UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Quinquennale in Informatica

"Metodi e prototipi software per la sincronizzazione automatica di segnali audio musicali compressi MP3 e partiture XML"

Relatore:
Chiar.mo Prof. G. Haus

Correlatori:
Dott. G. Vercellesi
Ing. L.A. Ludovico

Tesi di Laurea di:
Antonello D'Aguanno
Matr. 581564

Anno Accademico 2004 / 2005

Un grazie sincero a tutto lo staff del L.I.M. per aver reso piacevole questo lungo percorso, in particolare ad Adriano per l'infinita disponibilità

Un pensiero particolare spetta alla mia ragazza, Chiara, che ha sempre visto oltre

Ringrazio il prof. Goffredo Haus per la fiducia dimostratami e per avermi fatto scoprire l'aspetto più affascinante dell'informatica

Grazie anche a Luca e Giancarlo per i preziosi consigli e per essere stati sempre presenti

Dedico questo lavoro ai miei genitori e ai miei nonni

Indice

Capitolo 1

Introduzione

La grande quantità di librerie audio digitali compresse accessibili sia localmente sia tramite intranet che Internet, giustifica immediatamente la necessità di sistemi in grado di gestire il segnale audio con la stessa facilità ed efficacia con cui si manipolano ad esempio i testi. Il valore delle informazioni contenute in un archivio dipende dall'efficacia dei metodi utilizzati per trovarle, catalogarle e indicizzarle. Senza gli opportuni strumenti per la loro gestione, questo enorme quantitativo di dati disponibile può essere sfruttato solo parzialmente [1].

Appare sempre più stringente il vincolo di una ricerca limitata ad informazioni quali: titolo, autore, genere (quando queste siano poi effettivamente disponibili e corrette). Molti sforzi sono stati fatti per oltrepassare questo vincolo, per arrivare ad una "vera" catalogazione e ricerca nei file audio degli oggetti musicali: *"Quegli eventi sonori che il sistema percettivo umano riesce ad analizzare ed isolare in una qualsiasi complessità sonora"* [3], come ad

esempio una particolare progressione di accordi, la voce principale di un brano, o la sua tessitura ritmica...

Tutte queste problematiche hanno generato una disciplina scientifica a se stante il *"Music Information Retrieval"* (MIR).

Il primo passo per avere dei sistemi di gestione evoluti è avere degli affidabili sistemi di comprensione automatica. Per comprensione s'intende la trasformazione di un segnale musicale in una rappresentazione simbolica che descriva l'informazione relativa alle note suonate (altezza, durata, intensità e punti d'attacco), agli strumenti utilizzati e ad alcune caratteristiche dell'interpretazione dell'esecutore (tempo del brano, crescendi, legato, vibrato, ecc.) [3].

La conoscenza di queste informazioni permette di superare le ricerche tramite metadati quali genere, autore, titolo finora utilizzate.

A questo proposito si possono citare gli studi compiuti sull'estrazione automatica di partiture (*Pitch Tracking*) cioè partire da un qualsiasi frammento musicale e lasciare che il computer estragga automaticamente l'altezza delle note presenti, oppure l'estrazione dei beat (*Beat Tracking*) cioè riuscire a determinare in maniera automatica la velocità metronomica di un brano e la disposizione dei beat.

Queste tecniche fanno parte di quel filone di ricerca che è definito come "analisi cieca", in altre parole la ricerca dell'informazione musicale si basa sulla sola analisi del segnale a basso livello e quindi su conoscenze derivanti fondamentalmente dall'acustica.

Queste tecniche sono molto valide solo per file audio poveri sia dal punto di vista timbrico che armonico, per file più complessi questi metodi tendono a commettere errori. Nemmeno i prodotti commerciali oggi disponibili sul mercato che permettono un'estrazione automatica della partitura promettono dei risultati perfetti.

Citando proprio il sito ufficiale di uno tra i più famosi, "IntelliScore":

"*Don't expect IntelliScore to convert a CD to a finished MIDI file, but IntelliScore will get you well on your way. Although after using IntelliScore you may need to clean up the MIDI file, users say this saves them an average of 35% over having to figure out and enter the notes into a sequencer without IntelliScore help.*" [10].

Questi risultati, in linea con tutti i migliori algoritmi di pitch tracking, risultano molto utili per eseguire la trascrizione di un brano, ma li rendono, per ovvi motivi, poco utili per il

reperimento di informazioni affidabili e realmente utili se applicati per gestire grandi discografie.

Questo dimostra come sia effettivamente possibile estrarre informazioni significative dall'audio ma anche come si abbia un limite lavorando solo su questa parte.

L' esigenza di superare questi limiti, la necessità di avere dei sistemi che permettano di navigare contemporaneamente la partitura e il file audio ha fatto nascere un altro filone di ricerca molto interessante:

L' "*Automatic Synchronization of Musical Data*" [60] che trova la sua trattazione formale in:

"Automatic Synchronization of Musical Data: A Mathematical Approach" [73]

Data la partitura simbolica di un brano e data la sua esecuzione musicale lo scopo di questa famiglia di algoritmi è fornire un output che leghi ogni evento musicale presente sullo spartito con il suo inizio nel relativo file audio.

Le sue applicazioni sono molteplici, basti pensare alla didattica della teoria musicale, alla musicologia e come già detto al reperimento di frammenti melodici all'interno di archivi musicali.

Tutti gli algoritmi finora proposti hanno però un limite congenito: operano esclusivamente su codifiche non compresse PCM, risultando quindi poco utili visto che questi, per loro natura, non si prestano molto alla conservazione di brani musicali.

Questo lavoro di tesi nasce con l'idea di fornire un nuovo approccio al problema della sincronizzazione di oggetti musicali che bene si adatti alle caratteristiche fondamentali del formato MP3.

Un passo avanti è stato fatto anche per il formato simbolico di partitura utilizzato passando dal formato MIDI, unico formato supportato ad oggi [3][8][59][60], a formati di tipo Mark-up derivanti da XML come il MusicXML [30] e MX [29]. Questo porta il grande vantaggio di poter avere un unico file di output che abbia al suo interno non solo la parte simbolica ma anche il file audio, eventuali file video e le relative ancore di temporizzazione.

Un altro aspetto importante che ha portato a questo lavoro di tesi, è la qualità dei risultati raggiunti dai migliori algoritmi di sincronizzazione. Lo stato dell'arte di questi algoritmi appare ancora in fase embrionale non offrendo risultati perfetti in ogni situazione e in alcuni casi sincronizzando ad esempio solo l'inizio delle battute musicali o solo le note con accenti forti. In questa dissertazione invece si è cercato di giungere a dei prototipi che offrano la

possibilità di sincronizzare tutti gli oggetti musicali presenti in partitura indipendentemente dalla loro durata e dalla loro posizione rispetto ad accenti deboli o forti.
Questa scelta è stata fatta perché è lecito pensare a due necessità distinte di sincronizzazione. La prima legata principalmente ad aspetti indipendenti dalla musicologia, in cui sono tollerabili errori anche di decimi di secondo ma che devono funzionare per il maggior numero possibile di brani, mentre la seconda, legata maggiormente ad aspetti musicologici, che offra una precisione elevatissima, nell'ordine dei centesimi di secondo, che permetta di compiere analisi sull'esecuzione del brano, sulle caratteristiche di un musicista, sul suo stile interpretativo.

Questo lavoro di tesi si propone quindi di indagare nuove tecniche di analisi dei segnali audio complessi (polifonici e politimbrici) nel dominio compresso.
Nel capitolo 2 sarà descritto lo standard MPEG e nello specifico il formato MP3, evidenziandone gli aspetti che rendono inapplicabili i metodi finora proposti.
Nel capitolo 3 saranno presentati i formati MusicXML e MX evidenziando le motivazioni che ci hanno portato alla scelta di utilizzare questi formati.
Nel capitolo 4 sarà presentato lo stato dell'arte del MIR, dell'analisi cieca del segnale e della sincronizzazione automatica audio partitura, sia nel dominio compresso sia nei classici formati PCM.
Nel capitolo 5 verranno analizzati tutti gli approcci sviluppati per effettuare la sincronizzazione direttamene su un formato compresso, mostrando grafici sul loro grado di affidabilità, confrontandone i risultati e motivando le scelte di abbandonare o indagare ulteriormente determinati ambiti.
La descrizione del prototipo software proposto troverà spazio nel capitolo 6 in cui saranno presentate non solo le principali funzioni e strutture dati utilizzate, l'analisi dei costi computazionali, e i risultati raggiunti rispetto agli algoritmi finora utilizzati ma anche le motivazioni che hanno portato alla scelta dei vari linguaggi di programmazione utilizzati e un commento sui brani di test prescelti.
Le conclusioni saranno contenute nel capitolo 7 che comprenderà anche gli sviluppi futuri, le ottimizzazioni possibili e le applicazioni pratiche di questi algoritmi.
Una breve appendice sarà dedicata ad un manuale utente per l'utilizzo del prototipo e dei software correlati Infine, i riferimenti bibliografici concluderanno il lavoro.

Capitolo 2

MPEG e lo standard MP3

Il **Moving Picture Experts Group** (MPEG) è un gruppo di lavoro di ISO/IEC che ha il compito di sviluppare standard internazionali per la codifica, compressione, gestione di informazioni audio, video e delle loro possibili combinazioni.

Ad oggi MPEG ha introdotto 5 standard principali che sono tutti identificati dalla sigla MPEG e dal relativo numero: MPEG-1, MPEG-2, MPEG-4, MPEG-7, MPEG-21.

MPEG-1 è lo standard relativo alla codifica dei dati e alla loro organizzazione all'interno dei file sia per l'audio sia per i filmati e per le loro combinazioni.

MPEG-2 è lo standard per la televisione digitale, mentre MPEG-4 standardizza le applicazioni multimediali.

In MPEG-7 troviamo invece le informazioni per la standardizzazione dei meta-dati utilizzati per fare ricerche, filtraggi e manipolazioni dei contenuti dei file.

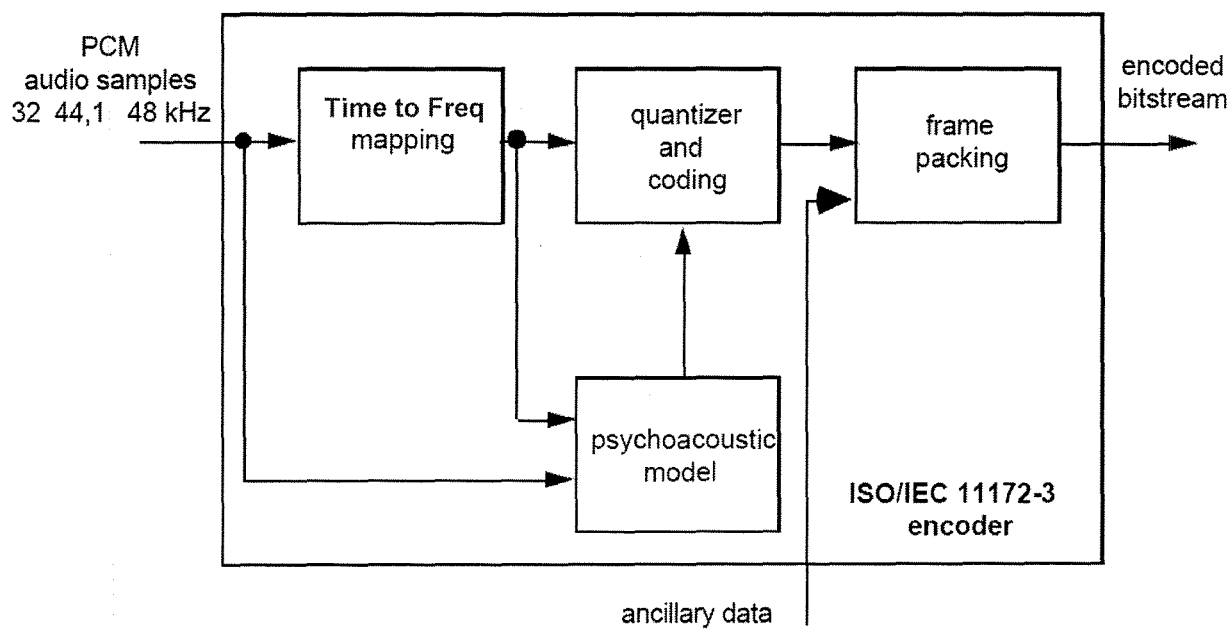
MPEG-21 è la standardizzazione del "multimedia framework", che formalizza tutto il ciclo vitale di un'informazione multimediale, dalla sua creazione fino alla fruizione da parte dell'utente finale.

Lo scopo della codifica MPEG-1 audio è la riduzione del flusso di dati audio mediante la rimozione dell'informazione che non è percepita dall'orecchio umano e che viene definita come *"Irrilevante"*. Gli algoritmi tramite i quali è possibile comprimere il segnale audio, sfruttando queste caratteristiche, sono definiti come algoritmi di compressione percettivi.

