della frase i esima. In particolare, il prefisso costituito da tutte le lettere eccetto l'ultima, c_i, è stato trovato in una iterazione j_i precedente, quindi $w_i = w_j$. La frase w_i viene codificata come la coppia (j_i, c_i), dove l'indice j_i è codificato utilizzando $\lceil \log i \rceil$ bit e c_i, l'ultimo carattere della frase, usando $\lceil \log M \rceil$ bit, con M dimensione dell'alfabeto. La frase stessa è aggiunta al vocabolario, così ogni aggiunta è un simbolo concatenato con una voce preesistente del vocabolario stesso.
Al fine di non incontrare problemi di memoria è utile fissare la dimensione del dizionario, altrimenti può accadere che essa diverga.
Una variante famosa di questo algoritmo è LZW, proposta da Welch, che non

spedisce la coppia (j_i, c_i), ma solamente j_i.

DEFLATE L'algoritmo DEFLATE deriva dalla combinazione tra l'algoritmo LZ77 e la codifica di Huffman. Fu sviluppato da Phil Katzm noto per aver inventato PKZIP.
L'algoritmo opera su blocchi di dimensione massima 64 KB. Ogni blocco viene preceduto da un header di 3 bit:

- Bit 1
  - 0, il blocco è l'ultimo della serie
  - 1, ci sono ancora altri blocchi
- Bit 2,3
  - 00, blocco non compresso
  - 01, il blocco usa codifica di Huffman con albero prestabilito
  - 10, il blocco usa codifica di Huffman con albero proprio
  - 11, riservato.

La maggior parte dei blocchi usa una codifica di Huffman con albero proprio, in tal caso le istruzioni per costruire l'albero seguono l'header.
La compressione si divide in due stadi: il primo usa una variante dell'algoritmo LZ77 per sostituire le stringhe duplicate con i puntatori, il secondo, se necessario, codifica secondo Huffman le informazioni.

Capitolo 3

Comparativa codec lossless

Data la moltitudine di compressori lossless che si sono resi disponibili nell'ultimo decennio, non è molto facile scegliere quello più adatto alle proprie esigenze. E' infatti un errore abbastanza comune scegliere un codec solamente in base alle performance sulla compressione, ignorando molte altre caratteristiche che i codec offrono.
Per esempio l'esigenza di un codec compatibile con più piattaforme ci può portare a scegliere *FLAC*, oppure se le nostre esigenze riguardano solamente il rapporto di compressione potremmo essere portati a scegliere *OptimFROG*.
E' per questo motivo che sono stati comparati diversi codec con le relative opzioni, i risultati sono riportati in Tabella 3.1:

Caratteristiche	**FLAC**	**OptimFROG**	**ALAC**	**TTA**	**MPEG4-ALS**	**MPEG4-SLS**
Velocità encoding	alta	bassa	media	alta	media	bassa
Velocità decoding	alta	media	alta	alta	alta	bassa
Rapp. compressione	medio	alto	basso	medio	alto	medio
Supporto software	ottimo	medio	scarso	medio	scarso	scarso
Codifica ibrida	no	si	no	no	no	si
Multichannel	si	no	si	si	si	si
Supporto OS	tutti	Win-Mac-Linux	Win-Mac	tutti	tutti	tutti
Categoria	open souce	registrato	registrato	open source	standard	standard

Tabella 3.1: Caratteristiche principali di alcuni codec lossless. La versione completa è disponibile sul wiki di Hydrogen audio.

Nel proseguo della tesi esamineremo in dettaglio due codec molto diversi come principio di funzionamento, *MPEG4-ALS* che utilizza predittori lineari e *MPEG4-SLS* che utilizza invece le trasformate.
La scelta è ricaduta sui tali codec perché essi sono standardizzati, gli schemi di funzionamento sono visionabili senza restrizioni ed inoltre rappresentano le due diverse metodologie di approcio alla codifica lossless.

3.1 Standard MPEG

Nel luglio del 2002, MPEG [1] istituì una commissione per valutare nuove tecnologie per la compressione audio lossless. Nel dicembre 2002 furono visionate le proposte di numerosi istituti di ricerca. Tali proposte utilizzavano due approci differenti: codifica con predittori e codifica con trasformate. MPEG decise quindi di supportare entrambi gli approci, scegliendo per ognuno un modello di riferimento. Nel 2006 furono pubblicati due standard internazionali: *ISO/IEC 14496-3:2005/ Amd 2:2006 Audio Lossless Coding (ALS)* e *ISO/IEC 14496-3:2005/Amd 3:2006 Scalable Lossless Coding (SLS)*.
Lo standard SLS utilizza la trasformata IntMDCT, mentre lo standard ALS utilizza una cascata di predittori lineari.

[1] Moving Picture Experts Group, è un comitato tecnico congiunto delle organizzazioni ISO e IEC incaricato di definire standard per la rappresentazione in forma digitale di audio, video e altre tipologie di contenuti multimediali in modo da soddisfare un'ampia varietà di applicazioni.

Capitolo 4

MPEG-4 ALS

In questo capitolo sarà approfondito il funzionamento del codec MPEG4-ALS, ove ALS è l'acronimo di Audio Lossless. L'analisi sarà concentrata su tutte le fasi della codifica, come riferimento si utilizzerà la Figura 4.1 che illustra lo schema a blocchi di principio del codec.
Brevemente: il segnale audio in ingresso è partizionato in blocchi, per ciascun blocco è calcolato il residuo, utilizzando predizioni a breve e lungo termine. Tale residuo è quindi codificato entropicamente prima di essere multiplexato insieme agli altri dati e quindi spedito.

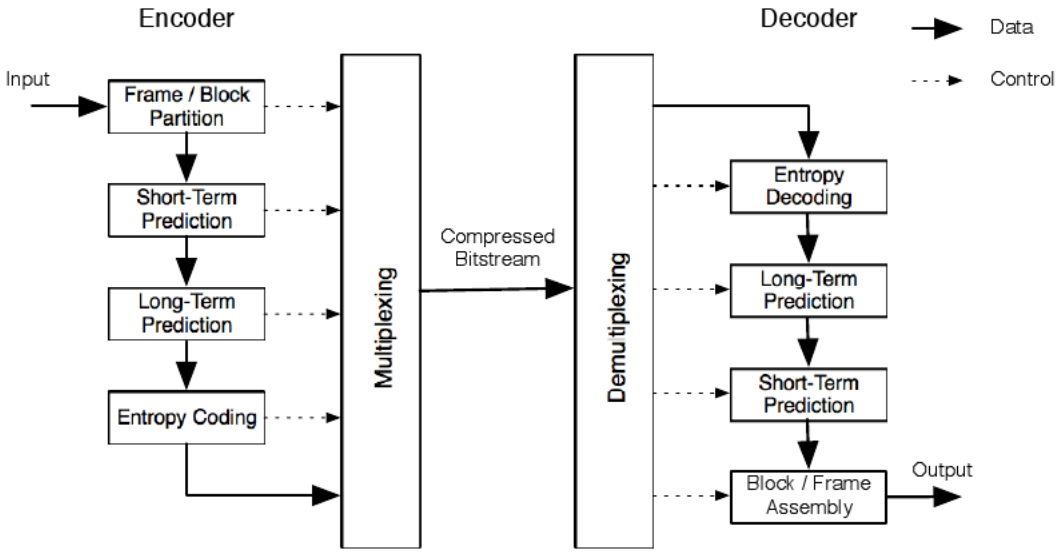


Fig. 4.1: Diagramma a blocchi dell'encoder e del decoder ALS.

4.1 Blocking

Il codec MPEG4-ALS offre la possibilità di scegliere se utilizzare dei blocchi di lunghezza fissa oppure dei blocchi di lunghezza variabile. Se viene scelta la seconda opzione ogni blocco di lunghezza N può essere gerarchicamente suddiviso in un massimo di 32 sotto blocchi. E' possibile creare qualunque combinazione di blocchi che abbiano dimensione: $N/2$, $N/4$, $N/8$, $N/16$ e $N/32$.
La scelta della più appropriata suddivisione dei blocchi è interamente gestita dall'encoder, e la sua implementazione non è specificata nello standard.

4.2 Accesso casuale

L'accesso casuale, vedi Figura 4.2, è un'opzione che consente di accedere a qualunque parte del segnale codificato senza dover decodificare tutta la parte precedente.
Per consentire questa operazione sono inseriti dei frame specifici che riproducono esattamente il segnale di ingresso. Prima che il segnale codificato sia inviato però è necessario rimuovere la correlazione dal segnale contenuto nel frame ad accesso casuale. Ovviamente la predizione su tali blocchi è molto difficile da eseguire con metodi standard, dal momento che per avere un filtro di ordine K dovremmo avere a disposizione K campioni dal frame precedente, il quale però è già codificato. Per aggirare tale inconveniente si utilizza una tecnica chiamata *predizione progressiva* la quale associa un filtro del primo ordine al secondo campione, uno del secondo al terzo e cosi via.

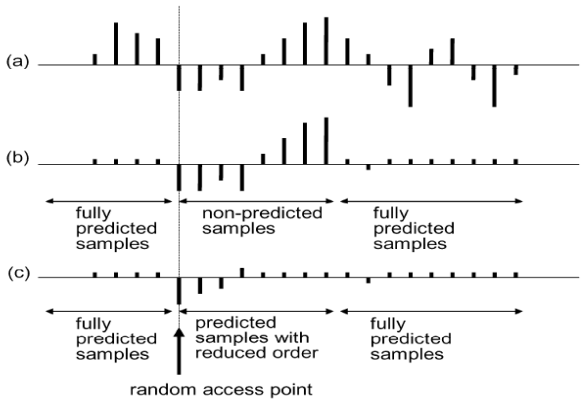


Fig. 4.2: Funzionamento blocchi ad accesso casuale.

