Dispense di Sergio Cicero; anno accademico 2021/22

Disclaimer: il seguente documento contiene degli appunti rielaborati, pertanto questo non sostituisce né il testo consigliato dal docente né tantomeno il materiale didattico fornito.

LA COMPRESSIONE

Questa operazione è fondamentale per ridurre le dimensioni in memoria delle audio, quindi ciò ne accresce anche la velocità di trasmissione e quindi aumenta l'economicità dell'hardware, che necessita di meno spazio per trattare gli audio.
I problemi però non mancano, infatti servono appositi dispositivi per eseguire le operazioni di compressione che accrescono la complessità dell'hardware del dispositivo.
In un'operazione di compressione si dovrebbe fissare una minima soglia al di sotto della quale le ampiezze vengano scartate, in questa maniera si è certi di eliminare rumori (ampiezza molto bassa) e silenzi (ampiezza nulla).
È già stata vista la relazione matematica per calcolare lo spazio che occupa in memoria una traccia audio e il bitrate al quale verrebbe trasmessa.
Per effettuare una compressione, quindi diminuire le grandezze del file audio, si deve effettuare una riquantizzazione del segnale. Esistono infatti 2 algoritmi di codifica (A-law e μ-law) che consentono di eseguire una quantizzazione simile.

Le due codifiche sopra citate sono state pensate per codificare e trasmettere la voce al telefono (suono che va da 0 a 4000Hz). Per ottimizzare la banda sulla quale esse operano, sono state ideate a partire da una quantizzazione non uniforme di tipo logaritmico (infatti come è stato più visto durante questo corso, il logaritmo produce scale la cui ampiezza è scostante) a 8 bit.

codifica μ-law

Questa codifica consente di utilizzare solo 8 bit ottenendo la stessa qualità di una codifica che invece ne utilizza 14. La quantizzazione a 14 bit è non-uniforme; la codifica μ-law impiega un numero maggiore di bit per i valori di ampiezza prossimi allo 0 dell'origine del grafico di quantizzazione, questo si fa per evitare di non avere informazioni sufficiente a digitalizzare i segnali di basse intensità (si è già affrontato questo argomento).
Questa codifica sfrutta una funzione che prende in input una quantizzazione e dà in output un'altra quantizzazione rimodificata secondo la logica μ-law.

Funzione di riquantizzazione μ-law:

$$Y = \begin{cases} 128 + \dfrac{127}{ln(1+\mu)} \times ln(1+\mu|x|) & x \geq 0 \\ 127 - \dfrac{127}{ln(1+\mu)} \times ln(1+\mu|x|) & x < 0 \end{cases}$$

Y: è una funzione che prende in input un quanto della vecchia codifica e restituisce in output il quanto nel quale cadrà la forma d'onda nella nuova codifica

La funzione Y si può graficare, essa è una funzione limitata fra 0 e 255 il cui andamento è uguale a quello dell'arcotangente, solo che a differenza di questa funzione goniometrica, la funzione Y non esplode a +infinito, ma assume valori per le x in [-32768 +32767].

Le x sono i valori della forma d'onda del segnale che prima cadevano in quasi diversi e che ora vengono raccolti in quanti comuni.
esempio: le x in [-32768; -32100], nella nuova quantizzazione cadono tutte all'interno del quanto numero 0 -> più di 600 valori di x sono codificati in un unico quanto!

Ovviamente, la funzione Y è invertibile, in questa maniera si possono ri-ottenere le x, ossia i valori della vecchia quantizzazione, questo passo è fondamentale per fare capire al dispositivo che riceverà i dati come ricostruire la funzione.

Come è deducibile, la ri-quantizzazione introduce delle perdite, pertanto la codifica μ-law è lossy.

codifica A-law
Anche questa logica di ri-quantizzazione sfrutta una legge non lineare; essa sfrutta 8 bit per ottenere una quantizzazione la cui qualità è pari a quella di una che ne usa 13.
Scelti:
x -> il valore di ampiezza del segnale normalizzato, tale che x vari fra 1 e -1
A -> una costante pari a 87.7 opp 87.6
Y -> valore della nuova codifica
Il valore Y, normalizzato in [-1; 1] si ottiene dalla legge:

$$Y = \text{sign}(X) \begin{cases} \frac{A|X|}{1+\ln A} & |X| < \frac{1}{A} \\ \frac{1+\ln A|X|}{1+\ln A} & \frac{1}{A} < |X| \leq 1 \end{cases}$$

Come per la codifica μ-law, anche questa legge è invertibile per le stesse identiche considerazioni.

codifica PCM
Già affrontata nella digitalizzazione.