Anche in MPEG-2 è presente una sezione di codifica audio che espande il campo di applicazione di questo standard ad ambiti che richiedano una compressione elevata. Considerato l'ambito applicativo di questo lavoro considereremo solo lo standard MPEG-1 che è quello comunemente utilizzato per la codifica dell'informazione musicale.

Lo standard MPEG-1 offre 3 diversi layer di codifica dell'informazione audio. I layer I, II, III. Tutti gli algoritmi si basano sullo stesso principio di funzionamento: prevedono come input dei dati PCM campionati a 32; 44,1 e 48 kHz e forniscono un output compatibile con lo standard che offre vari livelli di compressione. Nella figura 2.1 è presentato un diagramma a blocchi molto semplice del funzionamento di un generico encoder MPEG.
È da notare che lo standard non impone un algoritmo unico, ma piuttosto delle "linee guida" sul suo funzionamento.

Figura 2.1 Diagramma a blocchi basilare di un encoder MPEG-1 [74]



I layer hanno diversi utilizzi e applicazioni. La loro numerazione è direttamente proporzionale alla complessità computazionale e inversamente proporzionale alla qualità audio ottenibile rispetto alla dimensione dei file ottenuti dopo la compressione.
È da notare come l'aumento del costo computazionale si rifletta maggiormente sulla parte dedicata alla codifica dell'informazione piuttosto che sulla sezione di decodifica. Il Layer I è il meno costoso computazionalmente, ma offre una minore capacità di rimuovere le irrilevanze presenti nel segnale audio, è quindi indicato in quegli ambiti in cui la semplicità è un fattore fondamentale.
Il layer II nasce per applicazioni in cui abbiamo un encoder che serve tanti decoder in pratica in quelle applicazioni identificate come uno a molti.
Il layer III offre le migliori prestazioni nella gestione dell'informazione, ma è anche il più complesso. È il più indicato quindi per la codifica di dati "stabili" vale a dire in cui non è molto rilevante il tempo di codifica ma la loro qualità e il basso ingombro in termini di bit in previsione di una conservazione dei dati molto lunga.
Il formato MP3 è l'abbreviazione di standard MPEG-1 layer III e MPEG-2 layer III. Questo potente algoritmo è stato pubblicato ufficialmente come ISO-MPEG Audio (IS 11172-3 ed IS 13818-3) [74][75].
Accetta in ingresso campioni PCM campionati a 32 kHz, 44,1kHz e 48 kHz, restituendo un beatstream con un livello di compressione che va da 32 kbit/s fino a 320 kbit/s. Ovviamente minore è il bitrate utilizzato maggiore è il numero di artefatti inseriti nel file risultante.
Il contenuto di un file MP3 è organizzato in frame o bitstream [16][11][17][18], ognuno dei quali contiene informazioni per ricostruire i corrispondenti 1152 campioni PCM, in modo indipendente da tutto il resto del bitstream. Il frame è poi suddiviso in due parti logiche chiamate granuli, ognuna delle quali corrisponde a 576 campioni PCM [14].
Oggi sono disponibili moltissimi encoder MP3, sia hardware sia software, e quasi tutti i programmi audio offrono la piena compatibilità nei confronti di questo standard. MP3 è ad oggi il formato più usato per i file audio su internet, ma non solo, sta prendendo sempre più piede nella fruizione della musica da parte di ogni tipologia di utente, da queste considerazioni la scelta di orientare questo lavoro di tesi su questo formato.

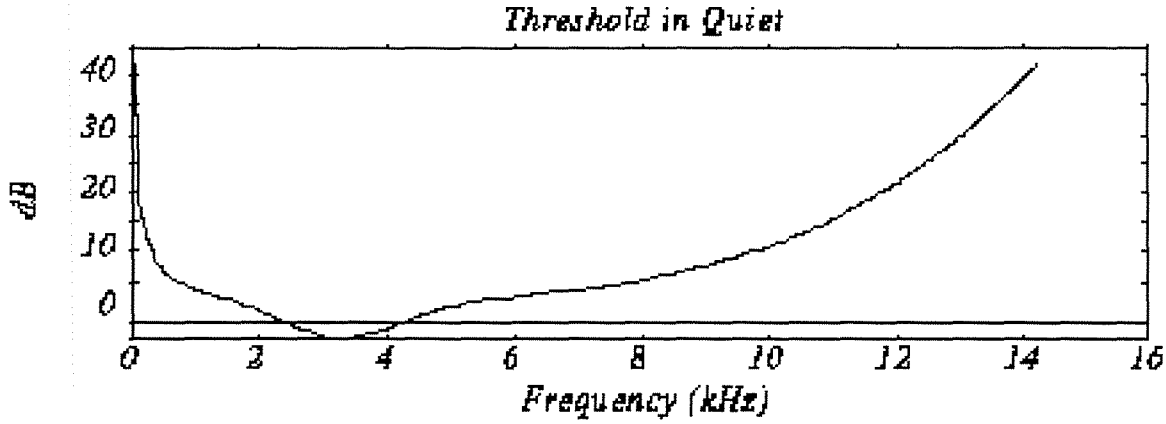
2.1 Il Sistema uditivo umano

L'orecchio umano può essere considerato in prima approssimazione come un banco di filtri passa banda che copre un'ampiezza di circa 10 ottave, ragionando in termini musicali, o frequenze da 20 Hz a 22 kHz in termini fisici. Ha un'ampiezza di banda di circa 50-100 Hz per segnali inferiori a 500 Hz e circa 5000 Hz per segnali ad alte frequenze. È composto da 26 bande critiche che coprono tutto lo spettro udibile [12]. Queste caratteristiche fisiche sono dovute alla conformazione fisiologica della membrana interna dell'orecchio. Questa membrana a varie zone che si differenziano in larghezza, spessore e rigidità di conseguenza differenti aree della membrana vibrano a frequenze differenti. Questo comportamento è alla base degli effetti di mascheramento psico-acustici fondamentali per ogni algoritmo di codifica percettiva.

2.1.1 Principi di Psicoacustica

In virtù della conformazione fisica del nostro sistema uditivo molte informazioni contenute in un suono non sono percepite; per esempio, noi non sentiamo intensità al di sotto di una certa soglia in funzione della frequenza oppure percepiamo come più "forti" toni con frequenza appartenente allo spettro vocale. Ciò significa che del segnale audio originale è necessario memorizzare soltanto le informazioni effettivamente percepite, eliminando tutto il resto, ed ottenendo così un'elevata compressione in cui la perdita d'informazione c'è, ma, *in linea di principio*, non si sente [14]. In figura 2.2 è presentata la soglia uditiva dell'orecchio umano in funzione di frequenza e pressione sonora. Tutti i suoni posti al di sotto della curva non saranno percepiti dall'ascoltatore e sono quindi irrilevanti.

Figura 2.2: curva di percezione del suono del nostro orecchio in stato di quiete [11]

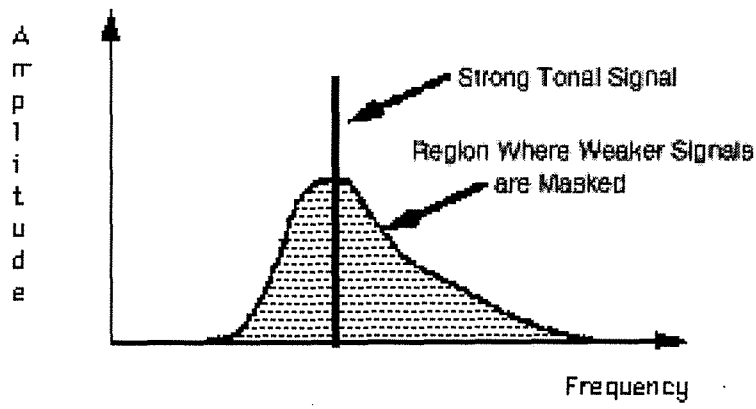


Gli effetti di mascheramento psicoacustici non sono statici rispetto all'ambiente, ma variano in funzione degli altri suoni percepiti. Componenti frequenziali che in stato di quiete sarebbero perfettamente udibili possono essere mascherati da frequenze più forti adiacenti ad esso sia in senso temporale sia frequenziale dando luogo ad effetti di mascheramento.

mascheramento frequenziale

Considerando due toni con frequenze molto vicine emessi nello stesso istante può avvenire che una frequenza nasconda l'altra rendendola inudibile. In figura 2.3 si osserva che tutte le componenti tonali poste sotto l'area ombreggiata non sono udite, sono quindi irrilevanti e possono essere eliminate dall'MP3 risultante.

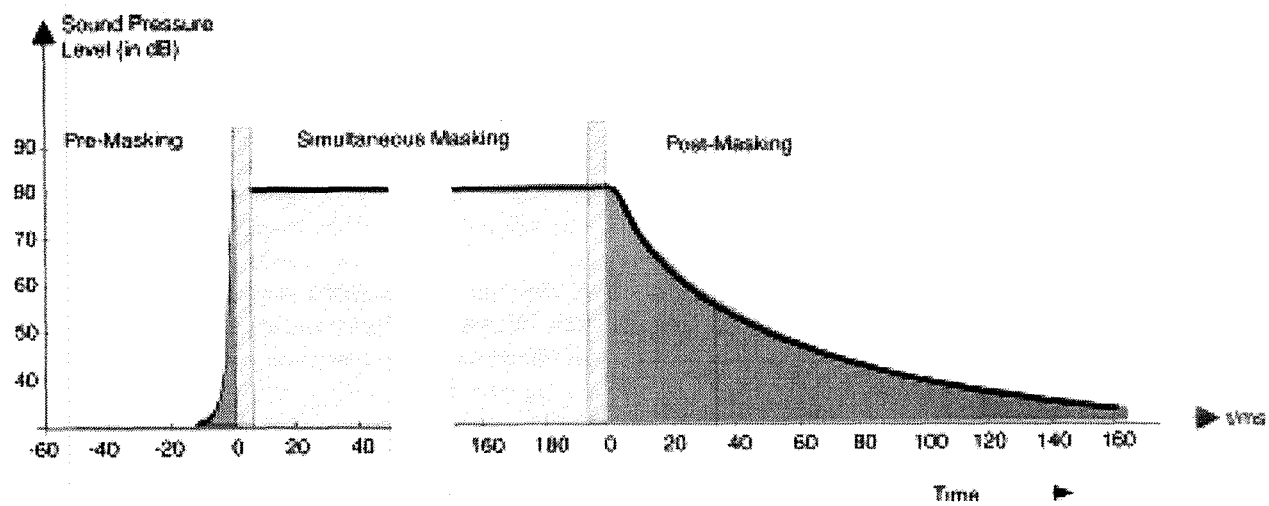
Figura 2.3 mascheramento frequenziale [11]



mascheramento temporale

Si verifica quando due toni a frequenze diverse vengono generati ad istanti di tempo diversi ma sufficientemente vicini. Come si vede dalla figura 2.4 esistono 2 tipi di mascheramento temporale: *pre-masking* e *post-masking*. Il primo si verifica subito prima dello stimolo uditivo e comporta che la parte ombreggiata a sinistra dello stimolo uditivo non sia percepita pur arrivando a "colpire" il timpano prima del tono realmente percepito. Questo effetto di mascheramento prosegue anche dopo la percezione del segnale bloccando anche gli stimoli successivi che si trovino al di sotto dell'area più scura a destra della figura 2.4.

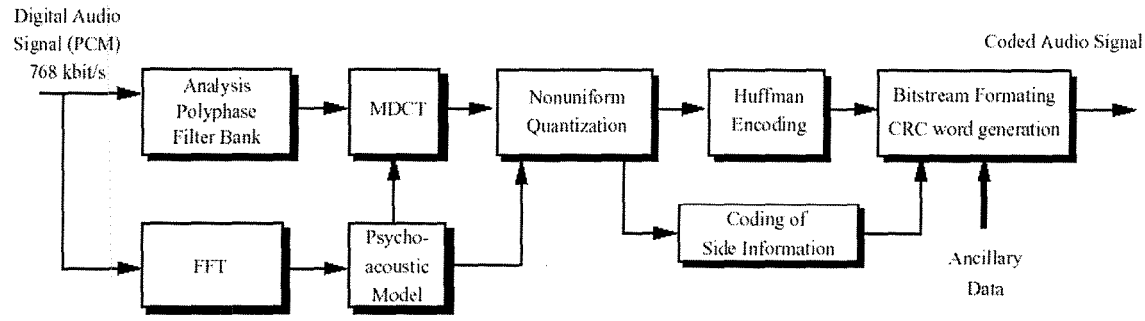
Figura 2.4 mascheramento temporale [12]



2.2 Encoder MP3

L'encoder MP3 può essere schematizzato dalla figura 2.5. Andremo ora ad analizzare brevemente le caratteristiche dei blocchi più importanti.