4.3 Predizione

MPEG4-ALS utilizza sia la predizione a breve termine che la predizione a lungo termine. La prima sfrutta la correlazione tra campioni vicini, mentre la seconda è utilizzata per determinare la correlazione di campioni distanti fra loro nel tempo. MPEG4-ALS utilizza tre tipi di predittori: *LPC*, *RLS-LMS* e *LTP*, ove i primi due appartengono alla categoria dei predittori a breve termine, mentre il terzo è un predittore a lungo termine.

4.3.1 Predizione LPC

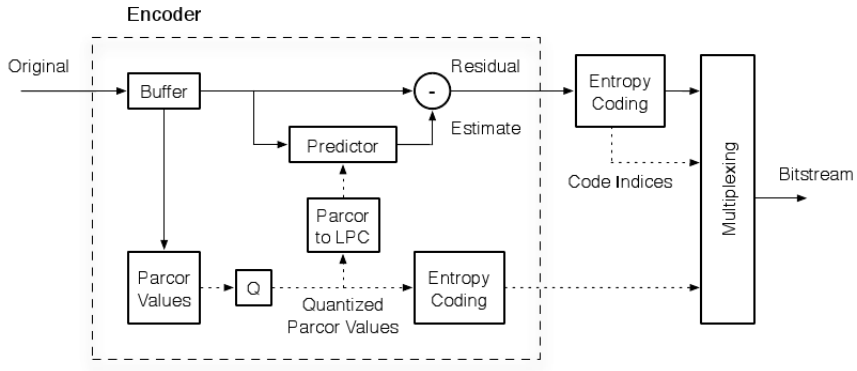


Fig. 4.3: Predittore LPC (encoder).

Il predittore LPC è mostrato in Figura 4.3
Per ogni blocco in ingresso è calcolato un insieme di coefficienti parcor (partial correlation) usando l'algoritmo di *Levinson-Durbin*. L'insieme dei coefficienti è ottimo, infatti il segnale residuo ha varianza minima. Tali coefficienti sono quindi quantizzati e convertiti in coefficienti LPC, i quali sono utilizzati dal predittore per generare il segnale residuo. La quantizzazione dei coefficienti parcor si rende necessaria dal momento che tali coefficienti vengono trasmessi. La decodifica esegue il procedimento inverso.

4.3.2 RLS-LMS

Il predittore RLS-LMS, vedi Figura 4.4 è di tipo adattivo, il suo utilizzo nel codec è opzionale perché migliora di molto il rapporto di compressione, ma allo

stesso tempo comporta un notevole incremento della complessità computazionale per la codifica del segnale.

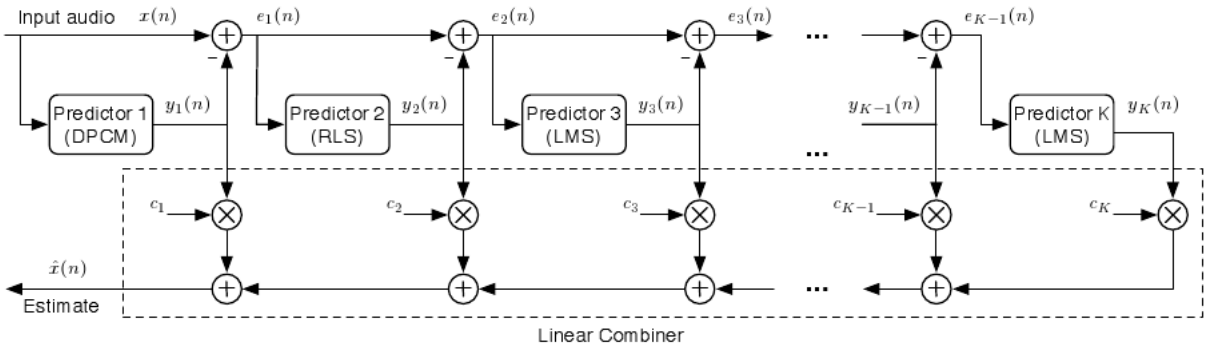


Fig. 4.4: Struttura del predittore RLS-LMS.

Il predittore consiste in una cascata di stadi di predizione così composta: predittore DPCM (Differential Pulse Code Modulation), predittore RLS (Recursive Least Square) e una serie di predittori LMS (Least Mean Square). La particolarità risiede nel fatto che il residuo di ogni stadio diventa l'ingresso dello stadio successivo, ottenendo un risultato sempre più preciso. Le stime prodotte da ogni stadio sono sommate da un *combinatore lineare* al fine di generare una stima complessiva del segnale di ingresso.
Analizziamo ora tutte le parti del predittore RLS-LMS.

DPCM Il primo stadio della predizione è il predittore DPCM, che è un semplice predittore del primo ordine, quindi la predizione è data da:

$$y_1(n) = x(n-1)$$

$$e_1(n) = x(n) - y_1(n)$$

La predizione DPCM rimuove la componente continua dal segnale audio, che se non fosse rimossa causerebbe problemi al seguente stadio di codifica adattiva.

Predittore RLS E' il secondo predittore della cascata. Lo scopo di questo blocco è di adattare i coefficienti del predittore. L'algoritmo è inizializzato settando la matrice di auto-correlazione inversa $\vec{P}$ come segue:

$$\vec{P}(0) = \delta\vec{I}$$

dove δ è un numero positivo sufficientemente piccolo e $\vec{I}$ è una matrice identità di dimensioni $M_2 \times M_2$ e l'ordine del predittore è appunto M_2.
Il vettore dei coefficienti è $\vec{a}_2(n)$ ed è definito come

$$\vec{a}_2(n) = [a_{2,1}(n), a_{2,2}(n), \ldots, a_{2,M_2}(n)]^T$$

ed è inizializzato ponendo $\vec{a}_2(0) = \vec{0}$. Definendo inoltre il vettore di input al predittore RLS come

$$\vec{e}_1(n) = [e_1(n-1), e_1(n-2)\ldots., e_1(n-M_2)]^T$$

è possibile stabilire per ogni istante $n = 1, 2, \ldots$ le equazioni che descrivono il sistema:

$$\begin{aligned}
\vec{v}(n) &= \vec{P}(n-1)\vec{e}_1(n) && (4.1)\\
m &= \frac{1}{\vec{e}_1(n)^T\vec{v}(n)} \quad \text{se} \quad \vec{e}_1(n)^T\vec{v}(n) \neq 0, \quad 1 \quad \text{altrimenti} && (4.2)\\
\vec{k}(n) &= m\vec{v}(n) && (4.3)\\
y_2(n) &= \vec{a}_2(n-1)^T\vec{e}_1(n) && (4.4)\\
e_2(n) &= e_1(n) - y_2(n) && (4.5)\\
\vec{a}_2(n) &= \vec{a}_2(n-1) + \vec{k}(n)e_2(n) && (4.6)\\
\vec{P}(n) &= \text{Tri}\{\lambda^{-1}(\vec{P}(n-1) - \vec{k}(n)\vec{v}^T(n)).\} && (4.7)
\end{aligned}$$

Il fattore λ è un numero reale positivo e minore di 1, mentre il simbolo *Tri* indica che l'operazione verrà prima eseguita sulla matrice triangolare inferiore, e seguentemente i valori di tale parte saranno copiati sulla matrice triangolare superiore, di modo che sia verificata la condizione $p_{i,j} = p_{j,i}$.
Il predittore RLS è usato per rimuovere le componenti ad alte frequenze, quindi anche il rumore, perché ha una velocità di inseguimento del segnale in ingresso molto elevata.

Predittore LMS Gli stadi LMS del predittore sfruttano l'algoritmo *NLMS* (Normalized Least Mean Square) per adattare i coefficienti. Per studiare il funzionamento del predittore poniamoci nello stadio k-esimo; il vettore dei coefficienti è

$$\vec{a}_k(n) = [a_{k,1}(n), a_{k,2}(n), \ldots, a_{k,M_k}(n)]^T$$

che è inizializzato ponendo $\vec{a}_k(0) = \vec{0}$, mentre M_k indica l'ordine del predittore. Definendo inoltre

$$\vec{e}_{k-1}(n) = [e_{k-1}(n-1), e_{k-1}(n-2), \ldots, e_{k-1}(n-M_k)]^T$$

come il vettore di input del k-esimo stadio, per ogni istante $n = 1, 2, \ldots$ è possibile stabilire le equazioni che descrivono il sistema:

$$\begin{aligned} y_k(n) &= \vec{a}_k(n-1)^T \vec{e}_{k-1}(n) && (4.8)\\ e_k(n) &= e_{k-1}(n) - y_k(n) && (4.9)\\ \vec{a}_k(n) &= \vec{a}_k(n-1) + \frac{e_k(n)\vec{e}_{k-1}(n)}{1 + \mu_k \vec{e}_{k-1}(n)^T \vec{e}_{k-1}(n)} && (4.10) \end{aligned}$$

dove μ_k è il passo dell'algoritmo NLMS. Dal momento che la componente continua e le componenti ad alte frequenze sono state rimosse dai blocchi precedenti, il predittore LMS avrà il compito di rimuovere le componenti a bassa frequenza.