Altre 2 codifiche sono la DCPM (Differential Pulse Code Modulation) e la ADPCM (Adaptive DCPM)

DCPM
È una PCM pensata per essere loseless, questa tecnica anziché concentrarsi sui valori di ampiezza, si focalizza sui valori di differenza fra un'ampiezza e l'altra, questa tecnica è appunto detta di differencing.

Differencing: produce l'informazione riguardante la differenza di ampiezza della funzione fra 2 quanti successivi

La differencing può essere utilizzata a comando, quando la si vuole utilizzare si attiva un bit che ne impone il suo utilizzato, altrimenti lo si spegne, in questa maniera si può anche usare la semplice PCM (il che conviene per operazioni molto semplici, essendo questa tecnica più elementare della DPCM).

La tecnica di differencing viene implementata combinando le tecniche RLE (Run Length Encording) e LUT (Look-up table).

RLE: utilizza delle coppie che dicono in che quantità si ripete un valore
esempio:
ho 3 ripetizione di +1 -> (3, +1)
ho 2 ripetizioni di 0 -> (2, 0)
ho 2 ripetizioni di -1 -> (2, -1)

LUT -> logiche deve essere presente sia nel ricevente che nel trasmettitore, altrimenti non funziona
La LUT fornisce degli indici che associa alle coppie del RLE

esempio:
alla coppia (3, +1) associo l'indice 1, se il destinatario riceve 1 capisce che essa corrisponde alla coppia (3, +1).

Questa corrispondenza coppia-indice è mantenuta in delle tabelle, ecco perché la LUT deve essere implementata su entrambi i dispositivi.
Significato dei bit settati a +1, 0, -1:
+1 -> il segnale sonoro sta crescendo
0 -> il segnale sonoro è costante
-1 -> il segnale sonoro sta decrescendo

I valori di differenze sono piccoli, più piccoli delle parole logiche impiegate dalla PCM normale, essi infatti possono essere "-1, 0, +1".
La DPCM è utile solo in situazioni in cui le parole logiche dei dati trasportati sono assai complesse, se nel sistema arrivano parole logiche abbastanza semplici (ossia rappresentate su pochi bit) è inutile introdurre l'operazione di differencing!

Ricapitolando: la RLE prevede la trasmissione della differenza di ampiezza, accompagnata dal numero di ripetizioni consecutive nella quale essa si presenta; con il DPCM che sfrutta le LUT basta inviare un solo indice (pochi bit), poi la tabella recupererà anche la sequenza di valori di differenze consecutive. Quanto appena detto spiega perché quando le operazioni di trasferimento non sono elementari, la PCM non è la migliore strada da percorrere dato essa, in queste situazioni, è più probabile che introduca un maggiore numero di errori avendo a che fare con una mole più corpulenta di dati.

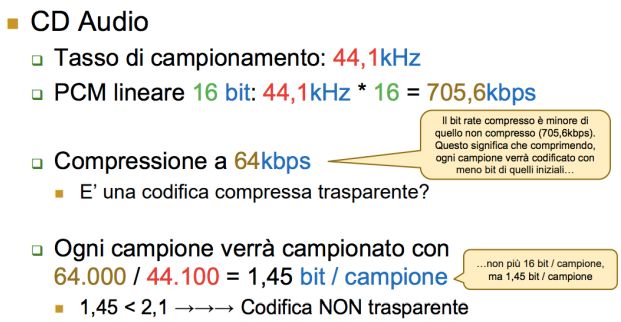
ADPCM
Tecnica di codifica più sofisticata della DPCM, oltre a lavorare come lavora la DPCM, essa implementa anche un algoritmo di predizione e di riquantizzazione. Questi algoritmi vengono utilizzati per stimare quali siano i prossimi valori che riceverà; quando poi arriveranno i valori reali essi verranno confrontati con quelli che erano stati predetti e aggiustati in seguito ad un confronto fra i due.
Per predire il campione successivo si somma "±1" al valore di differenza ricevuto per ultimo.

fattori di ri-quantizazione
Definite tutte le possibili tecniche di ri-quantizzazione, si osservino ora i possibili fattori utilizzabili per comprimere le codifiche basate sul modello PCM:
IMA ADPCM e ACE/MACE ADPCM sono 2 diversi metodi di implementare la PCM che forniscono due tecniche distinte per eseguire una compressione.
IMA (Interactive Multimedia Association): compressione di 4 bit a 1 -> si ottengono in output il 25% dei bit rispetto il file non ancora compresso
ACE/MACE: compressione di 2 bit a 1 -> si ottengono in output il 50% dei bit rispetto il file non ancora compresso

Queste 2 tecniche dal punto di vista matematico/informatico sono lossy, ossia è sicuro che smarriscano dati nel percorso "file -> file compresso -> file"; tuttavia a livello percettivo non è lossy perché la qualità del suono resta invariata all'ascolto.