Figura 2.5: struttura dettagliata di un Encoder MP3 [16]



Banco di Filtri Polifasico (Analysis Polyphase Filter Bank)

La prima fase del processo di codifica è il filtraggio del segnale audio attraverso un banco di filtri. La sequenza di 576 campioni PCM è filtrata parallelamente da 32 filtri passa-banda equispaziati. In ogni sottobanda sono quindi elaborati 18 (576/32) campioni PCM.

MDCT (Trasformata Coseno Discreta Modificata)

Il blocco MDCT è una delle caratteristiche fondamentali di MP3 nonché una sua caratteristica peculiare non essendo presente negli altri layer dello standard MPEG-1.
L'MDCT [21][22] è stata inserita per fornire una risoluzione frequenziale maggiore al *Quantizzazione Non-uniforme* permettendo quindi di sfruttare al meglio i risultati del modello psicoacustico. L'MDCT riceve in ingresso il segnale mappato nelle 32 sottobande dal banco di filtri polifasico da cui genera un segnale in cui ogni sottobanda è suddivisa in ulteriori 18 parti ottenendo così un totale di 576 linee frequenziali. Questi sono i valori appartenenti al primo granulo; l'MDCT viene rieseguita con un overlap del 50% per ottenere le frequenze del secondo granulo[14]. In presenza di forti rumori impulsivi ed in generale quando il segnale lo richiede l'MDCT non utilizza più i parametri sopra indicati ma effettua una transizione verso un altro tipo di configurazione in cui ogni sottobanda viene suddivisa da 18 a 6 parti.

La scelta relativa a quale configurazione dell'MDCT utilizzare è delegata ad un'operazione di finestratura che in base al valore assunto dal parametro di *Entropia Psicoacustica*, calcolato dal modello psicoacustico, seleziona la configurazione più adatta per l'MDCT. Le finestrature possibili sono 4:

1. *Long*
2. *Short*
3. *Long to Short*
4. *Short To Long*

La finestra di tipo *long,* è selezionata nel caso in cui è necessario avere maggiore risoluzione frequenziale (sottobande suddivise in 18 parti). Il tipo *short* quando si ha la necessità di avere una maggiore risoluzione temporale (sottobande suddivise in 6 parti). Le altre 2 finestre sono "finestre di transizione" e permettono di passare in tempo reale da una finestra *short* ad una *long* e viceversa.

FFT (Trasformata Veloce di Fourier)

L'FFT fornisce uno spettro frequenziale ad alta risoluzione. Questa risoluzione molto elevata è necessaria al modello psicoacustico per poter lavorare nella maniera più efficace. Ha 2 modalità di elaborazione: a 256 e 1024 punti [23][24].

Modello Psicoacustico (PsychoAcoustic Model)

Questo blocco è il fattore principale che determina la qualità di un encoder MP3 avendo il compito di selezionare il miglior tipo di mascheramento possibile con lo scopo di minimizzare la perdita di informazione e massimizzare il risparmio di spazio in termini di bit. Alcuni modelli di riferimento sono inseriti all'interno dello standard ma ogni sviluppatore è libero di implementare questa caratteristica privilegiando aspetti quali la velocità di esecuzione, la qualità del file finale o la compressione ottenibile. Molti sviluppatori hanno implementato modelli psicoacustici proprietari tra i quali possiamo citare: Blade, Lame, Xing, Fhg, RCA, Gogo ognuno dei quali può avere varie versioni e varianti [25][26][27][28].

Quantizzazione Non-uniforme (Non-uniform Quantization)

Questa fase ha lo scopo di minimizzare il rumore di quantizzazione in funzione del bitrate selezionato. Esegue una quantizzazione non-uniforme delle linee frequenziali ottenute dal banco di filtri ibrido.

Codifica Huffman (Huffman Encoding)

Prima di essere impacchettati, i dati vengono ulteriormente compressi, senza perdita d'informazione, con l'algoritmo di Huffman. Varie tabelle di compressione specificate nello standard sono utilizzate a questo scopo.

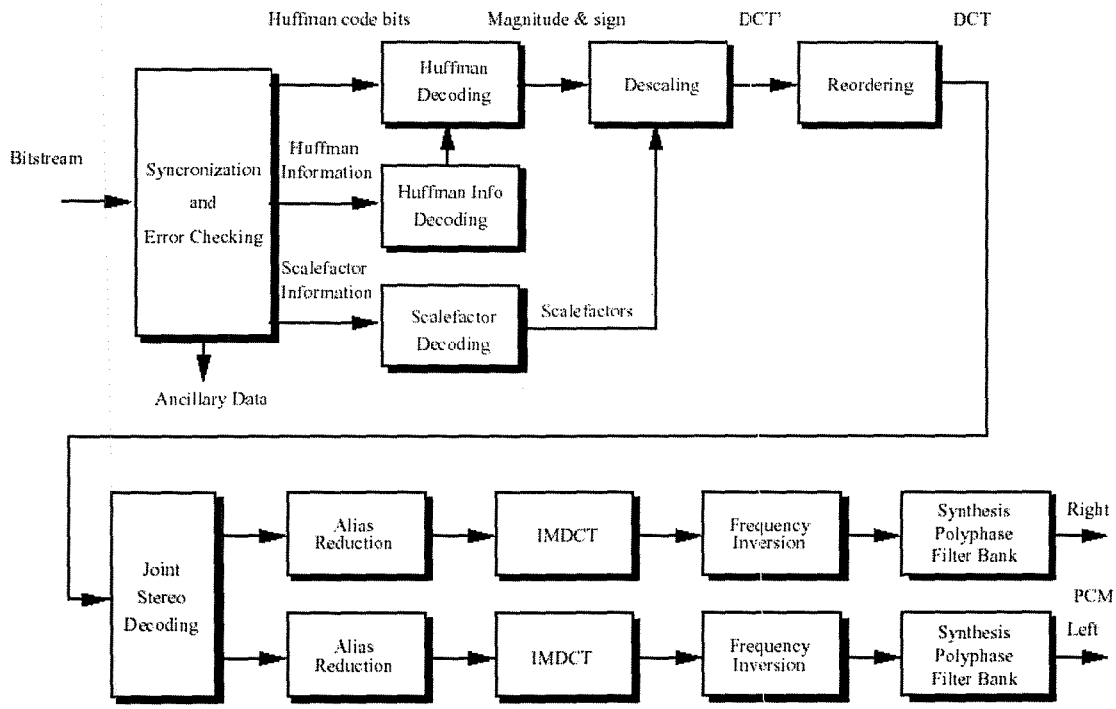
Impacchettamento Frame e generazione controllo CRC (Bitstream Formatting – CRC word generation)

L'ultimo blocco di MP3 ha il compito di inserire i frame e tutte le informazioni associate in un bitstream compatibile con lo standard.

2.3 Decoder MP3

Il decoder MP3 può essere schematizzato dalla figura 2.6. Andremo ora, come per l'encoder, ad analizzare brevemente le caratteristiche dei blocchi principali.

Figura 2.6 struttura dettagliata di un decoder MP3 [16]



Sincronizzazione e Controllo d'Errore (Syncronization and Error Checking)

Questo blocco riceve il bitstream MP3 e identifica la posizione di ogni frame.

Decodifica Huffman (Huffman Decoding – Huffman Info Decoding)

Esegue l'algoritmo di decodifica di Huffman selezionando quali tabelle utilizzare e dove. Questo blocco genera in output le 576 linee frequenziali del granulo analizzato; se ce ne sono di meno, le mancanti vengono sostituite da frequenze nulle in modo da ottenere sempre 576 valori.

Riordinamento delle frequenze (Reordering)

Questo blocco è attivato solo se in fase di encoding si è utilizzata una finestra di tipo *short*, infatti, in presenza di questo tipo di finestratura si ha un diverso ordinamento delle frequenze.

Queste appaiono ordinate prima per finestra e quindi per frequenza è dunque necessario effettuare un'operazione di ordinamento per ottenere la corretta sequenza di valori.

IMDCT (Inverse MDCT)

Il compito di questo blocco è l'esecuzione della Trasformata Coseno Discreta Modificata Inversa che trasforma lo spettro lineare composto da 1152 (576 + 576) valori in una matrice composta da 32 bande ognuna composta da 36 valori. Questo spettro è l'input per il banco di filtri polifasico. Ovviamente anche l'IMDCT utilizza la finestratura utilizzata in fase di encoding dall' MDCT.

Sintesi tramite Banco di Filtri Polifasico (Synthesis Polyphase Filter Bank)

L'ultimo blocco del decoder ha il compito di risintetizzare lo spettro in ingresso in modo da fornire un output di 1152 campioni PCM.

Capitolo 3

I formati MusicXML e MX

XML (Extensible Markup Language) è un linguaggio di mark-up disegnato per descrivere i dati contenuti in un documento. Per linguaggio di marcatura (mark-up language) si intende una codifica che fa uso di un insieme di marcatori (mark-up o tag) convenzionali, ossia aderenti a regole chiare, definite e comunemente accettate [29][33].

Un linguaggio di marcatura deve specificare:

- qual è il significato di ogni mark-up
- come si distinguono i mark-up dal testo
- quali mark-up sono consentiti
- quali mark-up sono richiesti

In XML i tag non sono prededefiniti ma devono essere dichiarati dall'utente tramite un DTD (Document Type Definition) o un XML Schema. Altra caratteristica saliente di questo formato è la sua particolare disposizione a rappresentare informazioni con una forte struttura gerarchica. Questo linguaggio è divenuto una raccomandazione del W3C il 10 febbraio 1998 [34] e si presta molto bene a descrivere l'informazione musicale [31].

3.1 MusicXML

MusicXML è un linguaggio per la notazione simbolica di partiture occidentali derivato direttamente da XML tramite la creazione di appositi DTD. Tutti i file scritti in questo formato devono essere compatibili con questi DTD. Questo formato è stato introdotto con il non facile obiettivo di fornire una piattaforma comune per lo scambio di partiture musicali. Prima della nascita di formati musicali derivati da XML l'unico formato adatto allo scambio di informazione musicale simbolica era il MIDI. Questo standard ha però dei grossi limiti nella rappresentazione delle partiture, ad esempio non ha la possibilità di rappresentare le legature, la direzione della coda di una nota e molte altre caratteristiche proprie della notazione musicale. I formati che gestiscono direttamente la rappresentazione grafica della musica, ad esempio il NIFF, sono molto più completi dal punto di vista notazionale ma non sono una soluzione migliore per lo scambio di partiture, in quanto hanno, ad esempio, dei grossi limiti se utilizzati in applicazioni su database.
Lo scopo di MusicXML appare subito chiaro leggendo la documentazione ufficiale:

> *"MusicXML is designed to meet the interchange needs for all these types of applications(sequencing, musical data-base, music notation)." [30]*

MusicXML è basato su due formati accademici:

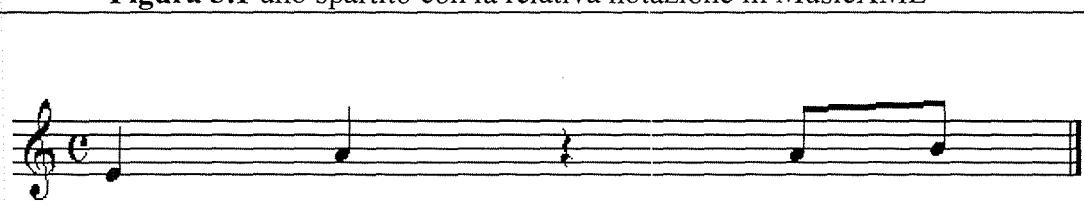
- MuseData, sviluppato da Walter Hewlett al "Center for Computer Assisted Research in the Humanities (CCARH), Stanford University" [35]
- Humdrum, sviluppato da David Huron, "Ohio State University" [36]

Questo formato è sostanzialmente una riedizione in XML di MuseData con l'aggiunta di alcune caratteristiche chiave di Humdrum.
La scelta di utilizzare questo formato per la rappresentazione della partitura è stata una scelta quasi obbligata considerando:

- il grande numero di applicativi che ne permettono una gestione completa (tra i più famosi si possono citare Finale for Windows, Sharp Eye Music Reader)
- la grande quantità di partiture esistenti in questo formato
- la sua predisposizione proprio alla condivisione di partiture
- la possibilità di lavorare su di un file testo e quindi leggibile tramite i comuni editor.

Nella figura 3.1 si può osservare la codifica MusicXML della battuta posta in alto nell'immagine. Sono stati evidenziati i blocchi di maggior interesse all'interno del file.