Combinatore lineare Il combinatore lineare (nel seguito CL) moltiplica e somma l'insieme dei coefficienti risultanti dalle stime dei blocchi: DPCM, RLS, LMS. Per adattare i coefficienti del CL viene usato l'algoritmo *Sign-Sign LMS*. Definiamo il vettore dei coefficiente come:

$$\vec{c}(n) = [c_1(n), c_2(n), \ldots, c_k(n)]^T$$

dove k è il numero di stadi di predizione utilizzati nella cascata. Il vettore di input è dato da:

$$\vec{y}(n) = [y_1(n), y_2(n), \ldots, y_k(n)]^T$$

Allora la stima del predittore RLS-LMS è calcolata come

$$\hat{x}(n) = \vec{c}(n)^T \vec{y}(n)$$

mentre i coefficienti del combinatore lineare sono aggiornati secondo la seguente regola:

$$\vec{c}(n+1) = \vec{c}(n) + \alpha \cdot \mathrm{sgn}[\vec{y}(n)] \cdot \mathrm{sgn}[x(n) - \hat{x}(n)]$$

ove $\mathrm{sgn}(t)$ è la funzione segno e α è il passo dell'algoritmo.

4.3.3 Predizione a lungo termine (LTP)

E' noto che la maggior parte dei segnali audio ha armoniche o componenti periodiche che sono originate dalle frequenze fondamentali degli strumenti musicali. Per rimuovere tali componenti è impensabile usare dei filtri standard, dal momento che sarebbe necessario un ordine molto elevato. E' per questo motivo che alla predizione a breve periodo è addizionata una seconda predizione prima che il segnale giunga all'encoder. In Figura 4.5 è illustrata brevemente l'idea che sta alla base della predizione. Riportiamo per completezza anche la formula per il calcolo dell'errore:

$$\hat{e}(n) = e(n) - (\sum_{j=-2}^{2} \gamma_{\tau+j} e(n-\tau+j))$$

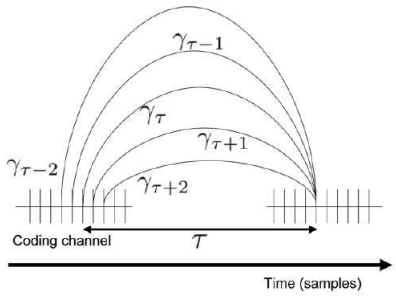


Fig. 4.5: Funzionamento predittore lungo termine.

I simboli $\hat{e}(n)$ e $e(n)$ indicano il residuo dopo e prima del predittore LTP, τ indica il ritardo del campione e γ il guadagno della quantizzazione.
L'uso di questa opzione è sconsigliato, dal momento che è molto difficile trovare le periodicità di un segnale audio.

4.4 Codifica Entropica

Nello standard MPEG4-ALS la predizione lineare applicata al segnale di ingresso genera un segnale residuo la cui dinamica è molto ridotta rispetto al segnale originale. La distribuzione di tale dinamica può essere modellata con sufficiente approssimazione con una distribuzione Laplaciana.
Per codificare il segnale residuo il metodo più semplice è quello di utilizzare la codifica di Rice.

Alternativamente il residuo può essere codificato con il più complesso ed efficiente codice di *Gilbert-Moore*, chiamato BGMC. Nel BGMC, la distribuzione del residuo è suddivisa in tre parti: una regione centrale limitata da due regioni di coda. Il residuo delle regioni di coda viene ri-centrato e codificato tramite Rice. Il residuo della regione centrale è ulteriormente diviso in: parte del segnale che appartiene ai bit meno significativi (LSB) e parte del segnale che appartiene ai bit più significativi(MSB). La parte LSB viene direttamente trasmessa utilizzando codici a precisione finita, mentre la parte MSB è codificata con il codice aritmetico di Gilbert e Moore.

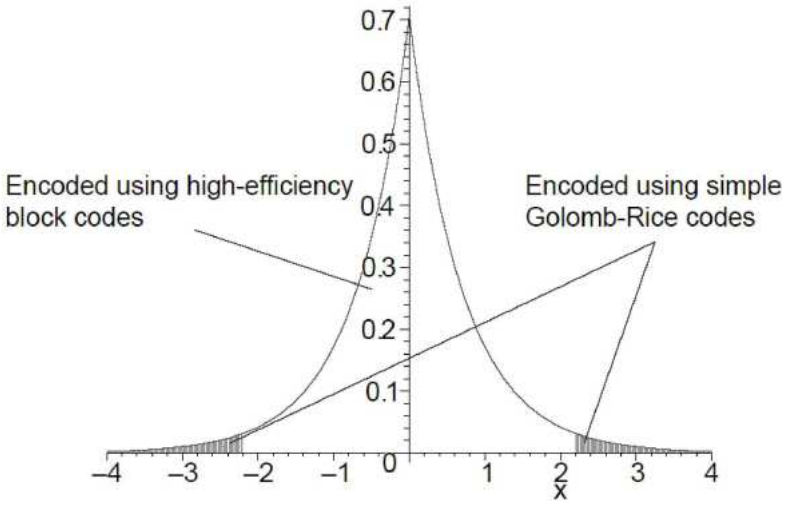


Fig. 4.6: Divisione della distribuzione del residuo.

Capitolo 5

MPEG4 - SLS

MPEG4-SLS, acronimo di Scalable Lossless Coding, è un codec che sfrutta le trasformate per ottenere una compressione audio lossless e utilizza una codifica ibrida lossy-lossless. Come si evince dalla Figura 5.1, i campioni PCM audio in ingresso sono trasformati attraverso un algoritmo chiamato *Integer Modified Discrete Cosine Transform* (in breve IntMDCT) in coefficienti IntMCDT. Questi coefficienti sono passati ad una struttura a due stadi costituita da uno stadio chiamato *core layer* e uno stadio di miglioramento *Lossless Enhancement* LLE.
Il core layer utilizza un coder audio lossy MPEG4-AAC (advanced audio coding) per generare un flusso dati di tipo lossy. Nel coder AAC i coefficienti IntMCDT sono quantizzati seguendo modelli psico acustici, in modo che il rumore percepito dall'apparato uditivo umano sia il minimo possibile. Tali coefficienti sono codificati entropicamente per generare il flusso di bit lossy.
Lo stadio LLE sottrae ai campioni IntMCDT il flusso dati lossy proveniente dal coder AAC tramite un processo chiamato *error mapping*. In seguito il segnale residuo è quindi codificato entropicamente. La funzione di error mapping è anche quella di mantenere la distribuzione di probabilità dei coefficienti IntMCDT residui il più possibile uguale a quella dei coefficienti IntMCDT originali; infatti questi ultimi hanno distribuzione Laplaciana che ben si presta alla codifica entropica.
MPEG4-SLS offre la possibilità di non utilizzare il coder AAC se si desidera ottenere solamente il segnale lossless.

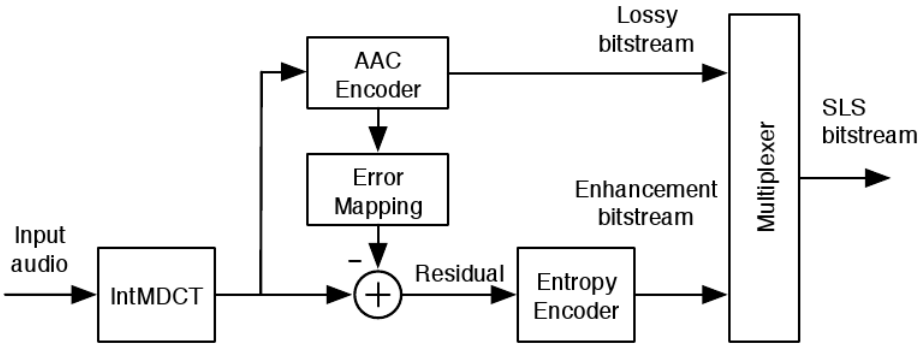


Fig. 5.1: Diagramma a blocchi dell'encoder SLS.

5.1 IntMCDT

Come noto i codec audio che utilizzano le trasformate operano nel dominio della frequenza. Per eseguire questa operazione MPEG4-SLS utilizza una particolare trasformata chiamata MCDT. La scelta è dovuta al fatto che essa offre una buona capacità di compattare l'energia del segnale e che gode della proprietà di *critical sampling*, che garantisce che non siano generati in output più simboli di quelli che sono presenti all'input. IntMCDT è stata introdotta come trasformata reversibile tra interi, i quali approssimano come meglio possibili i coefficienti di MCDT, ed eredita da MCDT tutte le proprietà. IntMCDT è derivato da MDCT,

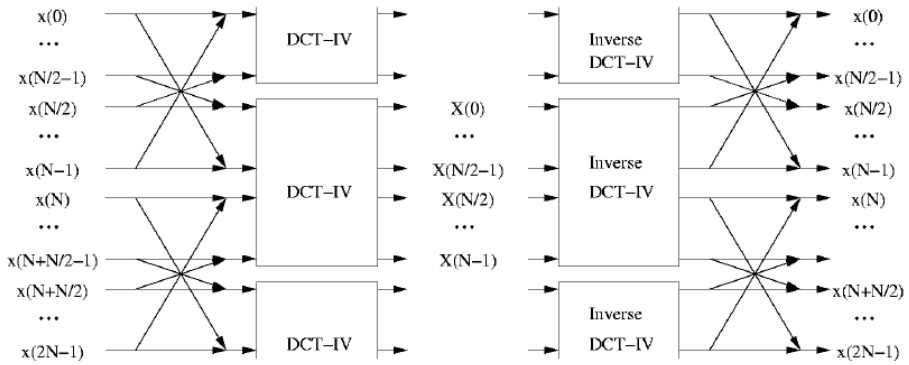


Fig. 5.2: MCDT e MCDT inverso con finestramento e DCT-IV.

il quale può essere scomposto in un'operazione di finestramento e in una DCT-IV (Discrete Cosine Transform type-IV) come mostrato in Figura 5.2.