James D. Johnston è colui che si occupò di definire un limite all'operazione di comprensione affinché essa non inquini il suono a livello percettivo.
Una codifica, ossia una compressione, si dice **trasparente** quando un ascoltatore non si rende conto che il suono ha subito una compressione, ossia quando a livello percettivo il suono prima e dopo la compressione resta lo stesso.
Una codifica non è più trasparente quando si scende al di sotto del rapporto di "2,1" bit per campione.

Esempio:
La proprietà di trasparenza è stato visto dipendere dal bitrate supportato dall'hardware, ciò significa che uno strumento che attua una compressione non può essere progettato affinché l'operazione che esso svolgerà sarà o meno trasparente, sarà il contesto in cui verrà inserito a



determinare se la sua compressione sarà o meno trasparente (quindi se si vuole ottenere una compressione trasparente bisogno regolare il bitrate affinché essa lo sia).

La compressione di tipo percettivo
I compressori amplificano i rumori di fondo (essendo questi al di sotto della soglia di compressione).
Gli espansori invece fungono da operatore dinamico per attenuare i suoni di alte intensità.
Combinando il risultato dell'espansore, con quello del compressore, si è in grado di amplificare suoni i quali di norma non sarebbero accessibili al compressore poiché al di sopra della sua soglia di azione.
Questa tecnica è detta **compansion**, essa è appunto una compressione dinamica del suono:
espansione -> compressione -> de-compressione -> de-espansione

la compansion viene utilizzata in fase di registrazione del suono al fine di creare un netto distacco fra suoni di fondo e suoni principali. La compansion è una compressione di tipo percettivo perché agisce sulla percezione sonora finale del suono in uscita.

μ-law, A-law, ADPCM sono anch'esse codifiche di tipo percettivo.
Si introducano adesso altre 4 codifiche di questo tipo:
codifica a blocchi (block coding); transform coding; sub-band coding; Huffman coding.

codifica a blocchi
È una quantizzazione non uniforme a virgola mobile, quindi è necessario definire fra i dati che essa andrà ad utilizzare: un esponente, una mantissa e il segno.
Questa codifica impiega le potenze del 10 e rappresenta l'intensità in dB del campione. Come le alte codifiche osservate, la block coding introduce una perdita di precisione.

10^{-e} : esponente utilizzato dalla codifica (e: numero di bit usato dalla codifica)
m: mantissa utilizzata dalla codifica

Come lavora questa codifica:
Viene preso il grafico della forma d'onda e partito in blocchi individuati lungo l'asse delle ascisse (questi blocchi possono o meno essere uniformi). Preso un blocco, in esso sarà presente un pezzetto di funzione; in una normale quantizzazione ogni pezzo di funzione cadrebbe in quanti diversi, qui invece si sceglie un unico esponente per blocco (e) al quale tutto il pezzo di funzione in esso contenuto viene approssimato per effetto dell'operazione di quantizzazione. Ma una cosa del genere, normalmente, sarebbe causa di gravissimi errori in fase di ricostruzione del segnale poiché le variazioni della funzione non verrebbero con considerate! Ciò però non avviene poiché per ognuno dei campioni contenuti in uno stesso blocco si salva la mantissa, grazie a questa sarà poi possibile costruire fedelmente i valori con le loro variazioni.
Tuttavia questa tecnica introduce la problematica dei "pre-echi": quando in un blocco la variazione fra 2 (o più) punti della funzione è estrema, cioè molto alta, l'approssimazione ad un solo valore di e è impossibile poiché l'informazione che si andrebbe a perdere è troppo elevata; una soluzione che verrebbe in mente è quella di aggiungere più esponenti per i blocchi in cui ciò avviene, ma dal punto di vista di funzionamento di questo metodo questa scelta ne diminuirebbe di molto le prestazioni.

Ci sono 2 metodi per risolvere il problema dei pre-echi:
1) Ridurre la durata di tutti i blocchi fino a quando le variazioni delle funzioni nei blocchi non sono tutte più o meno equilibrate
2) Ideare una suddivisione in blocchi non-uniforme in modo tale da spezzettare i pezzi di funzione in cui la variazione è elevata; in questa maniera si produrranno più blocchi in cui la variazione è più contenuta

L'epsilon di macchina è il più basso valore numerico che una macchina è in grado di rappresentare, se la macchina avesse a che fare con un valore più piccolo di questo, lo approssimerebbe ad epsilon macchina (è bene dare questa definizione poiché la rappresentazione tramite un esponente e una mantissa è una rappresentazione numerica di macchina).