Figura 3.1 uno spartito con la relativa notazione in MusicXML



```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE score-partwise (View Source for full doctype...)>
- <score-partwise>
 - <identification>
   - <encoding>
       <software>Finale 2003 for Windows</software>
     </encoding>
  </identification>
 - <part-list>
   - <score-part id="P1">
       <part-name>Violino</part-name>
     - <score-instrument id="P1-I1">
         <instrument-name>
           Insieme archi 1
         </instrument-name>
       </score-instrument>
     - <midi-instrument id="P1-I1">
         <midi-channel>1</midi-channel>
         <midi-program>49</midi-program>
       </midi-instrument>
     </score-part>
  </part-list>
 - <part id="P1">
   - <measure number="1">
     - <attributes>
         <divisions>2</divisions>
       - <key>
           <fifths>0</fifths>
           <mode>major</mode>
         </key>
       - <time symbol="common">
           <beats>4</beats>
           <beat-type>4</beat-type>
         </time>
       - <clef>
           <sign>G</sign>
           <line>2</line>
         </clef>
       </attributes>
```

Score Header

Part-List

Armatura di Chiave

Segnatura di Tempo

Chiave

Intestazione del Pentagramma

```
- <note>
  - <pitch>
      <step>E</step>
      <octave>4</octave>
    </pitch>
    <duration>2</duration>
    <voice>1</voice>
    <type>quarter</type>
    <stem>up</stem>
  </note>
- <note>
  - <pitch>
      <step>A</step>
      <octave>4</octave>
    </pitch>
    <duration>2</duration>
    <voice>1</voice>
    <type>quarter</type>
    <stem>up</stem>
  </note>
- <note>
    <rest />
    <duration>2</duration>
    <voice>1</voice>
    <type>quarter</type>
  </note>
- <note>
  - <pitch>
      <step>A</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
    <voice>1</voice>
    <type>eighth</type>
    <stem>up</stem>
    <beam number="1">begin</beam>
  </note>
- <note>
  - <pitch>
      <step>B</step>
      <octave>4</octave>
    </pitch>
    <duration>1</duration>
    <voice>1</voice>
    <type>eighth</type>
    <stem>up</stem>
    <beam number="1">end</beam>
  </note>
- <barline location="right">
    <bar-style>light-heavy</bar-style>
  </barline>
</measure>
</part>
</score-partwise>
```

Altezza della nota

Nota

Il primo blocco di un file MusicXML è lo *Score Header*. Questo contiene: alcuni metadati basilari relativi allo spartito musicale (titolo, autore), la *Part-List,* che elenca tutte le parti musicali presenti nello spartito e il tipo di DTD utilizzato, in questo caso *score-partwise*. La gerarchia relativa alla vera e propria notazione musicale può essere schematizzata da due DTD distinti: *score-partwise* e *score-timewise*. Nell'esempio, relativo allo *score-partwise*, si ha una struttura con vari alberi le cui radici sono proprio le parti musicali presenti nello spartito. Ad ogni parte è associato un secondo livello con le battute ed un terzo con gli accordi e le note. Nel caso si utilizzi il DTD *score-timewise* si ha un inversione tra le parti musicali e le battute. In questo caso si ha una foresta con una battuta radice di ogni albero e le parti sotto elementi delle battute. La struttura delle informazioni relativa ad accordi e note rimane costante. Il formato Music-XML permette di passare da un DTD all'altro applicando due fogli di stile XSLT(Extensible Stylesheet Language Transformations), quindi un'applicazione che utilizzi questo formato può utilizzare indipendentemente una struttura gerarchica o l'altra. Ogni misura in MusicXML viene mormalmente numerata partendo da 1. La presenza di una eventuale battuta 0 nel file indica che il brano inizia con una battuta anacrusica.

Analizzando più approfonditamente il DTD *score-partwise* si osserva che: per ogni parte presente nello spartito il primo elemento della prima battuta contiene sempre un elemento *attributes*. All'interno di questo elemento trovano posto le informazioni relative alla chiave utilizzata *(key)* e alla sua armatura *(clef),* nonchè i dati per la segnatura del tempo *(time)*. L'elemento attributes può comparire, inoltre, in un qualsiasi punto del pentagramma per modificare informazioni relative al tempo, alla tonalità o alla chiave proprio come avviene talvolta nella comune notazione musicale in presenza di un cambio di chiave, di tempo o di tonalità.

In MusicXML il tempo è descritto da tre elementi: *beats*, *beat-type* e *divisions*. Come nella comune notazione musicale gli elementi *beats* e *beat-type* indicano rispettivamente "il numero di suddivisioni contenute nella misura e il valore di tali suddivisioni" [37]. Nella notazione classica le durate musicali sono comunemente riportate come una frazione, 1/4, 1/8 e così via, quindi ad ogni nota è associata una qualsiasi frazione che ne indica la durata. MusicXML, invece, adotta un metodo simile al MIDI per indicare la durata di una nota specificando il numero di *divisions* in cui è diviso il singolo *beat* e poi associando alla nota un campo *duration* che riporta il numero di *divisions* occupate dalla nota.

Le singole note in questo formato sono delimitate dal tag *note*. Questo elemento contiene una serie di sotto elementi che delimitano le informazioni relative alla nota presa in esame quali:

il tipo, la voce, il verso della coda. Un sotto elemento particolare, *pitch*, è dedicato all'insieme di caratteristiche che identificano l'altezza della nota. In esso trovano posto infatti le indicazioni relative al nome della nota, all'ottava di appartenenza e ad eventuali alterazioni. Le alterazioni sono riprodotte da un numero intero. Il valore assoluto di questo numero rappresenta di quanti semitoni è alterata la nota, il segno indica se si tratta di un'alterazione che aumenta o diminuisce l'altezza della nota. Differentemente da come avviene per la notazione musicale comune eventuali alterazioni in chiave non vengono considerate da MusicXML. Esse sono riportate su ogni singola nota. Ad esempio, considerando una tonalità di sol maggiore (un diesis in chiave, nota alterata fa) nella notazione classica tutti i fa presenti nello spartito sono considerati aumentati di un semitono, a meno di ulteriori alterazioni. Invece in MusicXML tutte le note fa avranno sempre riportata l'alterazione presente nell'armatura di chiave relegando quest'ultima ad un puro aspetto grafico. Il campo *duration*, come già accennato, è in relazione con il campo *divisions* presente nell'*Intestazione del pentagramma* e rappresenta la durata della nota espressa come numero di *divisions*.

3.2 MX

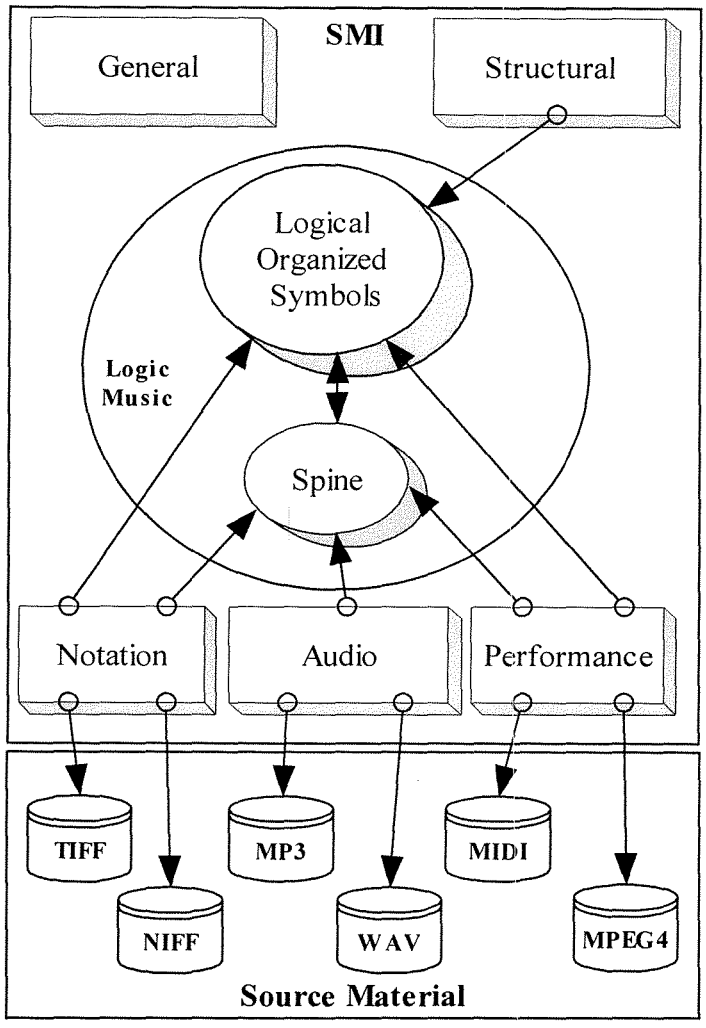
MX è un linguaggio derivato da XML tramite appositi DTD che si prefigge il non facile obbiettivo di rappresentare l'informazione musicale nella sua interezza andando quindi ad abbattere le frontiere che normalmente separano formati quali MIDI e NIFF. Entrambi sono dedicati alla rappresentazione della musica ma ognuno si occupa di un settore specifico senza fornirne una visione strutturata e completa. Lo scopo di MX è proprio questo. Fornire un linguaggio in grado di rappresentare la musica strutturandone i contenuti attraverso diversi strati detti *layers*. Ogni *layers* rappresenta un grado di astrazione diverso per l'informazione musicale.

In MX sono stati individuati sei diversi *layers* per descrivere l'informazione musicale nella sua interezza:

1. general
2. music logic
    a. spine
    b. los (logically organized symbols)
    c. layout
3. structural
4. notation
5. performance
6. audio

La struttura fondamentale di MX è lo spine. Lo spine può essere visto come una funzione di mappatura bidimensionale tra i domini dello spazio e del tempo, e rappresenta in pratica un collante tra i diversi strati. Esso si compone di eventi, ciascuno dei quali presenta un riferimento nel dominio dello spazio e del tempo. Gli strati structural, notation, performance e audio / video sono dei contenitori nei quali sono inseriti i vari formati normalmente utilizzati per la rappresentazione di questi aspetti. Ad esempio MP3 e WAV per l'audio, MIDI e MPEG per la performance, TIFF e JPG per il notation e così via. Il compito dello spine è quindi di fornire un legame certo che permetta di navigare l'informazione musicale nella sua interezza tessendo riferimenti che taglino verticalmente ogni livello di questa struttura. L'unico layer che non è legato dirattemente allo spine è il layer general perchè questa parte del formato non ha informazioni che necessitino di collegamenti temporali contenendo solo informazioni di carattere generale quali il titolo del brano, il nome del compositore e così via. Nella figura 3.2 è presentato uno schema grafico della struttura di MX. Si può notare come tutti i layer hanno un riferimento diretto allo spine generando una sorta di struttura a stella, ad eccezione del layer general.
Un brano musicale può dunque essere descritto nella sua interezza (oggetti musicali, scansioni della partitura, esecuzioni sintetizzate, registrazioni e quant'altro), e questo porta indubbi benefici, in quanto costringe a una sua visione strutturata portando degli indubbi benefici nel campo dell'analisi e dell'organizzazione dell'informazione.

Figura 3.2 lo schema di riferimento della struttura di MX[32]



Da un punto di vista tecnico, ogni layer rappresenta un elemento figlio del tag radice e dunque un generico file sarà strutturato nel seguente modo:

```
<mx>
        <general>...</general>
        <logic>...</logic>
        <structural>...</structural>
        ...
</mx>
```

Solo l'identificazione generale (*general layer*) e i dati sulla "logica" (*music logic layer*) del brano devono essere necessariamente presenti. Per quanto concerne gli altri layer, essi o sono presenti o non lo sono; la loro cardinalità non è mai superiore a 1.

3.2.1 MX – Music Logic Layer

Il layer logic presenta tre sottolivelli (elementi figli)

1. *spine* contiene la funzione di mappatura spazio-temporale
2. *los* descrive gli oggetti in partitura
3. *layout* si occupa di rappresentare il layout del brano

Chiaramente, spine e los sono fondamentali per un file MX: la loro presenza è dunque richiesta.

In particolare, *los* contiene le informazioni ricavabili dalla partitura intesa come insieme di oggetti musicali (note, pause, segni di articolazione,...), mentre ignora gli aspetti di layout (margini, impaginazione,...) cui è dedicato l'elemento *layout*.
Il layer *spine* è fondamentale data la natura multi-livello dell'MX: senza lo spine, che funge da collante tra i vari livelli di rappresentazione, la codifica MX stessa non avrebbe significato.

In questo esempio è riportato uno spezzone di un generico spine.

```
<mx>
   <general>
         ...
   </general>
   <logic>
         <spine>
               <event id="evento0" timing="NULL" hpos="NULL"/>
               <event id="evento1" timing="0" hpos="6"/>
               <event id="evento2" timing="1000" hpos="0"/>
               <event id="evento3" timing="0" hpos="0"/>
               <event id="evento4" timing="2000" hpos="6"/>
         </spine>
         ...
   </logic>
   ...
</mx>
```

L'attributo *id* assegna un identificativo univoco all'oggetto musicale (*event*) : *id* duplicati costituiscono errore e il file MX non risulta valido. Il tipo e la dimensione dell'oggetto musicale non è prefissata. Il campo di elementi da contrassegnare può essere arbitrariamente esteso. Ad esempio, segnature di chiave e di tempo, armature di

chiave e cambiamenti di tonalità, segni di agonica e di articolazione e via dicendo possono essere ritenuti significativi e dunque aggiunti allo spine. In modo analogo, lo spine può anche essere arbitrariamente ridotto. Se di ogni pezzo interessano solo alcune note (ad esempio le prime di ogni battuta), o solo il testo cantato e via dicendo, lo spine non dovrà necessariamente includere le informazioni ritenute inutili.