5.1.1 Finestramento

L'operazione di finestramento (time domain aliasing) consiste in un gruppo di *rotazioni di Givens*. In IntMCDT ogni rotazione di Givens è implementata usando tre passaggi di lifting:

$$\begin{pmatrix} x(k) \\ x(n-1-k)-1 \end{pmatrix} \mapsto$$

$$\begin{pmatrix} 1 & -\frac{w(N-1-k)-1}{w(k)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -w(k) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{w(N-1-k)-1}{w(k)} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x(k) \\ x(n-1-k)-1 \end{pmatrix}$$

dove $k = 0, \ldots, \frac{N}{2} - 1$ e w(k) sono i coefficienti della finestra MCDT. Dopo ogni passo di lifting è operata un'operazione di arrotondamento per assicurare che i valori rimangano nel dominio dei numeri interi.

5.1.2 DCT-IV

Per ottenere la trasformazione IntMCDT, la parte DCT-IV è calcolata tramite un modello reversibile chiamato Integer DCT-IV. Per ottenere questo modello il coder SLS utilizza lo schema MDL (multidimensional lifting), che permette di ottenere il minor numero possibile di operazioni di arrotondamento.
La seguente scomposizione della matrice invertibile T mostra il principio dello schema MDL (I indica la matrice identità):

$$\begin{pmatrix} T & 0 \\ 0 & T^{-1} \end{pmatrix} = \begin{pmatrix} -I & 0 \\ T^{-1} & I \end{pmatrix} \begin{pmatrix} I & -T \\ 0 & I \end{pmatrix} \begin{pmatrix} 0 & I \\ I & T^{-1} \end{pmatrix}$$

I tre blocchi di questa scomposizione sono appunto i passi del Multi-dimensional lifting. In modo simile queste operazioni possono essere trasferite all'operazione di mappatura reversibile intera arrotondando i valori frazionari dopo la moltiplicazione tra matrici e scambiando i segni delle operazioni di somma. Queste operazioni sono riportate in Figura 5.3. Applicando lo schema MDL a DCT-IV si ottiene la seguente scomposizione:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \begin{pmatrix} C_N^{IV} & 0 \\ 0 & C_N^{IV} \end{pmatrix} =$$

$$\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} I & \frac{C_N^{IV}}{\sqrt{2}} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ -\sqrt{2}C_N^{IV} & I \end{pmatrix} \begin{pmatrix} I & \frac{C_N^{IV}}{\sqrt{2}} \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -\frac{1}{2}I \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ I & I \end{pmatrix}$$

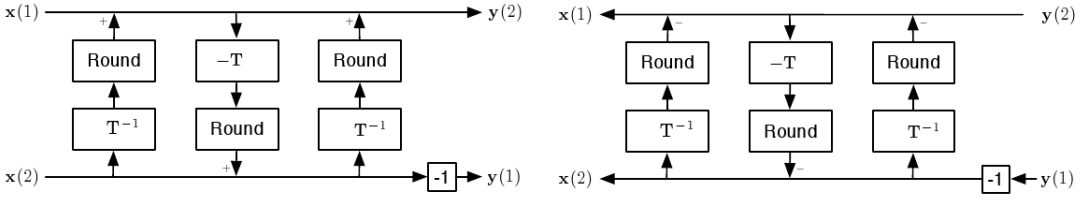


Fig. 5.3: Matrici di lifting: trasferimento della mappatura intero-intero (sopra) e mappatura inversa (sotto).

Con poche modifiche la struttura di figura 5.3 è adatta sia a segnali stereo che a segnali mono.

5.1.3 Modellazione del rumore

A causa delle operazioni di arrotondamento, IntMCDT introduce un rumore di approssimazione allo spettro del segnale audio. Questa approssimazione è udibile per alcuni segnali audio oppure alle alte frequenze, dove generalmente il segnale audio ha poca energia. Per risolvere questo inconveniente è utilizzata una tecnica di modellazione del rumore. Dal momento che generalmente i segnali audio hanno molta più energia alle basse frequenze che alla alte, si cerca di portare il rumore nella zona a basse frequenze, mascherando quindi il rumore con il segnale audio. In questo modo il rumore diventa non udibile. Per eseguire questa operazione viene utilizzato un filtro passa-basso del primo ordine, Figura 5.4.

5.2 Error Mapping

Il processo di error mapping ricostruisce la porzione dello spettro IntMCDT che è già stata codificata dal coder AAC. L'encoder AAC può essere visto come un processo di quantizzazione e codifica dove i valori spettrali dell'IntMCDT sono prima raggruppati in differenti scale di valori di banda (nel seguito abbreviato in sbf), successivamente quantizzati con differenti passi e quindi codificati entropicamente per produrre il bit stream AAC.
Sia $i[k]$ il coefficiente IntMCDT quantizzato di $c[k]$ per i valori di scala s, allora il

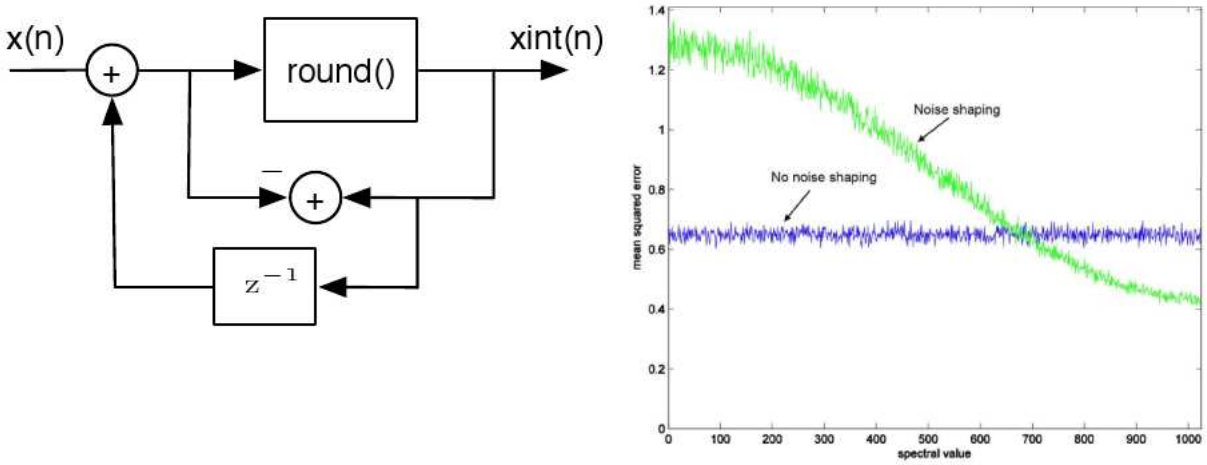


Fig. 5.4: Filtro per modellazione rumore ed errore quadratico medio con (verde) e senza (blu) filtraggio sul rumore.

valore spettrale ricostruito sarà:

$$ric(i[k]) = \begin{cases} \text{sgn}(i[k]) \left[\sqrt[4]{2^{f_s(s)}} (|i[k]| - C)^{4/3} \right], & i(k) \neq 0 \\ 0, \qquad i[k] = 0 \end{cases}$$

ove $f_s(s)$ è il fattore di scala che determina il passo della quantizzazione a seconda del gruppo di valori spettrali, e $C = 0.4054$ è un offset di arrotondamento fissato.
Il processo di error mapping calcola quindi il residuo dell'IntMCDT nel seguente modo:

$$e[k] = \begin{cases} c[k] - \lfloor ric(i[k]) \rfloor, & i[k] \neq 0 \\ c[k], & i[k] = 0 \end{cases}$$

I vantaggi derivanti dall'uso di questo error mapping sono sostanzialmente due. Il primo è che se $c[k]$ è sufficientemente grande, quindi $i[k] \neq 0$, allora il segno del residuo di IntMCDT $e[k]$ sarà identico a quello di $i[k]$ e quindi non dovrà essere codificato. Il secondo è illustrato in Figura 5.5, dalla quale notiamo che se $c[k]$ è un vettore aleatorio laplaciano, allora $e[k]$ potrà essere approssimata da una distribuzione Laplaciana unilatera.