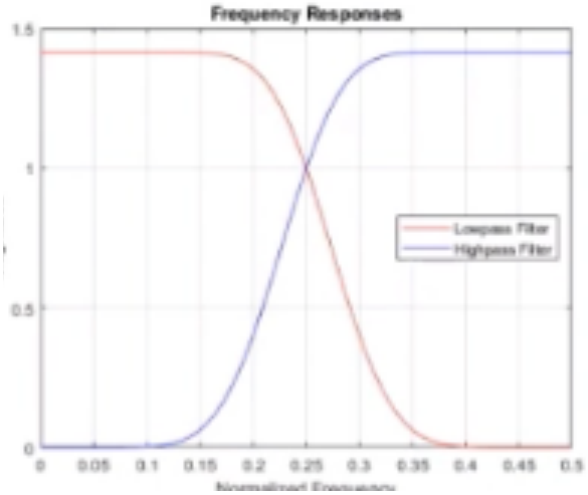
Transform coding
Utilizza delle trasformate differenti dalla DFT (trasformata che era invece impiegata nella codifica a blocchi); essa sfrutta la FFT o la DCT, tuttavia la seconda è preferibile alla prima dato che opera con valori apparente all'insieme dei numeri reali.
Questa codifica si comporta come la precedente, solo che è più vantaggiosa quando si ha che fare con un segnale la cui forma d'onda ha una bassa gamma dinamica (ossia varia in maniera controllata).
Per evitare i pre-echi questa codifica calcola la trasformata su blocchi (finestre) sovrapposti, in questa maniera si ottiene un valore medio fra i due (questa è in tutto e per tutto un'operazione di windowing). Questa tecnica è efficace perché di fatto previene il problema dei pre-echi però produce il doppio dei campioni necessari a raffigurare il suono.
La compressione che utilizza questa codifica avviene nel dominio delle frequenze, infatti lo strumento matematico della trasformata di Fourier attua il cambio dal domino del tempo a quello delle frequenze.

Nota: visto che questa codifica produce uno spettro, ossia una funzione, su questo risultato si può applicare successivamente una codifica a blocchi (si esegue insomma una codifica sulla codifica) che però agisce nel dominio delle frequenze.
Considerazioni -> la codifica a blocchi non agisce in un dominio particolare (tempo o frequenza che sia) essa è solo una tecnica che opera su dei grafici di funzione. Qualora il range dinamico dello spettro ottenuto dalla codifica di transform coding abbia un basso valore di range dinamico (ossia questa funzione non varia in maniera "violenta") è consigliabile eseguire la codifica a blocchi, questo perché il risultato finale sarà ulteriormente semplificato.

Sub-band
È il nome che assume la codifica a blocchi quando viene eseguita nel dominio delle frequenze. Il processo di suddivisone dello spettro in blocchi (ossia in bande di frequenza, dato che di fatto queste si stanno individuando) è detto "band-splitting".
Le bande ottenute possono essere o uniformi o variabili, ma grazie a Bark è noto che le bande di percezione della frequenza per l'essere umano sono scostanti, quindi è più ingegnoso adottare delle bande di frequenza a grandezza variabile che seguano lo schema dato da Bark -> in questa maniera la qualità dell'audio in uscita risulterà migliorata.
Per realizzare un band-splitting che tenga conto delle proprietà psico-acustiche scoperte da Bark, si ricorre ad una variante della MDCT (trasformata discreta del coseno).

Discusso come funziona questa codifica, si osservi ora come implementarla; ci si serve di un filtro che applica una divisione non-uniforme delle bande.
Il QMF (Quadrator Mirror Filter) combina l'effetto di un filtro passa alto e un filtro passa bassa per sezionare una precisa banda di frequenza; ponendo a cascata diversi filtri QMF è poi possibile individuare le diverse bande che consentono la realizzazione della codifica.

Si osservi di seguito il grafico della funzione implementata al QMF; l'area di intersezione fra la funzione rossa (filtro passa-basso) e la funzione blu (filtro passa-alto) è la banda di frequenza individuata dal filtro.