L'attributo *timing* costituisce una temporizzazione virtuale in *VTU* (*virtual timing units*). L'utente è libero di scegliere a quale ordine di grandezza temporale (millisecondi, decimi di secondo e così via) collegare il valore di *VTU* . Il significato di questa grandezza è molto simile al concetto di *duration* precedentemente esposto in MusicXML. Sostanzialmente il valore della singola suddivisione del tempo in partitura viene scomposto nel numero di VTU scelto quindi ogni evento riporterà il numero di VTU trascorse dall'inizio dell'evento precedente al momento del suo inizio. Un discorso analogo a quello fatto per il *timing* vale per l'*hpos*, che però tiene conto della spazializzazione orizzontale (sempre virtuale) anziché della temporizzazione. Entrambi gli attributi *timing* e *hpos* supportano valori interi positivi e il valore speciale NULL, che va inserito quando non ha senso parlare di temporizzazione o di spazializzazione. Ad esempio, la temporizzazione dell'armatura di chiave non è in generale determinabile né significativa!. La contemporaneità spaziale e/o temporale viene denotata dal valore 0. Nell'esempio sopra riportato, evento0 non ha spazializzazione né temporizzazione; evento1 occorre nell'istante 0, evento2 dopo 1000 VTU, evento3 è contemporaneo a evento2, ed evento4 dopo 2000 VTU rispetto a evento2 ed evento3. Un discorso analogo vale per lo spazio.

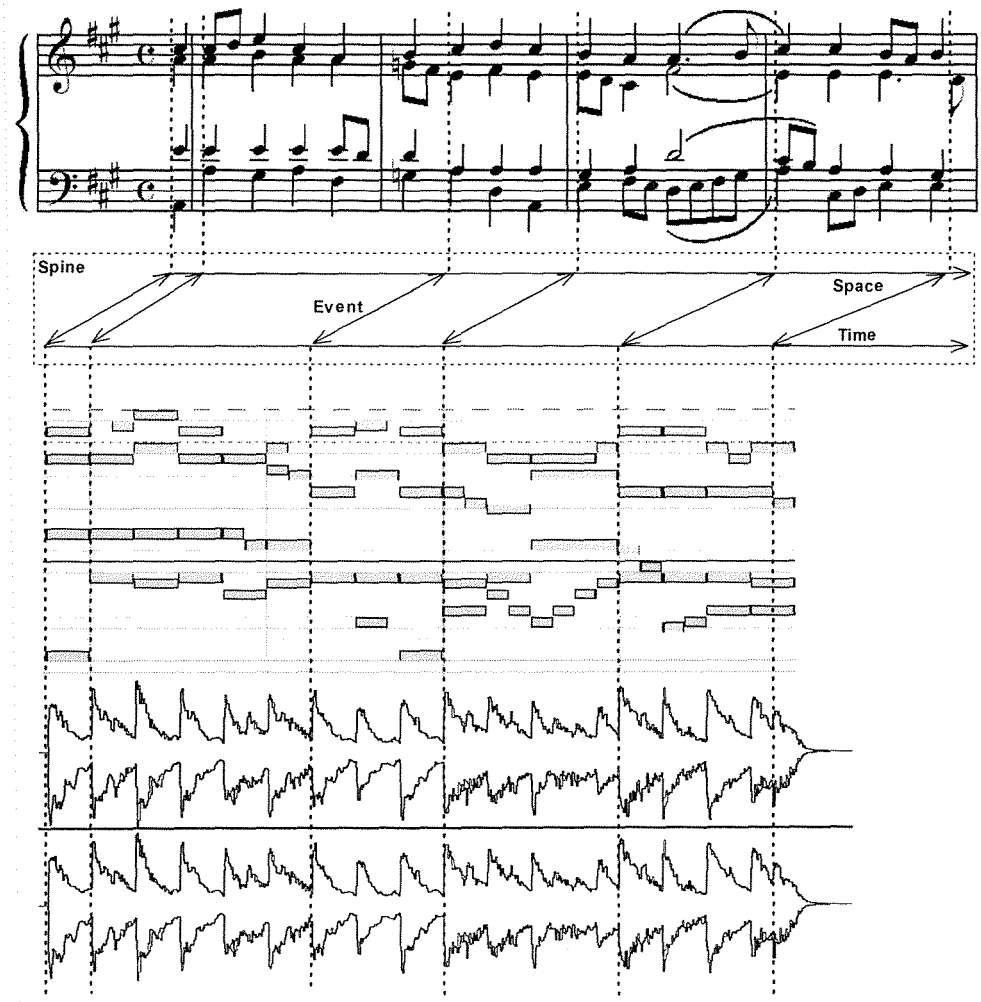
3.3 Scelta dei formati

MX è un linguaggio estremamente potente, versatile e innovativo ma è anche un formato ancora in corso di standardizzazione da questo consegue che il numero di partiture disponibili in questo formato è ancora molto limitato ed inoltre che alcune sue caratteristiche potrebbero subire delle variazioni anche molto significative nel corso del suo sviluppo finale. Questa everntualità renderebbe quindi necessaria la riscrittura di eventuali metodi per adattarli alle modifiche apportate nella versione finale.

Si è scelto quindi di utilizzare un altro formato derivato da XML, MusicXML. Questa scelta è stata fatta per ottenere dei metodi e degli algoritmi "facilmente" adattabili ad MX senza sacrificare la quantità di partiture effettivamente disponibili per il suo utilizzo.

La scelta di predisporre da subito il software per permettere l'aggiunta di un modulo dedicato alla scrittura di file MX appare subito chiara dalla natura stessa di MX, ad oggi l'unico formato che permetta di strutturare l'informazione musicale e di contenere al suo interno sia informazioni di partitura che audio. Come si può vedere dalla figura 3.3 questo linguaggio si presta perfettamente come formato base per un eventuale player che metta a disposizione dell'utente la possibilità di navigare coerentemente sia l'esecuzione di un brano che la sua eventuale partitura.

Figura 3.3 Relazioni tra lo spine e i layers audio, notazionale e perfrmance [32]



Capitolo 4

M.I.R.

Il MIR (*music information retrieval*) è una nuova area di ricerca scientifica con una forte connotazione multidisciplinare [39]. Oggi il punto di riferimento per questa disciplina che raccoglie esperti provenienti dalle aree scientifiche più disparate è la bibliografia elettronica, liberamente consultabile tramite Internet, Music-ir.org [38]. Questa ampia biblioteca virtuale raccoglie articoli su tutte le problematiche relative al MIR quali: la ricerca di informazioni musicali nei brani audio[41], la rappresentazione simbolica della musica in formati adatti al computer [42][43][45], la gestione di basi dati specifiche per la musica [40] e così via.

In questo capitolo analizzeremo una parte delle problematiche affrontate dal MIR relative alla *comprensione automatica* dei file audio, precisamente:

- *Beat Tracking [45][46][47][48][49]*
- *Pitch Tracking [50][51][52][53][54][55][72]*
- *Score Extraction [9][10][56][57][58][4]*
- *Automatic Music Synchronization [3][8][59][60]*

Per ognuno di questi ambiti saranno presentati vari approcci relativi sia ai formati compressi sia ai formati non compressi e alcune considerazioni sui risultati raggiunti. Per tematiche quali: il *riconoscimento timbri*, partire da un file audio e comprendere quali e quanti strumenti siano effettivamente presenti, la *separazione automatica dei canali* [61], partire da un file audio codificato, ad esempio, con 2 canali stereo e portarlo ad una codifica a 5 canali, il *riverse mixing* [62] , separare le tracce degli strumenti che sono presenti in un brano audio avendo a disposizioni solo la traccia audio successiva ad un operazione di missaggio, e le nuove frontiere di *genre extraction* [63], estrazione automatica del genere musicale di un brano, si rimanda alle note bibliografiche citate.

4.1 Beat Tracking

Lo scopo degli algoritmi che eseguono Beat Tracking è quello di comprendere in maniera automatica la velocità metronomica del brano e la disposizione degli impulsi relativi non solo al tipo di divisioni all'interno della battuta ma anche ai suoi sotto multipli. Questo tipo di programmi trovano un'applicazione diretta in problematiche quali la computer graphic guidata automaticamente dal tempo del brano o il controllo automatico di luci ed effetti luminosi in genere. Indirettamente questi algoritmi possono essere utilizzati per effettuare una sincronizzazione generica tra audio e partitura oltre ad avere una certa utilità nella gestione di basi di dati musicali. In questa sezione sono analizzati 3 diversi approcci per risolvere il problema del beat tracking[47][49][45]. I primi due algoritmi sono adatti per formati PCM, mentre il terzo è studiato appositamente per MP3. Tutti gli algoritmi descritti necessitano esclusivamente il file audio e non formulano alcuna ipotesi sul genere, la

partitura o altro; sono applicabili indistintamente su ogni file audio (rispettivamente PCM e MP3) indipendentemente dalla presenza di strumenti percussivi ma hanno delle limitazioni sulla velocità metronomica, sulla segnatura del tempo o altro. Per ogni algoritmo saranno specificate le eventuali limitazioni.

4.1.1 Beat Tracking PCM

Il primo approccio presentato è stato sviluppato da Masataka Goto [47], in figura 4.1 è schematizzato il funzionamento di questo algoritmo.

Questo algoritmo ha la peculiarità di lavorare in real-time e di tracciare non solo la figurazione base del tempo (il quarto) ma anche i 2/4 e l'inizio di ogni misura. Ha una forte limitazione sulla segnatura di tempo (solo 4/4) e sulle velocità metronomiche, sempre comprese tra 61 e 185 beat al minuto per i brani in cui non si ha una batteria a scandire il tempo, e tra 61 e 120 beat al minuto in assenza di batteria. Questo algoritmo è diviso in 2 fasi: inizialmente si estraggono dal file audio degli elementi musicali in cui è lecito supporre ci sia un beat successivamente si interpretano e selezionano i risultati in modo da fornire l'output corretto.

Gli elementi musicali considerati sono:

- Istanti di inizio di un qualsiasi evento musicale (*onset time*)
- Cambio degli accordi (*chord changes*)
- Pattern di batteria (*drum patterns*)

Inizialmente viene calcolato lo spettro del segnale tramite una FFT a 1024 punti con una finestra di Hanning.[71]

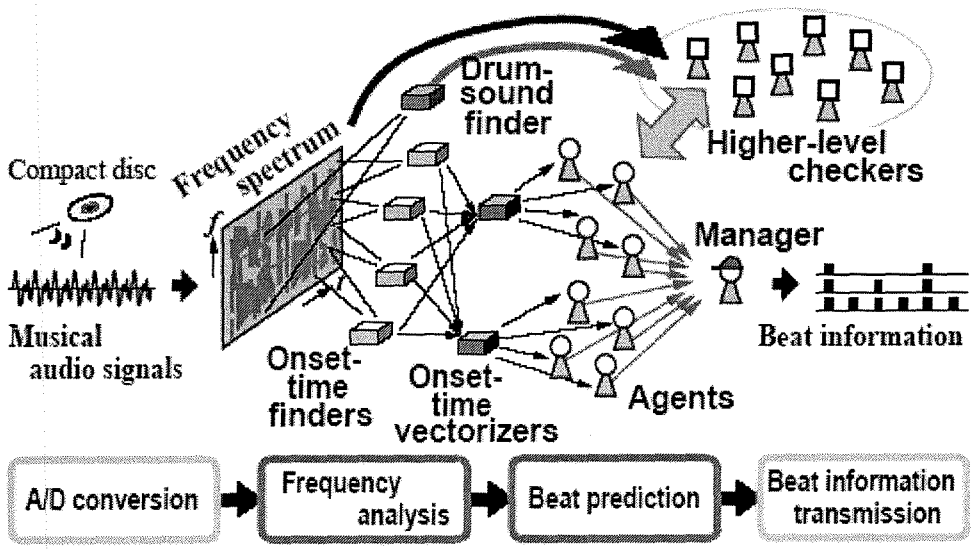
Lo spettro ottenuto è poi ridiviso in 7 bande critiche, per ogni banda critica viene calcolata la rapidità dell'incremento della potenza del segnale e la potenza presente nelle bande critiche adiacenti. Quando il segnale manifesta delle forti variazioni tra queste grandezze l'istante di tempo considerato è inserito in una lista che rappresenta gli *onset times* del segnale. Successivamente è applicato un algoritmo che analizza il segnale generando una serie di punti detti *previsional beat times* che rappresentano i potenziali istanti in cui è presente un beat.