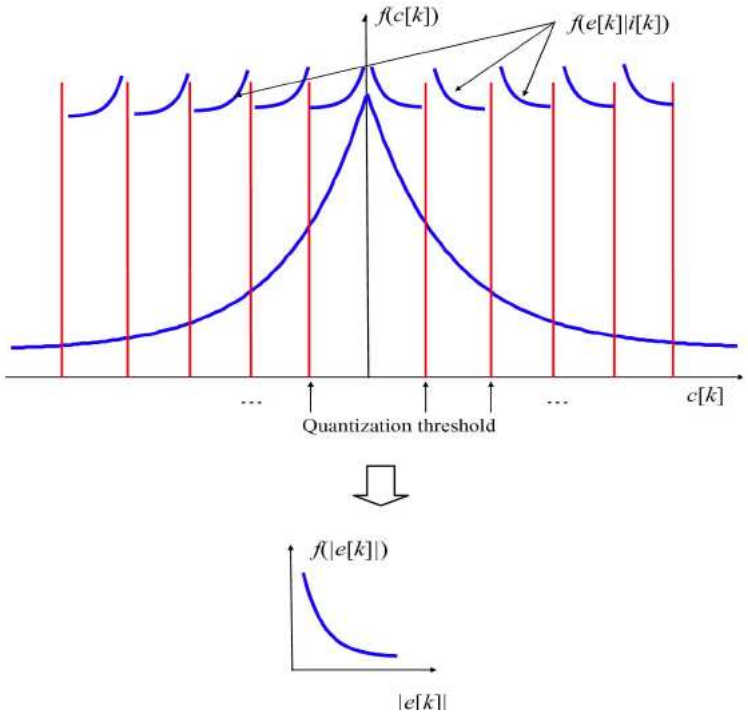


Fig. 5.5: Sopra: pdf dei coefficienti IntMCDT distribuiti secondo una Laplaciana e la pdf condizionata del residuo di IntMCDT. Sotto: la pdf incondizionata delle ampiezze dei residui di IntMCDT.

5.3 Codifica Entropica

5.3.1 Codifica Bit-Plane

La tecnologia di codifica bit-plane (letteralmente piani di bit) è usata dal codec SLS al fine di produrre un flusso dati lossless scalabile. Consideriamo un vettore residuo in ingresso del tipo $\vec{e} = \{e[0], \ldots, e[N-1]\}$, dove N indica la dimensione del vettore. Nella codifica bit-plane ogni elemento $e[k]$ è rappresentato in forma binaria come

$$e[k] = (2s[k]-1)\sum_{j=0}^{M-1} b[k,j]2^j \qquad k = 0, \ldots .N-1.$$

che comprende un simbolo per il segno generato da $s[k]$ in base alla regola:

$$s[k] = \begin{cases} 1 & e[k] \geq 0 \\ 0 & e[k] < 0 \end{cases}$$

e un simbolo bit-plane $b[k, j] \in \{0,1\}$.
M rappresenta il bit più significativo (MSB) per $\vec{e}$ che soddisfa
$2^{M-1} \leq max\{|e[k]|\} < 2^M, k = 0, \ldots, N-1$. I simboli dei vari bit-plane sono quindi scannerizzati e codificati partendo dal MSB ed arrivando al LSB per tutti gli elementi di $\vec{e}$, in modo da produrre uno stream di dati compresso. In Figura 5.6 è rappresentato l'ordinamento dei bit per ogni sfb.

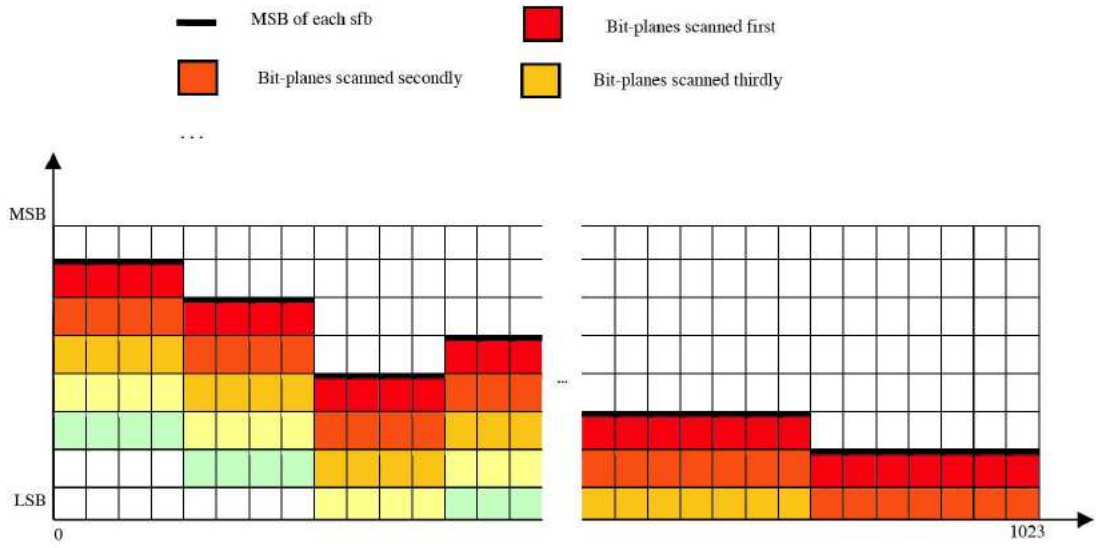


Fig. 5.6: Codifica Bit-Plane.

BPGC *Bit Plane Golomb Code* è un codice aritmetico utilizzato per la codifica entropica del segnale residuo. Il funzionamento è relativamente semplice, i simboli dei vari bit-plane vengono codificati attraverso un codice aritmetico che utilizza una tabella di frequenze fissate. Le frequenze sono assegnate in base ad una regola che deriva dalle proprietà statistiche di una sorgente con distribuzione geometrica. Per questo motivo la codifica BPGC ottiene eccellenti risultati con sorgenti a distribuzione geometrica.
Nella codifica BPGC un simbolo di un bit-plane al piano j-esimo è codificato con probabilità data da:

$$Q(j) = \begin{cases} \dfrac{1}{1+2^{j-lazybp}} & j \geq lazybp \\ \dfrac{1}{2} & j < lazybp \end{cases}$$

dove il parametro *lazybp* è scelto in base alla regola:

$$lazybp = min\{L^{'} \in Z \mid 2^{L^{'}+1}N \geq A\}$$

con N ed A indicanti rispettivamente la lunghezza e la somma in valore assoluto dei dati contenuti nel vettore codificato.

CBAC Per incrementare le prestazioni è stata successivamente introdotta una tecnica di codifica entropica più sofisticata, chiamata *Context-Based Arithmetic Code*. L'idea di CBAC è ispirata dal fatto che la distribuzione di probabilità dei simboli dei bit-plane è generalmente correlata alla posizione in frequenza e ai valori delle linee spettrali adiacenti ad essa. Per esplorare questo tipo di correlazione sono introdotti tre concetti di *contesto*:

- *Frequency Band context*: la distribuzione di probabilità di un simbolo bit-plane varia a seconda della banda di frequenza in cui ci troviamo. Si possono classificare tre tipi di bande:

  1. Bassa: 0 kHz $\sim$ 4 kHz, FB = 0.
  2. Media: 4 kHz $\sim$ 11 kHz, FB = 1.
  3. Alta: superiore a 11 kHz, FB = 2.

- *Distance to lazy*: è definita come la distanza tra il bit-plane corrente e il parametro lazy-bp della codifica BPGC (in breve D2L).

  $$D2L = \begin{cases} 3 - j + lazybp & se \quad j - lazybp \geq -2 \\ 6 & altrimenti \end{cases}$$

  La logica alla base di questo contesto è basata sul fatto che l'alterazione della distribuzione di probabilità tra il simbolo dei bit-plane e la sorgente, che ha distribuzione simile alla Laplaciana, tende a diminuire come il numero $D2L$. Per ridurre il numero totale, tutti i $D2L \geq 6$ sono raggruppati in un solo gruppo con probabilità 0.5, dal momento che l'alterazione della probabilità in questi casi è cosi piccola che un ulteriore codifica aritmetica comporterebbe un guadagno irrisorio nella codifica.

- *Significant State*: lo scopo di questo contesto è quello di evidenziare la correlazione tra l'ampiezza dei dati IntMCDT correnti, le linee spettrali ad essi adiacenti e l'intervallo di quantizzazione del core AAC.

5.3.2 Codifica a bassa energia

Sia BPGC che CBAC sono processi di codifica entropica che ottengono risultati ottimi solamente nel caso in cui la sorgente abbia distribuzione Laplaciana o simile ad essa.
Esistono comunque zone T/F (tempo/frequenza) che sono silenziose, oppure che hanno poca energia come le sbf ad alte frequenze, per cui i valori spettrali IntMCDT sono dominati dagli errori di arrotondamento accumulati in tutti i passi dell'algoritmo. La distribuzione di questi IntMCDT è molto diversa dalla distribuzione Laplaciana. E' per questo motivo che è stata introdotta la codifica a bassa energia. La codifica a bassa energia è utilizzata per tutti quelle sbf tali che il parametro lazy-bp di BPGC sia minore o uguale a zero. L'ampiezza dei residui è prima convertita in modo unario (vedi capitolo 2) e successivamente codificata tramite codifica aritmetica facendo uso dei parametri *pos* che indica la posizione del simbolo, e quindi la sua forma unaria, e *lazy-bp*.

Capitolo 6

Analisi sulle prestazioni di MPEG4-ALS

In questa parte ci proponiamo di analizzare come le diversi opzioni di cui dispone il codec preso in esame nel Capitolo 4 influenzino significativamente il risultato della compressione. Per le simulazioni è stato utilizzato l'encoder di riferimento disponibili in rete. Per i test sono state utilizzate diverse tracce audio, che differiscono per genere musicale o per composizione. Essendo infatti la codifica lossless basata sulla rimozione della ridondanza i risultati ottenibili variano a seconda del genere musicale che si vuole codificare.

6.1 MPEG4-ALS

I test effettuati mirano ad evidenziare la relazione che intercorre tra il risultato finale e l'uso di opzioni di compressione sempre più incisive. In particolare è stato analizzato come siano decisivi, per ottenere risultati ottimi, la predizione adattiva e l'algoritmo RLSLMS.
Le specifiche opzioni utilizzate in ogni test sono riportate in seguito.