Codifica di Huffman
Codifica più semplice da implementare. Ha 3 caratteristiche principali:

-è la codifica che più si avvicina al limite di Shannon
-elimina i prefissi, eliminando le ambiguità in fase di ricezione
-è loseless

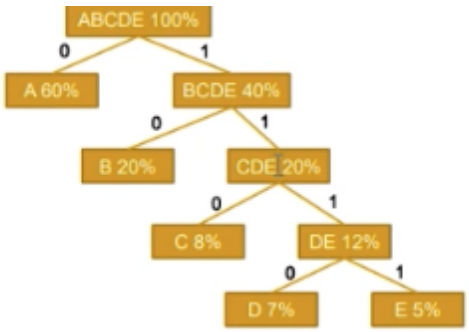
Si spieghi meglio la caratteristica riguardante i prefissi:
Si ipotizzi di avere una codifica del genere:
0=A; 1=B; 10=C; 11=D
Si ipotizzi che il dispositivo riceve la parola logica “11111”
Come devo procedere per decodificare? Per iniziare devo prendere un solo bit 1 e tradurlo come B? O devo prendere due bit 1 e tradurli come D?
Questo è un esempio di codifica che sfrutta prefissi e che crea delle ambiguità in fase di decodifica.
(ovviamente al posto di A,B,C,D immagina ci siano valoriampiezza in dB tradotti in informazione numerica)
L’ambiguità nasce dal fatto che l’informazione circa B e codificata come 1, è prefisso dell’informazione circa D codificata come 11; la codifica di Huffman elimina il sistema dei prefissi eliminando questo problema.

L’algoritmo di questa codifica è greedy, ciò significa che in corso d’opera prende sempre la decisione di rappresentazione dei dati più vantaggiosa: elementi della funzione che avranno alti valori di frequenza, verranno salvati in porzioni di bit molto ristrette (ciò diminuisce la precisione in fase di ricostruzione, ma poco importa perché, come è stato già visto, gli elementi di frequenza elevata attraversano un grande numero di quanti, quindi la loro variazione è sempre ben nota); elementi della funzione che avranno bassi valori di frequenza, verranno salvati in porzioni di bit più grandi, in questo modo sarà possibile trasmettere un maggiore numero di informazioni circa le piccole variazioni che questi elementi della funzione seguono.
L’algoritmo di Huffman non è deterministico, esso cioè codifica i valori che tratta secondo una logica ben precisa, questa prevede di assegnare una codifica tanto piccola quanto è grande la probabilità di codificare uno stesso valore e viceversa.

Si osservi un esempio:
Siano dati i valori (sempre da interpretare come ampiezze in dB) A, B, C, D, E con i loro valori di probabilità di incontro:

A: 60% - B: 20% - C: 8% - D: 7% - E: 5%

Si vuole dare una codifica a questi valori, per farlo si sfrutta una struttura ad albero:



Spiegazione del procedimento:
Nella parte più bassa dell'albero vengono collocati i 2 valori con la probabilità di incontro più bassa (D e E), mentre in cima quello con probabilità di incontro più alta (A); in mezzo alla base e alla cima vengono inseriti gli altri valori di incontro, sempre rispettando un ordine di probabilità crescente.
L'algoritmo procede realizzando dei nodi che uniscono fra loro le diverse "foglie" di quest'albero, partendo dal basso verso l'alto: il nodo fra D e E è il nodo DE; il nodo fra C e DE è CDE…
Quando tutti i nodi sono fra loro uniti si numera ramificazione con un bit (in questo esempio il ramo di destra è stato numerato con 1 mentre quello di sinistra con 0, ma nulla vieta di fare al contrario, è una questione di stile). Percorrendo i rami dalla cima, fino al valore che si vuole codificare, si ottiene la sua codifica in bit. Perciò:

A: 0
B: 10
C: 110
D: 1110
E: 1111

Nota:
A è il valore la cui probabilità di incontro è la più alta, infatti come accennato sopra è quello la cui codifica è la più piccola -> ciò è perfettamente sensato perché avendo che A sarà il valore che trasmetterò più spesso è mio interesse fare sì che sia salvato in un formato piccolo per accelerarne la trasmissione.
D ed E son invece i valori la cui probabilità di incontro è la minore, pertanto sono salvati sulla codifica di dimensione maggiore -> è vero che sono salvati su una maggiore mole di dati però è raro che vengano trasmessi, quindi a livello di prestazione non incidono.
Nessuno dei valori assegnati in fase di codifica è prefisso degli altri: nessuna sequenza inizia con 0; nessuna sequenza inizia con 10; nessuna sequenza inizia con 110…

Considerazioni finali:
Block-Coding: segmenta la traccia audio in frame temporali
Transform coding: analizza nel dominio delle frequenze le tracce audio
Sub-band: ottimizza la codifica del range dinamico nelle frequenze
Huffman: elimina le ambiguità sfruttando una codifica loseless
ADPCM: elimina le ambiguità sfruttando una codifica lossy