Il cambio degli accordi è calcolato utilizzando i *provisional beat times* trovati. Il sistema riconsidera lo spettro del segnale nei punti di interesse (i *provisional beat times*) ed

in questi punti è calcolata la frequenza dominante che è successivamente messa a confronto con quella del punto di interesse successivo, se si osservano grosse differenze questo punto viene considerato un *chord changes*.

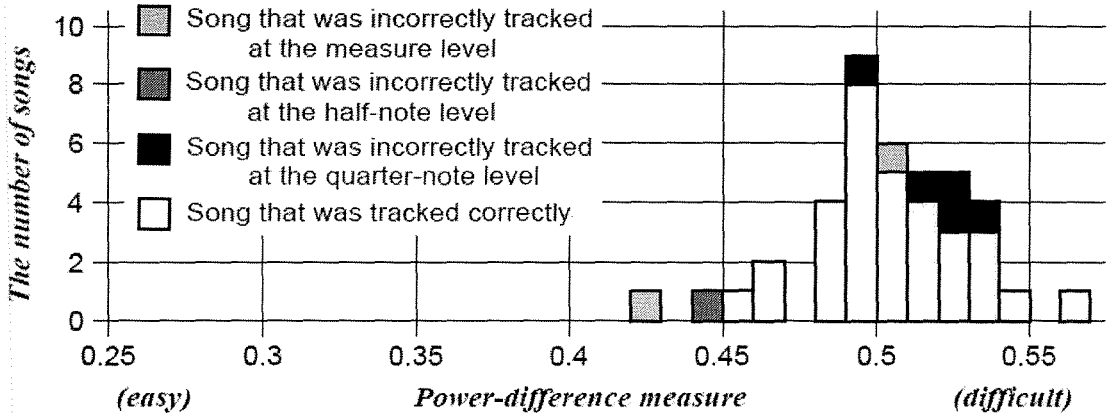
I pattern di batteria sono ristretti a cassa e rullante. La prima è calcolata partendo sempre dai dati ricavati precedentemente, mentre il secondo è calcolato attraverso un noise detector. La seconda fase dell'algoritmo è gestita da vari agenti che selezionano i risultati, con regole personalizzate e forniscono varie possibili strutture ritmiche ad un manager. Questa figura ha il compito di selezionare la strategia migliore fornendo l'output vero e proprio. Come si vede dalla figura 4.1 la presenza della batteria aiuta l'algoritmo fornendo altre informazioni su cui generare ipotesi, ma non è indispensabile.

Figura 4.1 schema dell'algoritmo presentato in [47]

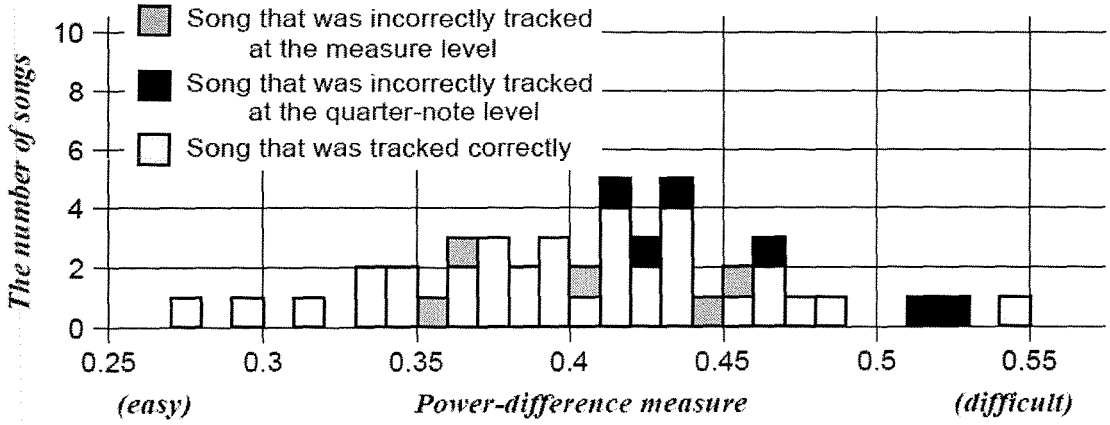


Nella figura 4.2 sono presentati i risultati ottenuti da questo algoritmo. L'istogramma in basso si riferisce a brani con batteria, mentre il secondo istogramma fa riferimento a brani che ne sono privi. Gli istogrammi hanno sulle ordinate il numero di brani in cui i beat sono stati tracciati correttamente e i brani su cui sono stati rilevati errori nei singoli tipi di beat precisamente l'inizio delle misure, le note da 2/4 e le note da 1/4. Le ordinate degli istogrammi rappresentano il *Power-difference measure*. Questo indice rappresenta la complessità ritmica del brano considerato: assume valori tra 0 e 1 ed è calcolato mettendo in relazione la potenza del brano nel punto in cui si ha un beat e la potenza massima raggiunta dal segnale tra 2 beat distinti.

Figura 4.2 Risultati dell'Algoritmo [47]



(a) Histogram for 40 songs without drum-sounds.

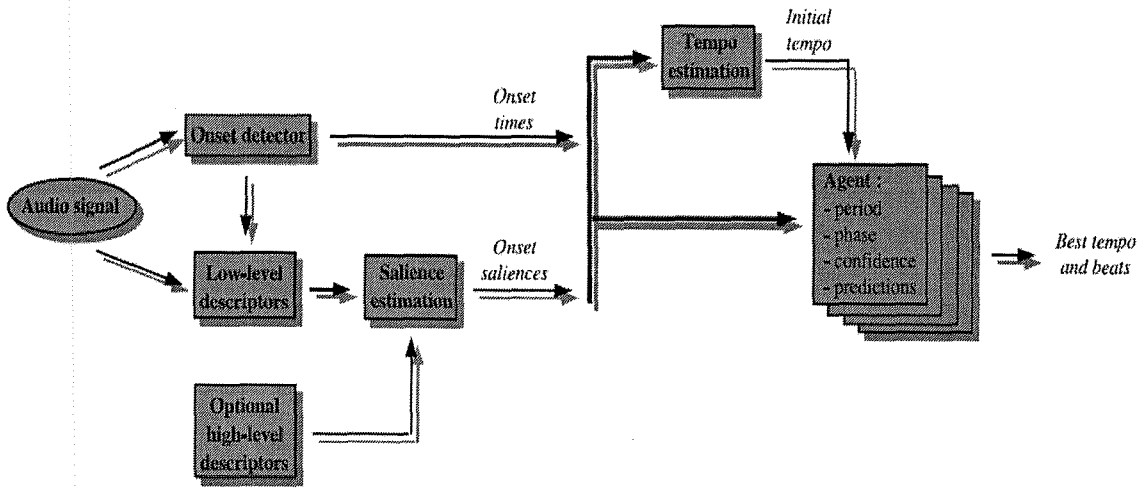


(b) Histogram for 45 songs with drum-sounds.

Questo algoritmo grazie al suo approccio real-time è molto utile ad esempio per la gestione automatizzata delle luci su di un palco oppure per "guidare" delle animazioni tridimensionali che devono muoversi a ritmo di musica, purtroppo i risultati non sono abbastanza robusti da essere utilizzati come sincronizzazione tra un'eventuale partitura e la sua corrispondente esecuzione.

Il secondo algoritmo analizzato per il Beat Tracking su formato PCM è stato proposto da Nicolas Scaringella e Giorgio Zoia [49].

Figura 4.3 schema a blocchi dell'algoritmo presentato in [49]



Questo algoritmo è strutturato in 2 fasi. La prima esegue un'analisi di basso livello sul segnale calcolandone lo sviluppo dell'energia e applicandovi un algoritmo di peak-detection. La fase successiva è demandata ad un "manager degli agenti". Questo manager ha il compito di creare, distruggere gli agenti e di valutarne e aggiornarne il grado di affidabilità. Gli agenti hanno il compito di fornire ognuno una possibile predizione sulla posizione del prossimo beat. Ogni agente mantiene, al suo interno, uno *stato* e una *storia*. Lo stato è composto: dall'istante di tempo che l'agente sta esaminando, dal suo grado di affidabilità e dalla sua predizione sulla posizione del prossimo beat. La storia di un agente è invece la lista di tutti i beat che ha predetto correttamente. Un agente è cancellato dal manager, perchè è considerato palesemente in errore, se le sue previsioni appaiono poco probabili. Quando un agente arriva ad un grado di affidabilità molto elevato il manager assume la sua struttura dei beat come la posizione reale dei beat stessi all'interno del brano. Prima della fine del processo l'agente considerato nel giusto può essere sostituito da un altro nel caso in cui quest'ultimo abbia un'affidabilità molto superiore al precedente. Questo permette al sistema di fare beat tracking anche su brani con una ritmica non costante.

Questo algoritmo ha mostrato un'affidabilità molto bassa su brani di musica classica e lirica offrendo un'attendibilità dei risultati pari a circa il 45%. Oltre un beat su 2 non è stato identificato nella posizione corretta. Nella tabella successiva è presentato il risultato

della sperimentazione di questo algoritmo. Il parametro ρ indica la percentuale di beat correttamente individuati dall'algoritmo.

Figura 4.6 Risultati completi dell'algoritmo [49]

Genre	Songs	ρ
Complete corpus	90	72.52%
Popular	3	91.66%
Ballads	3	96.69%
Rock	3	96.64%
Heavy-metal	3	73.66%
Rap/Hip-hop	3	98.55%
House	3	95.27%
Techno	3	89.52%
Funk	3	97.16%
Soul/R&B	3	99.05%
Big band	3	86.87%
Modern jazz	3	26.37%
Fusion	3	87.59%
Bossa nova	3	53.18%
Samba	3	100.00%
Reggae	3	73.60%
Tango	3	31.33%
Baroque	4	53.03%
Classic	3	25.21%
Romantic	3	51.53%
Modern	2	36.68%
Brass band	3	56.56%
Blues	3	83.28%
Folk	3	48.40%
Country	3	75.44%
Gospel	3	85.19%
African	2	17.28%
Indian	2	66.79%
Flamenco	3	59.29%
Chanson	1	28.41%
Traditional Japanese	3	95.22%
Japanese Folk Min'you	2	64.37%
Ancient Jap. Gagaku	1	39.08%
A cappella	1	22.46%

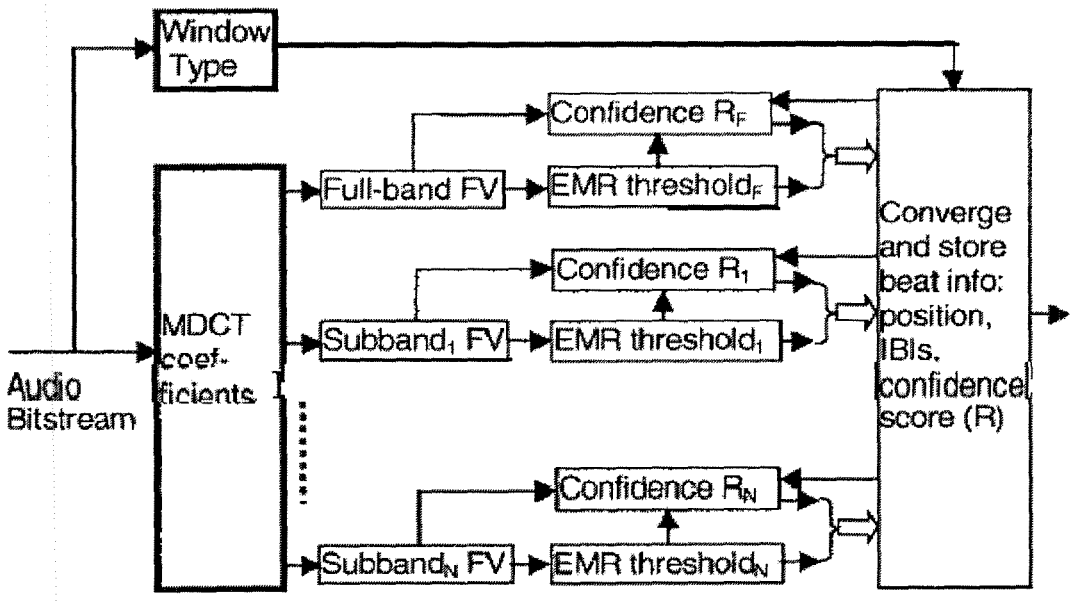
Come si osserva dalla figura 4.6 il sistema ha un indice di affidabilità pari al 72,52% nell'insieme dei brani analizzati è da notare, però come le prestazioni in generale non siano ugualmente affidabili tra i vari generi musicali oscillando da un minimo di 17,28% fino al 100% in funzione dei vari generi analizzati.