Tipo di test

1. *Test 1*
    - no predizione adattiva, no predizione a lungo termine
    - canali indipendenti, ordine di predizione : 2
    - no RLS-LMS
2. *Test2*
    - predizione adattiva abilitata, predizione a lungo termine abilitata
    - canali indipendenti, ordine di predizione : 5
    - no RLS-LMS
3. *Test 3*
    - predizione adattiva abilitata, predizione a lungo termine abilitata
    - canali dipendenti, ordine di predizione : 5
    - no RLS-LMS
4. *Test 4*
    - predizione adattiva abilitata, predizione a lungo termine abilitata
    - canali dipendenti, ordine di predizione : 5
    - RLS-LMS abilitato, uso dell'opzione best
5. *Test 5*
    - predizione adattiva abilitata, predizione a lungo termine abilitata
    - canali dipendenti, ordine di predizione : 5
    - RLS-LMS abilitato, uso dell'opzione quick
6. *Test 6*
    - predizione adattiva abilitata, predizione a lungo termine abilitata
    - canali dipendenti, ordine di predizione : 10
    - no RLS-LMS

Traccia	Originale (MB)	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
1	49,54	48,52	38,7	38,25	37,17	37,26	37,98
2	31,3	26,29	15,8	15,48	14,9	15,04	15,61
3	33,49	31,41	23,61	22,87	21,9	22,1	22,8
4	33,56	24,18	8,66	8,13	7,7	8,03	8,44
5	18,57	16,76	10,23	9,98	9,85	9,86	10,05
6	84,58	75,94	50,53	50,22	45,1	45,92	50,45
7	6,43	5,5	3,31	3,07	3,07	3,07	3,16
8	29,84	24,57	13,82	13,41	12,48	12,95	13,69

Tabella 6.1: Tabella riassuntiva delle dimensione dei file pre e post codifica.

Traccia	Titolo	Autore	Genere
1	Bad romance	Lady Gaga	Pop
2	El condor pasa	Simon - Garfunkel	Musica andina
3	2+2 = 5	Radiohead	Indie-rock
4	Jesu Joy	J.S.Bach	Piano
5	Monostatic	Ritchie Hawtin	Elettronica
6	Red House	Jimi Hendrix	Blues
7	Panpot Spliff	Ritchie Hawtin	Minimal
8	Toccata e fuga	J.S.Bach	Classica

Tabella 6.2: Tabella riassuntiva delle tracce utilizzate nei test.

Come riportato a pagina seguente, in Figura 6.1 sono stati prodotti 4 grafici, che mirano ad evidenziare come l'uso di determinate opzioni sia determinante per ottenere un buon risultato.
Il grafico 1 evidenzia gli indubbi vantaggi che si possono trarre dall'uso della predizione adattiva e dall'aumento dell'ordine del predittore. In alcuni casi la dimensione del file prodotto è circa la metà rispetto a quanto riscontrato nel test 1, questo perché l'uso di una predizione non adattiva non consente di ottenere risultati ottimi, come già evidenziato nel Capitolo 2.
Il grafico 2 evidenzia la differenza presente nella dimensione finale scegliendo l'opzione best o quick in RLS-LMS. Ciò che è interessante notare è che anche senza utilizzare l'algoritmo RLS-LMS ponendo l'ordine del predittore a 10, si riesca ad arrivare a risultati in alcuni casi molto buoni, a scapito della complessità computazionale e della velocità di codifica.
Per concludere, il grafico 4 dimostra come il sistema di codifica inter-channel utilizzato da MPEG4-ALS riesca a migliorare di qualche punto percentuale il rapporto finale di compressione.

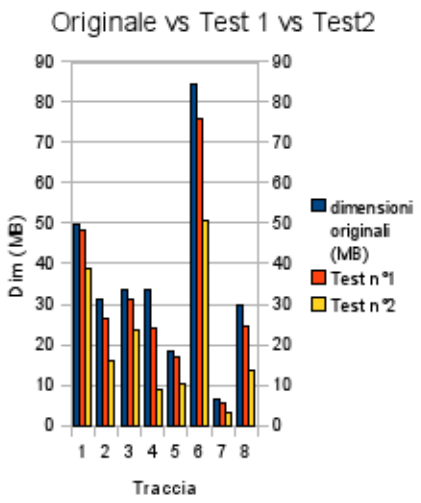


Grafico 1

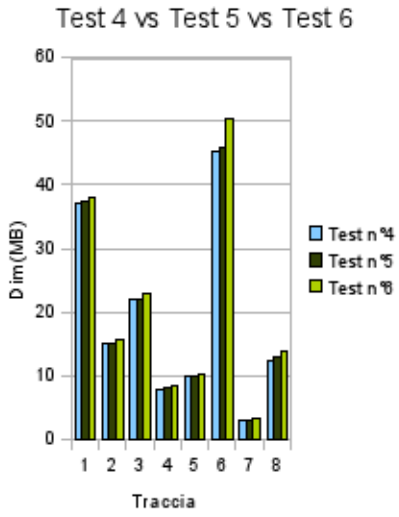


Grafico 2

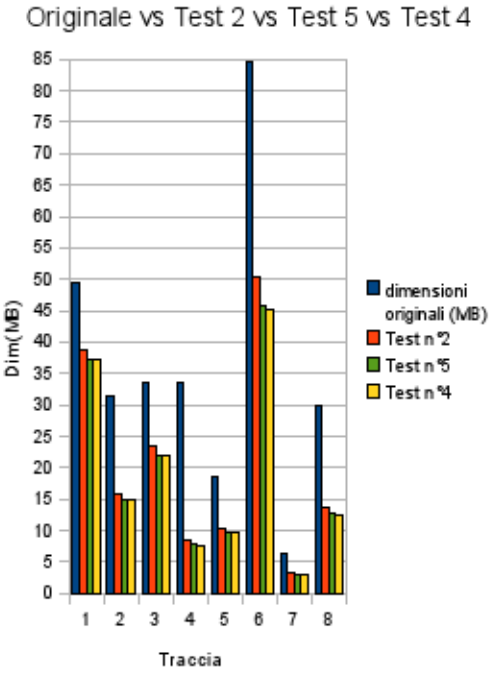


Grafico 3

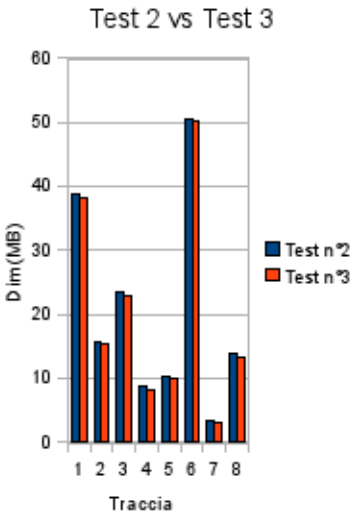


Grafico 4

Fig. 6.1: Grafici 1,2,3,4 relativi ai test svolti su MPEG4-ALS

Capitolo 7

Parte sperimentale

Considerando l'ammontare considerevole dei software di codifica lossless già esistenti, il presente capitolo non si propone di introdurre qualche innovazione su metodi o algoritmi di codifica, ma vuole solamente essere un banco di prova per applicare alcune delle fondamentali conoscenze apprese durante la stesura dei capitoli precedenti.
E' stato scelto di implementare un codificatore che sfrutta un particolare tipo di predizione, chiamata MEQP, perché si presenta come una predizione non troppo pesante dal punto di vista computazionale e adattiva, quindi sicuramente consona ad ottenere dei buoni risultati.
Di seguito viene riportata una descrizione sull'implementazione, quindi un'analisi approfondita di ogni singolo passo di codifica e quindi verrano riportati i risultati ottenuti dalle simulazioni.

7.1 Descrizione del progetto

In Figura 7.1 viene riportato lo schema a blocchi sul quale si base il funzionamento del software. Il segnale originale viene pre-elaborato, in questo modo si acquisiscono le conoscenze sulla sua struttura, come frequenza di campionamento o numero di bit, successivamente il segnale viene diviso in blocchi, e per ogni blocco si calcola la inter-channel correlation. A questo punto, disponendo di un segnale pronto per essere elaborato, si procede con l'algoritmo MEQP, che produce in output una vettore contenete i coefficienti della predizione e l'errore. Quest'ultimo viene codificato tramite il metodo di Golomb-Rice, e quindi viene multiplexato

insieme al vettore dei coefficienti prima di essere salvato in un file esterno. Il procedimento viene reiterato fino al termine del segnale originale in ingresso.

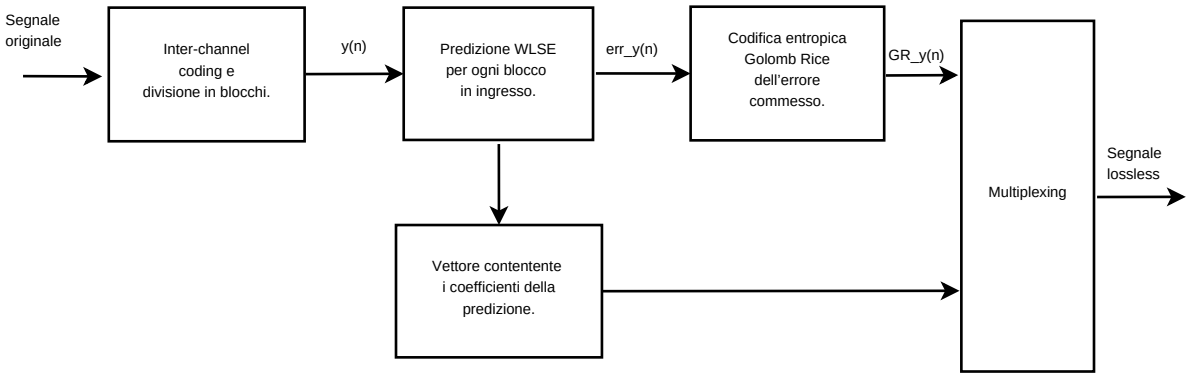


Fig. 7.1: Schema a blocchi del software sviluppato in Matlab.