4.1.2 Beat Tracking MP3

L'algoritmo analizzato[45] in questa sezione è il solo presente in letteratura ad effettuare Beat Tracking direttamente sul formato MP3. È da notare che questo algoritmo è stato sviluppato con la precisa finalità di tracciare i beat di un file MP3 per poter compiere un recupero delle informazioni perse durante una trasmissione di questo brano su un canale con forti disturbi, quindi il sistema non nasce con precise finalità di Beat Tracking. La scelta di utilizzare un sistema simile ad un Beat Tracker nasce dalla considerazione che il miglior frame MP3 per sostituirne uno andato perso è un frame che abbia la stessa posizione rispetto al tempo della partitura. Per selezionare questo frame è ovviamente necessario conoscere la disposizione dei beat. Questo algoritmo è stato studiato appositamente per canzoni che abbiano una sezione ritmica molto semplice con dei bassi prominenti rispetto agli altri strumenti. Per ammissione degli stessi autori è inapplicabile su generi con sezioni ritmiche molto complesse o su generi quali la musica classica.

Questo algoritmo è composto da 2 analisi distinte di basso livello e da una successiva fase di valutazione dei risultati che ha il compito di selezionare i beat reali tra tutti i possibili beat candidati. Nella figura 4.7 è presentato lo schema a blocchi delle due fasi di analisi di basso livello: la prima relativa al solo *Window Type* e la seconda basata sui coefficienti *MDCT* dello spettro MP3.

Figura 4.7 Schema a blocchi dell'algoritmo proposto in [45]



Il blocco *Window Type* presente in figura 4.7 ha il solo scopo di estrarre dal file MP3 la posizione dei granuli che presentano una finestra *Short*. Poiché questo tipo di finestra è utilizzato in presenza di forti rumori impulsivi, il suo impiego nella codifica di un brano MP3 fa inserire automaticamente nella lista dei possibili candidati il rispettivo granulo.

L'analisi dei coefficienti MDCT è invece più complessa. I 576 valori che compongono lo spettro MDCT sono raggruppati in 6 vettori chiamati *FV(feature vector)*. L'accorpamento dei coefficienti MDCT non è lineare, cioè i vettori considerati hanno lunghezze diverse che aumentano in funzione dell'altezza della frequenza considerata. In figura 4.8 è presentata la tabella che specifica come sono aggregati i coefficienti MDCT all'interno dei 6 *FV*. Per ogni vettore è calcolata l'energia della parte di spettro in esso considerato.

Figura 4.8 Parametri di aggregazione dei coefficienti MDCT nei vettori FV [45]

Sub-band	Frequency interval (Hz)	Index of MDCT coefficients	Scale factor band index
1	0-459	0-11	0-2
2	460-918	12-23	3-5
3	919-1337	24-35	6-7
4	1338-3404	36-89	8-12
5	3405-7462	90-195	13-16
6	7463-22050	196-575	17-21

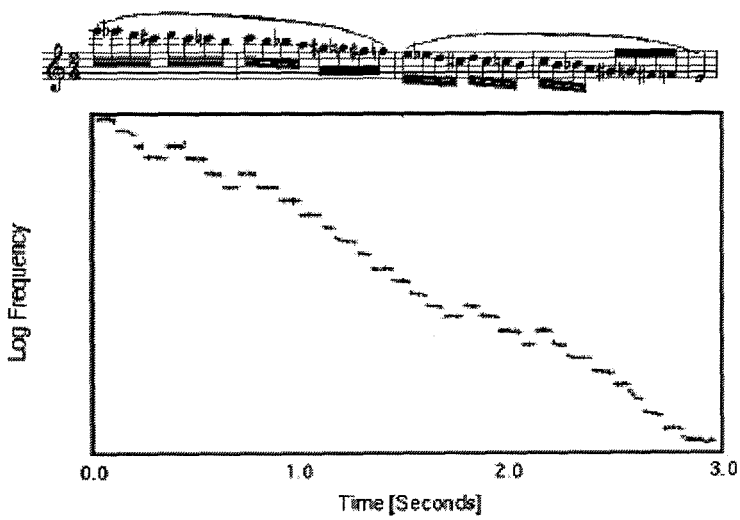
Le energie risultanti sono salvate in una matrice con 6 righe (quanti i FV) e un numero di colonne pari ai granuli che compongono il brano. È quindi selezionata una finestra di ricerca con ampiezza variabile. Questa finestra di ricerca si muove su ogni singola riga coprendo tutti i granuli del brano. All'interno di ogni finestra di ricerca si esegue un'operazione di sogliatura che ha lo scopo di trovare i possibili candidati. Tramite un modello statistico si valuta il grado di affidabilità dei possibili candidati presenti in ogni sottobanda (vettori *FV*). L'ultimo blocco dell'algoritmo prende in input i migliori candidati del blocco precedente e i risultati dell'analisi dei block type e quindi seleziona i candidati migliori e fornisce l'output finale. La fase sperimentale ha confermato le ipotesi iniziali.

Sono stati considerati 6 brani. 4 sono stati tracciati senza errori, nei rimanenti, l'algoritmo è sostanzialmente fallito. Questo fallimento è imputabile ad un'implicita condizione imposta dall'algoritmo: i beat devono avere un valore energetico sempre maggiore dei non beat, se questa condizione non è verificata, ad esempio in brani di musica classica e più in generale in pezzi con sezioni ritmiche atipiche, l'algoritmo semplicemente fallisce.

4.2 Pitch Tracking

Un algoritmo di Pitch Tracking ha lo scopo di fornire un output contenente i nomi delle note presenti in un brano in relazione al tempo, ma senza preoccuparsi di inserirle in un contesto di partitura e quindi senza la necessità di trasformare le durate temporali assolute dedotte dal file audio in durate temporali relative adatte alla partitura. Un semplice esempio di output di un algoritmo di Pitch Tracking è fornito in figura 4.9. Lo spartito in alto è un semplice riferimento grafico; non devono conoscerlo e non devono generarne uno. Sulle ordinate del grafico sono inseriti i tempi in secondi, mentre sulle ascisse sono riportate le frequenze fondamentali delle note trovate.

Figura 4.9 Output di un algoritmo di Pitch Tracking



La problematica del pitch tracking è stata storicamente affrontata partendo da vari punti di vista in relazione alle successive applicazioni. Possono essere distinti algoritmi che fanno Pitch Tracking su file monofonici e che quindi devono individuare solo una nota per ogni istante di tempo, algoritmi che si occupano di file polifonici e una classe di algoritmi che devono svolgere questo compito in real-time. Il problema del pitch tracking su file monofonici può considerarsi ormai risolto, molti sono i programmi anche commerciali che riescono bene a portare a termine questo compito [9][10], viceversa il pitch tracking su file polifonici è ancora lontano dal raggiungere una soluzione realmente competitiva. Nel capitolo successivo ci soffermeremo su alcuni metodi comunemente usati dagli algoritmi di Pitch Tracking per selezionare le frequenze candidate che genereranno poi il risultato finale. Particolare interesse rivestono gli algoritmi di pitch tracking real-time che trovano un'applicazione diretta per risolvere problematiche legate ad esecuzioni dal vivo di brani musicali. Sono, ad esempio, in grado di risolvere errori legati ad una cattiva accordatura o ad errori esecutivi. È da notare che non sono ancora stati sviluppati algoritmi di pitch tracking adatti al formato MP3 dunque tutti i metodi analizzati sono rivolti esclusivamente a codifiche non compresse.

4.2.1 Algoritmi di estrazione del pitch

In questa sezione sono esaminati i seguenti algoritmi di estrazione del pitch :

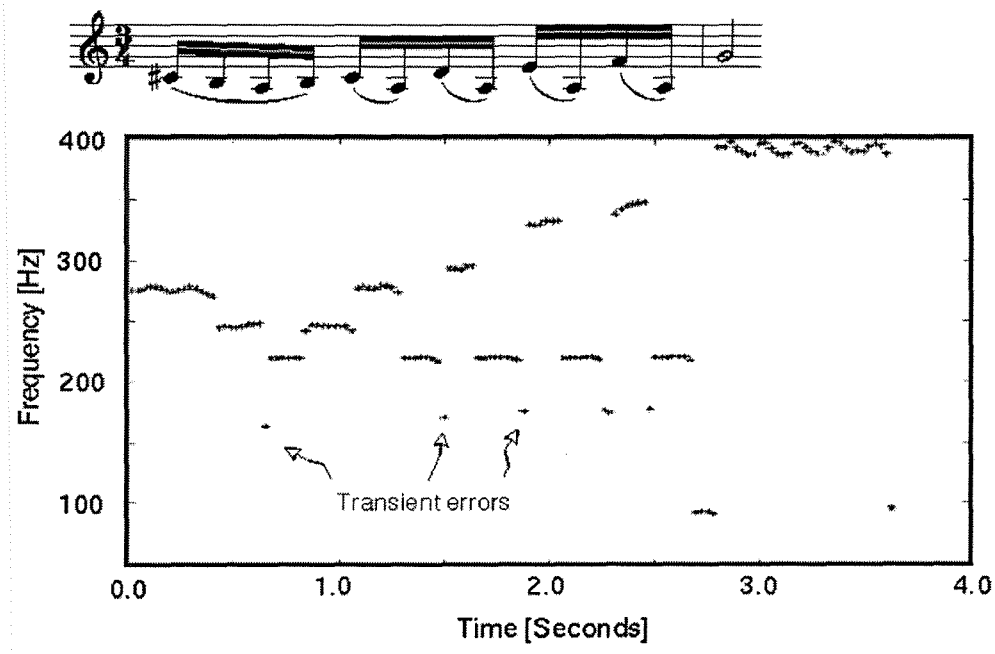
- Harmonic Product Spectrum [51]
- Maximum Likelihood [51]
- Cepstrum-Biased HPS [52]
- Weighted Autocorrelation Function [53]

Tutti gli algoritmi citati offrono dei risultati validi su file monofonici. Il Cepstrum-Biased HPS è l'unico che permette di avere dei risultati abbastanza validi su file polifonici [50].

In applicazioni real-time i risultati di questi algoritmi non subiscono particolari controlli e il loro output è considerato corretto, viceversa, in ambiti non real-time si aggiunge un successivo strato che ha il compito di esaminare i risultati trovati ed eliminare eventuali errori generati sfruttando conoscenze più o meno complesse di armonia ed acustica. In figura 4.10 è riportato un tipico caso di errore che può essere facilmente risolto da un algoritmo di verifica dei risultati. Il brano esaminato nell'esempio in figura 4.10 (di cui è riportato lo

spartito per maggiore chiarezza) è una piccola parte di violoncello registrato in presa diretta. Come nella figura 4.9 sulle ordinate sono riportati i secondi relativi al brano mentre sulle ascisse le frequenze individuate dagli algoritmi di estrazione del pitch. Le frecce indicano i punti in cui si sono verificati degli *errori di transiente*. Questa tipologia di errore è molto comune. Si verifica in presenza di note che abbiano un forte attacco. La fase di attacco di molti strumenti, infatti, non ha subito una frequenza fondamentale ma genera uno spettro che contiene molte componenti frequenziali di conseguenza questa situazione di instabilità porta l'algoritmo di estrazione del pitch a commettere errori.

Figura 4.10 analisi del pitch di una parte di violoncello con *errori di transiente* [50]



4.2.1.1 Harmonic Product Spectrum

Questo algoritmo è stato proposto da Noll nel 1969 [51] è il metodo più semplice per implementare un algoritmo di estrazione del pitch e funziona abbastanza bene in un'ampia varietà di casi. La prima fase di questo algoritmo prevede l'amplificazione delle basse frequenze e successivamente l'applicazione di una trasformata da tempo a frequenza.