Nel seguito saranno analizzati approfonditamente i singoli passi di codifica.

7.1.1 Divisione in blocchi

E' stato scelto di utilizzare una divisione in blocchi di dimensione prefissata. Il motivo di tale scelta è, oltre alla facilità di implementazione, la natura predittiva dell'algoritmo di predizione. Tale caratteristica consente infatti di ottenere risultati buoni indipendentemente dall'uniformità del segnale in ingresso. Per determinare la dimensione ottimale dei blocchi sono state effettuate diverse simulazione, e si è visto che impostando la dimensione a 256, i risultati ottenuti sono stati sensibilmente migliori rispetto a quelli che si sarebbero potuti ottenere operando scelte diverse.

7.1.2 Inter-channel coding

Generalmente i file wave si presentano in formato stereo, quindi dotati di canale *left* e *right*. Come scritto nel Capitolo 2 è conveniente ai fini del risultato finale tenere in considerazione la correlazione che vi può essere tra i due canali. In questo caso è stato utilizzato il metodo *Mid-Side coding* che utilizza per la codifica il segnale $mid = left + right$ e $side = left - right$.

7.1.3 Algoritmo MEQP

La predizione MEQP [1], acronimo che sta per *minimo errore quadratico pesato*, cerca di minimizzare la somma pesata degli errori rispetto ad un determinato insieme di pesi. L'algoritmo è di carattere adattivo, infatti per ogni istante $t = k$ è determinato un nuovo insieme di coefficienti, e questi sono utilizzati per la predizione all'istante $t = k+1$. Dal momento che i campioni presenti sono influenzati solamente da un limitato insieme dei campioni passati non è necessario tenere traccia di tutta l'evoluzione passata del segnale. Per questo motivo è necessario che i pesi dei campioni passati possano essere elaborati in modo da tenere conto di questa particolarità. A tal proposito viene introdotta una variabile $\alpha_k = \alpha^{k-1}$ ove $\alpha < 1$ ed è chiamata *fattore di memoria*. Generalmente $\alpha = 0.99$, questo consente di ottenere un buon compromesso tra l'utilizzo dei campioni passati e l'ottimizzazione del risultato finale.
L'algoritmo procede come segue:

1. Calcolo l'output della predizione corrente: $\hat{x}(k) = \vec{a}_M(k-1)\vec{u}(k)$

2. Aggiorno il vettore dei coefficienti

$$\vec{a}_M(k) = \vec{a}_M(k-1) + \frac{\vec{P}(k-1)\vec{u}(k)}{\alpha + \vec{u}^T(k)\vec{P}(k-1)\vec{u}(k)}\{x(k) - \hat{x}(k)\}$$

3. Aggiorno la matrice $\vec{P}$

$$\vec{P}(k) = \frac{1}{\alpha}\left(\vec{P}(k-1) - \frac{\vec{P}(k-1)\vec{u}(k)\vec{u}^T(k)\vec{P}(k-1)}{\alpha + \vec{u}^T(k)\vec{P}(k-1)\vec{u}(k)}\right)$$

In Figura 7.2 è riportato un esempio tratto da una simulazione specifica in Matlab. La simulazione è stata effettuata su un blocco da 512 campioni di una traccia audio. In rosso è riportata l'ampiezza del campione della predizione, in colore blu l'ampiezza del segnale originale, mentre in verde l'errore, ottenuto come differenza tra i segnali precedenti. Si noti il buon comportamento dell'algoritmo MEQP, che per la simulazione in questione utilizza un filtro di ordine 2. Come si evince dall Figura 7.2 l'ampiezza dell'errore si attesta in range di valori compreso tra -5 e +5, questo consente di ottenere in fase di codifica un cospicuo risparmio di bit.

[1] Maggiori informazioni sono reperibili sul testo: Introduction to Adaptive Arrays, John Wiley Sons, 1980.

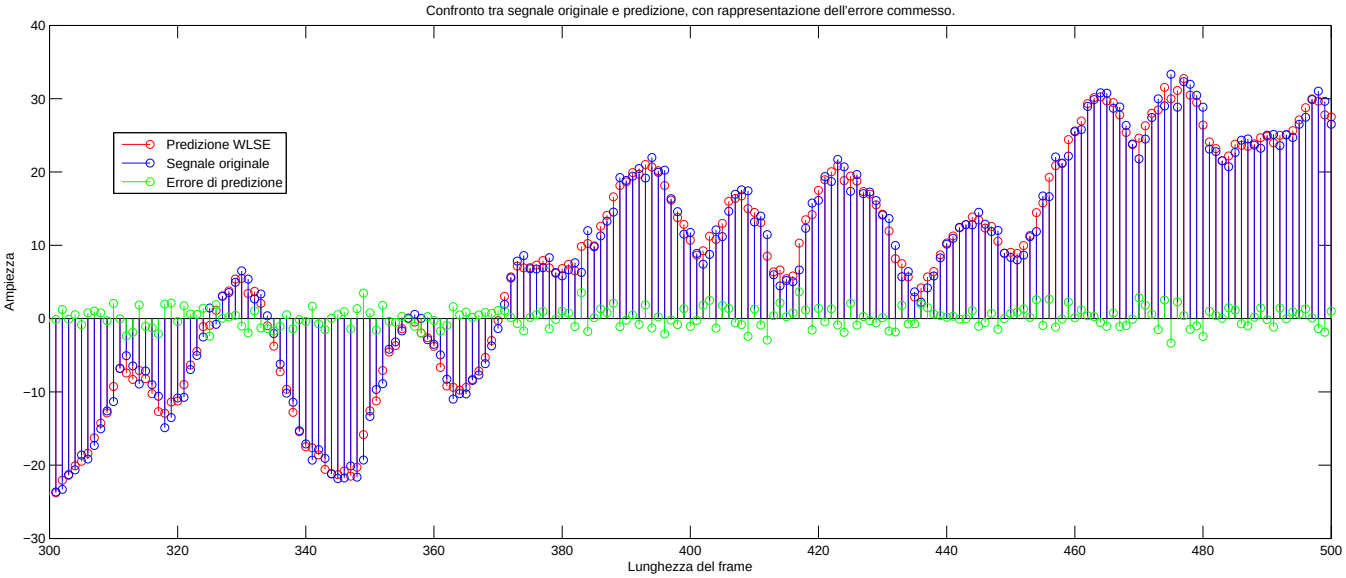


Fig. 7.2: Segnale originale - Segnale predizione - errore

7.1.4 Codifica entropica

L'errore determinato come differenza tra il segnale originale è codificato entropicamente utilizzando il metodo di *Golomb-Rice*. Per ogni blocco di campioni errore è calcolato un parametro *p*, che determina la lunghezza ottimale del codice contenente i bit meno significativi. A tale codice viene giustapposta la codifica unaria dei bit più significativi e quindi un bit di segno. Il risultato viene salvato su un vettore che in seguito sarà esportato in forma binaria su un file di uscita.

7.1.5 Multiplexing

Per non perdere informazioni fondamentali per la corretta decodifica, al termine dell'elaborazione di ogni blocco di campioni viene salvato il vettore dei coefficienti dell'algoritmo MEQP, il valore del parametro *p*, e la codifica entropica dell'errore.

7.2 Simulazioni

Per le simulazioni e la stesura del codice è stato utilizzato l'ambiente Matlab. Tale scelta non ha consentito di svolgere simulazioni sulla velocità dell'encoder, data l'inefficienza per questo tipo di operazioni di Matlab.
Sono stati testati quattro brani audio, appartenenti a generi musicali diversi, le cui specifiche sono riportate in Tabella 7.1. Data la diversità delle tracce audio in ingresso ci si aspetta di trovare rapporti di compressione eterogenei.

Traccia	Titolo	Autore	Genere
1	Monostatic	Ritchie Hawtin	Elettronica
2	Suite No 2 in Bm	J.S.Bach	Musica Classica
3	Some unholy war	Amy Winehouse	Pop
4	Fire	Jimi Hendrix	Rock Blues

Tabella 7.1: Tabella riassuntiva delle tracce utilizzate nei test.

Ogni file audio è stato compresso singolarmente e successivamente è stata trascritta la dimensione del file, i risultati ottenuti sono riportati nella Tabella 7.2:

Traccia	Dimensione Originale (MB)	Dimensione compressa (MB)	Rapporto compressione
1	19.47	10.2	52.39
2	15.07	7.33	48.64
3	25.17	15.54	61.74
4	33.91	21.36	62.99

Tabella 7.2: Tabella risultati simulazione.

Il rapporto di compressione è un parametro che definisce la bontà dell'encoder in termini solamente riguardanti le dimensioni finali, e non la velocità di compressione. Viene definito come:

$$rapporto.compressione = \frac{file.compresso}{file.originale} * 100$$

pertanto più il suo valore è basso e più la compressione è stata efficace.
In Figura 7.3 è illustrato un istogramma contenente i risultati conseguiti, che come previsto sopra sono eterogenei a seconda del genere musicale della traccia in ingresso.
D'ora in avanti nei grafici nomineremo il software implementato **TLC**, ossia Tesi Lossless Codec.

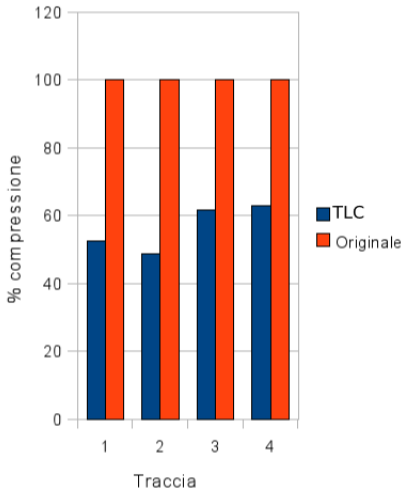


Fig. 7.3: Rapporto compressione software implementato.

Comparativa

Per verificare l'effettiva bontà dei risultati ottenuti è stata stilata una comparativa tra alcuni encoder e l'encoder implementato. Le tracce audio utilizzate sono le stesse riportate in Tabella 7.1. Inoltre tutti gli altri encoder sono stati settati in modo da ottenere performance di medio livello, ed ognuno di essi utilizza la predizione lineare come base per la compressione. Riportiamo in Tabella 7.3 e in Figura 7.4 i risultati ottenuti.

Traccia	Originale (MB)	TLC	Apple Lossless	FLAC	OptimFrog	Monkey's audio	TTA
1	19.47	10.2	10.29	10.41	9.79	9.88	10.13
2	15.07	7.33	6.9	6.79	6.28	6.48	6.76
3	25.17	15.25	16.09	15.76	14.9	15.02	15.56
4	33.91	21.36	21.45	21.24	20.2	20.42	21.15

Tabella 7.3: Tabella comparativa compressione usando diversi encoder.

Come espresso dalla figura l'encoder TLC riesce ad ottenere un buon livello di compressione per qualunque file in ingresso. I risultati ottenuti sono molto positivi, pur considerando che gli altri encoder non sono stati utilizzati con le opzioni che consentono la miglior codifica
Per completezza è stato aggiunta anche la Tabella 7.4, con relativo grafico, Figura 7.5, che mostra il rapporto medio di compressione ottenuto dai vari encoder per tutte e quattro le tracce in ingresso.

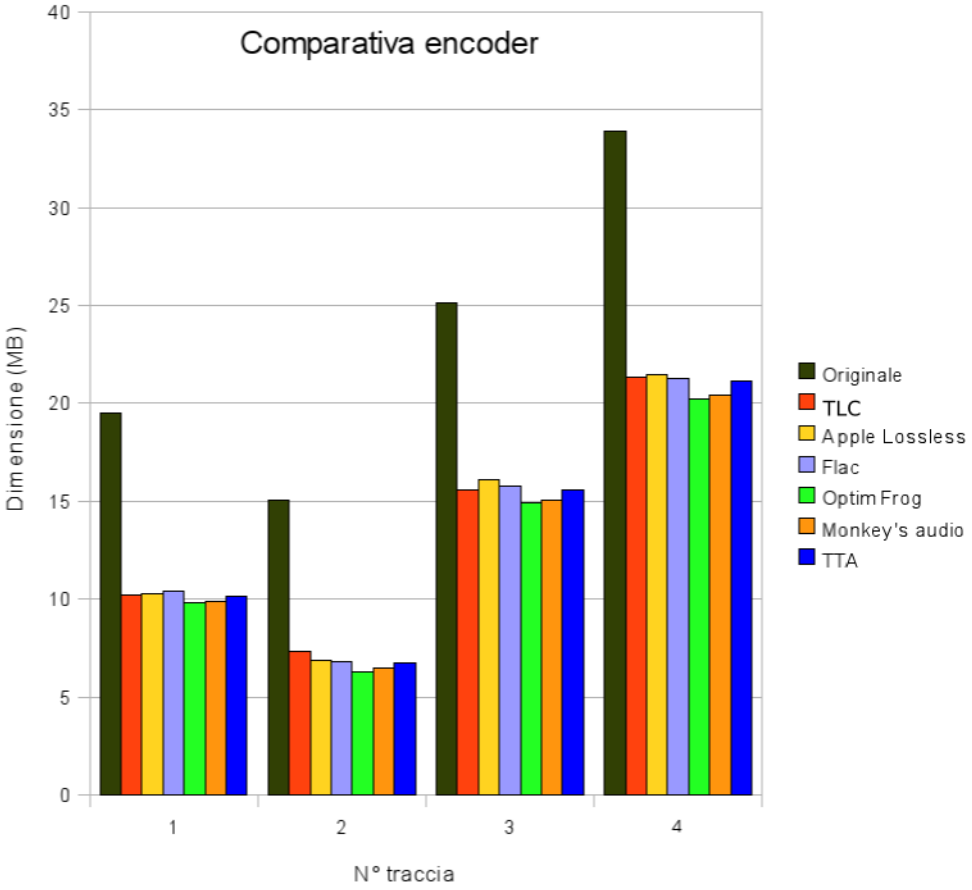


Fig. 7.4: Comparativa compressione tracce audio.

Traccia	TLC	Apple Lossless	FLAC	OptimFrog	Monkey's audio	TTA
1	52.39	52.82	53.47	50.28	50.74	52
2	48.64	45.79	45.06	41.67	43	44.86
3	61.74	63.93	62.61	59.2	59.67	61.82
4	62.99	63.26	62.24	59.57	60.22	62.36
MEDIA	56.44	56.45	55.94	52.68	53.41	55.26

Tabella 7.4: Tabella compressione media.

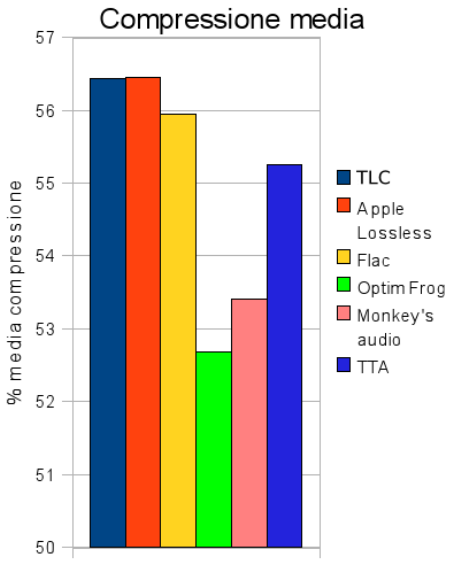


Fig. 7.5: Compressione media dei diversi encoder.

Quantitativamente i risultati ottenuti indicano una compressione di buon livello, in particolare questo risultato è stato ottenuto grazie all'algoritmo di predizione MEQP.
Qualitativamente non è stato possibile effettuare dei test che tengano traccia ad esempio della velocità di codifica o degli eventuali problemi su sistemi operativi diversi, pertanto i risultati ottenuti, come del resto già preventivato, hanno la sola finalità di confermare praticamente i risultati teorici di cui al Capitolo 2.

Capitolo 8

Conclusioni

In questa tesi si è esplorato il mondo della codifica lossless per segnali audio, descrivendone nei primi capitoli gli aspetti di base e cercando di approfondire nei capitoli successivi come operano le due grandi famiglie di compressori, cioè la famiglia dei compressori che sfruttano la predizione e la famiglia dei compressori che utilizzano le trasformate. A tal proposito sono stati descritti approfonditamente sia lo standard MPEG4-ALS che lo standard MPEG4-SLS.
Successivamente parte della tesi è stata dedicata alla sperimentazione e alla simulazioni di diversi encoder, tra i quali uno è stato ideato sulle basi di quanto appreso nel Capitolo 2 e quindi implementato in Matlab.
I test svolti sull'encoder MPEG4-ALS avevano la finalità di dimostrare praticamente come l'uso della codifica inter channel e della predizione RLS-LMS consentano effettivamente di ottenere risultati migliori. I risultati ottenuti hanno infatti dimostrato l'effettiva utilità della codifica iter-channel e i miglioramenti derivanti dall'uso della predizione RLS-LMS su un ampio insieme di brani musicali.
Per quanto concerne lo sviluppo di un encoder, la motivazione che mi ha spinto alla implementazione di tale progetto è stata quella di voler testare praticamente quanto appreso, durante la stesura della tesi, dal punto di vista teorico. L'encoder in questione cerca infatti di riassumere in se buona parte delle fondamentali componenti atte ad ottenere risultati soddisfacenti, che sono state trattate nel Capitolo 2. Esso si basa sulla predizione lineare, questa scelta è stata fatta perché è meno complessa dal punto di vista matematico e implementativo. Come riportato nel Capitolo 7 i risultati ottenuti sono stati buoni.
Personalmente ritengo che, soprattutto in prospettiva di applicazioni per internet, il futuro della codifica lossless appartenga ai compressori con trasformate, perché

essi garantiscono l'indubbio vantaggio di poter disporre sia di una traccia lossy, quindi di dimensioni molto ridotte, sia di una traccia lossless contemporaneamente. A tal proposito penso che sarebbe interessante in futuro cercare di implementare un compressore di tale tipo.