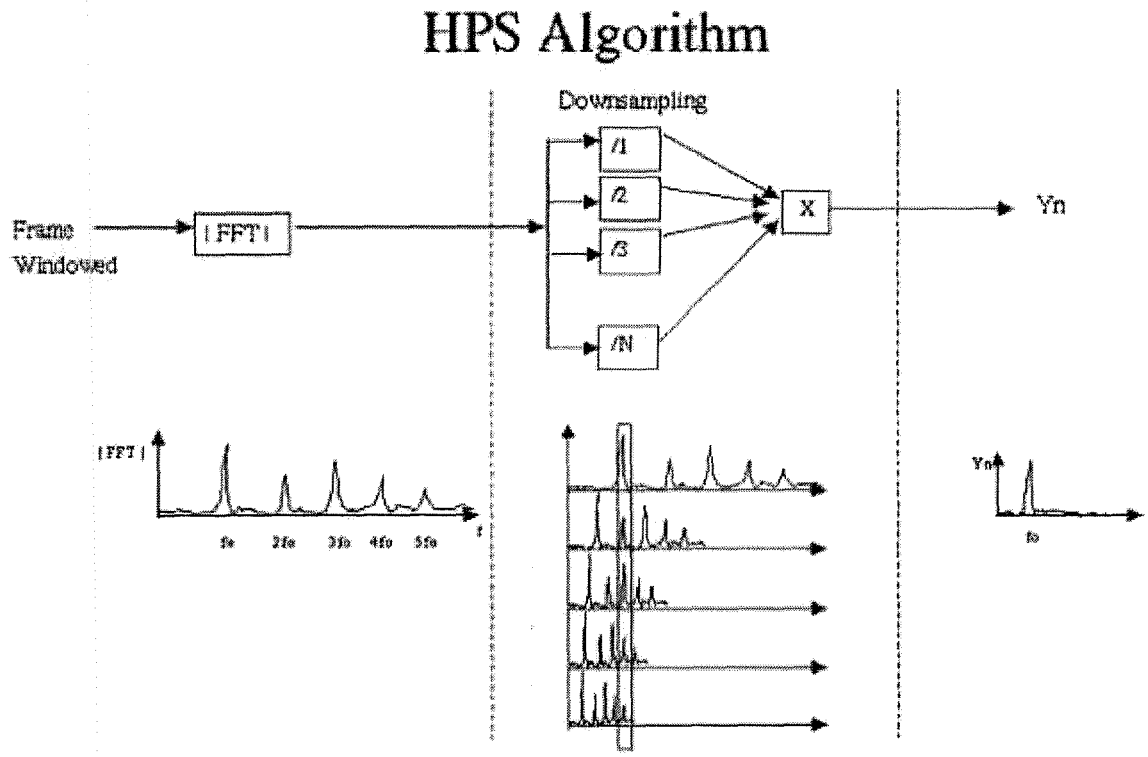
L'algoritmo HPS misura la massima coincidenza delle armoniche secondo l'equazione (4.1) per ogni frame dello spettro $X(\omega)$.

$$Y(\omega) = \prod_{r=1}^{R} |X(\omega r)| \qquad (4.1)$$

Dove R è il numero delle armoniche considerate, ω la frequenza fondamentale presa in esame e $X(\omega r)$ è il downsampling delle spettro di un fattore r [72].

Una semplice operazione di estrazione del massimo all'interno dello spettro $Y(\omega)$ risultante da (4.1) restituisce il pitch richiesto. Molto spesso questo algoritmo seleziona pitch trasposti di un'ottava verso l'alto. Questo errore è però facilmente recuperato applicando una semplice condizione che forza l'algoritmo a scegliere un pitch di un'ottava più in basso rispetto al massimo se il secondo valore massimo trovato è posto alla metà della frequenza del picco inizialmente scelto. A causa del rumore di fondo le implementazioni pratiche di questo algoritmo escludono automaticamente dalla ricerca tutti i pitch sotto i 50 Hz.

Figura 4.11 Esempio di funzionamento di HPS a sinistra lo spettro FFT del segnale, in mezzo gli $X(\omega r)$ con R uguale a 5 e a destra il risultato della produttoria $Y(\omega)$.



4.2.1.2 Maximum Likelihood

L'Algoritmo di Maximum Likelihood è stato presentato da Noll nel 1969 [51]. Questo algoritmo ricerca in un insieme di possibili spettri ideali quello che meglio si allinea con lo spettro del segnale reale. Gli spettri ideali sono definiti come un treno di impulsi alla frequenza ω convoluti con lo spettro del segnale reale. Questo processo cerca di minimizzare l'errore tra lo spettro del frame analizzato e i possibili spettri candidati. La frequenza ω dello spettro ideale che minimizza l'errore è la finestra cercata. Questo metodo è molto rapido e si presta perfettamente ad applicazione di tipo real-time. Ha dei grossi limiti dovuti alla numerosità dei possibili spettri ideali, infatti, nel caso in cui la frequenza cercata non rientri tra le possibili frequenze ω dei treni di impulsi questo metodo da luogo ad errori. L'algoritmo ML si presta bene ad analizzare strumenti quali tastiere o chitarre che hanno delle frequenze ben definite ma genera errori su parti cantate e strumenti quali violini in cui le frequenze fondamentali non sono sempre uguali a quelle ideali. ML è molto sensibile al rumore di fondo e richiede di una serie di template ideali per ogni frequenza che si deve analizzare. In altre parole non permette di utilizzare un solo spettro ideale ad esempio per tutti i Do, ma necessita di un template apposito per ogni ottava, infatti, già una distanza di due armoniche tra il treno di impulsi e la nota cercata fa fallire l'algoritmo.

4.2.1.3 Cepstrum-Biased HPS

Questo algoritmo è stato proposto da Master nel 2000 [52] e prevede la fusione di due metodi distinti l'HPS e il Cepstrum per cercare di risolvere il problema dell'estrazione del pitch su file polifonici. Il primo passo per combinare questi due algoritmi è di trasportare l'operazione di Cepstrum dal dominio del tempo a quello delle frequenze. I risultati dell'operazione di Cepstrum sono quindi moltiplicati con la funzione risultante da HPS. I picchi rimanenti offrono una buona approssimazione delle frequenze presenti nel segnale perchè gli errori generati separatamente dai 2 algoritmi tendono ad annullarsi reciprocamente. Un'operazione di soglia completa l'algoritmo selezionando i picchi di reale interesse. Questo algoritmo è molto utile nelle analisi di segnali polifonici grazie alla sua robustezza rispetto a rumore ed ad eventuali errori nella selezione del pitch.

4.2.1.4 Weighted Autocorrelation Function

Questo metodo propone di utilizzare la funzione di autocorrelazione per calcolare il pitch fondamentale del segnale. È noto che un segnale periodico ha un coefficiente di autocorrelazione molto elevato quando il ritardo con cui lo si confronta con se stesso è multiplo della sua frequenza fondamentale. È lecito supporre quindi che la funzione di autocorrelazione mostri un picco quando il ritardo corrisponde ad un periodo del segnale.

Per rendere il metodo più robusto è stata aggiunta una funzione di media tra le differenze del valore assoluto tra un segnale e se stesso. Questa operazione di media si comporta diversamente rispetto all'operazione di autocorrelazione avendo una distribuzione statistica differente[53].

Considerando insieme i risultati di queste due operazioni tramite pesi differenti si ottiene un sistema molto robusto rispetto al rumore e che offre i migliori risultati quando applicato su file monofonici.

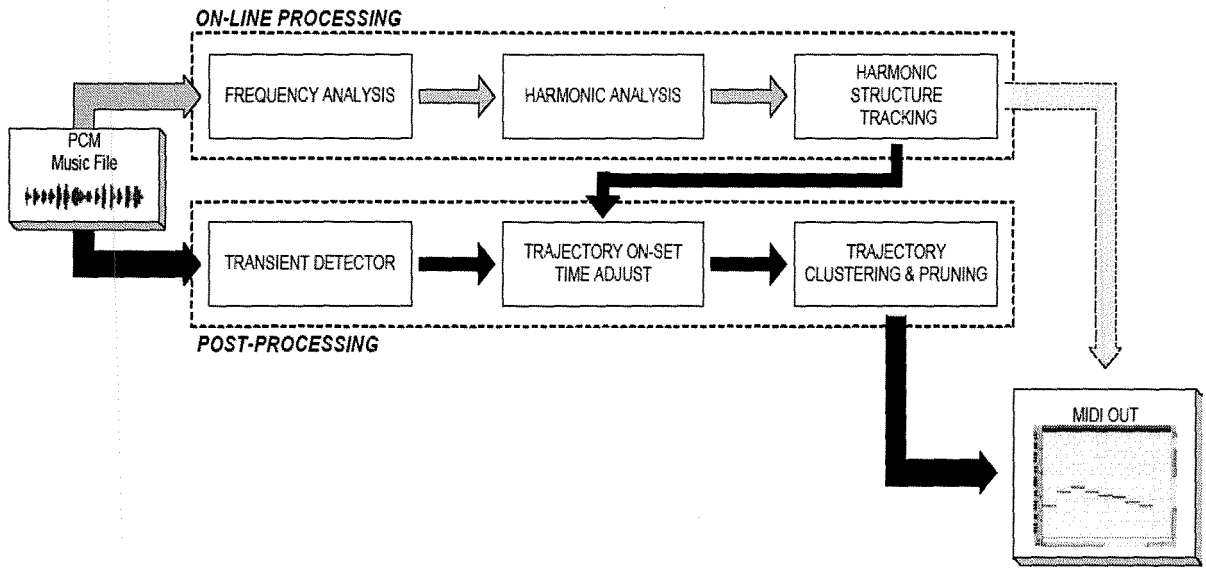
4.3 Score Extraction

L'estrazione di una partitura può essere definita come l'ascolto di un brano musicale e la scrittura delle note che lo compongono. Questo implica l'estrazione di caratteristiche specifiche come le note presenti, la loro durata, l'ottava di appartenenza, la dinamica e l'identificazione del timbro. Gli algoritmi che compiono automaticamente questa operazione hanno moltissimi ambiti applicativi tant'è che il problema dell'estrazione automatica di partiture partendo dai soli file audio è la problematica che, tra quelle precedentemente citate, ha interessato di più le aziende, è, infatti, possibile trovare molti software che permettono di effettuare questa operazione. Nessuna di queste applicazioni riesce ad eseguire una trascrizione completa ma cerca di fornire una partitura di partenza che deve poi essere ritoccata manualmente dall'utente.

La teoria alla base di questa tipologia di algoritmi è similare a quanto visto nel paragrafo 4.2. Ovviamente in questo caso passa in secondo piano la velocità esecutiva, ma diventa predominante l'affidabilità dei risultati e la loro correttezza.

In figura 4.12 è presentato lo schema a blocchi di un convertitore PCM-MIDI proposto da Martins nel 2002 [56]. Come si può notare il sistema è diviso in due parti *on-line processing* e *post-processing*. La prima parte può essere eseguita in real-time e non si discosta molto dagli algoritmi precedentemente illustrati in 4.2, la seconda parte invece effettua una successiva analisi del segnale cercando di migliorare i risultati della precedente. Questo schema generico è applicabile a tutti i software che compiono score extraction. Si ha una prima parte dedicata ad un'analisi di basso livello del file audio che estrae un numero molto elevato di possibili note; una seconda parte che ha il compito di valutare questi risultati andando a realizzare analisi più complesse che permettano di eliminare tutte le false note trovate.

Figura 4.12 Schema del convertitore proposto in [56]



Come si osserva dallo schema in figura 4.12 la fase di post processing è composta da 3 blocchi:

- *Transient detector*
- *Trajectory on-set time adjust*
- *Trajectory clustering & pruning*

Il blocco *transient detector* ha il compito di identificare le fasi di non stazionarietà all'interno del brano. Deve dunque riconoscere gli attacchi delle note presenti nel file audio (come visto nel paragrafo 4.2.1 punti critici per gli algoritmi di pitch tracking).

Lo scopo del *trajectory on-set time adjust* è quello di rifinire il lavoro fatto dal blocco precedente utilizzando le caratteristiche armoniche trovate dalla fase di *harmonic structure tracking* cercando di eliminare eventuali falsi transienti e di separare transienti erroneamente uniti.

L'ultimo blocco ha il compito di creare degli insiemi contigui nel piano temporale (la durata delle note) e nel piano armonico (invarianza del pitch); mettendo in relazione questi insiemi si cercano di eliminare tutte le false note trovate.

I risultati di questo algoritmo sono abbastanza buoni per file poco complessi. Purtroppo all'aumentare della complessità armonica e timbrica il sistema diventa sempre più instabile generando partiture anche molto lontane dalla realtà.

Questi risultati sono in linea anche con i risultati dei prodotti commerciali [9][10].

Sfortunatamente non sono disponibili degli standard per valutare la bontà di questi algoritmi, di conseguenza si ha una certa difficoltà nel giudicare i risultati raggiunti dal singolo algoritmo/programma[4]. È invece possibile avere una visione generale dei risultati raggiunti in quest'ambito.

Le tipologie di segnali audio trattabili sono sostanzialmente 4:

- Monofonici e monotimbrici
- Monofonici e politimbrici
- Polifonici e monoitimbrici
- Polifonici e politimbrici

Per la prima tipologia di segnale abbiamo una precisione molto elevata nel tracciare le relative partiture, precisione che scende nel secondo caso e che continua a degradare fino a raggiungere risultati solo semi-automatici per l'ultima tipologia di segnali.

4.4 Automatic Music Synchronization

Come già detto lo scopo di questi algoritmi è la sincronizzazione tra la partitura e la sua esecuzione. In questa sezione analizzeremo tre algoritmi proposti per i formati non compressi valutandone risultati, limiti e peculiarità. In letteratura non è disponibile alcun approccio a questa problematica per file compressi e per partiture simboliche XML.

Il primo approccio, proposto da D'Onofrio nel 1999 [3], prevede l'implementazione di un sistema in due fasi. La prima fase prevede la sincronizzazione degli inizi delle battute mentre la seconda sincronizza il tipo di divisione indicata nella segnatura di tempo.

Inizialmente il sistema crea una *pseudo-partitura* generandola direttamente dal file audio tramite un banco di filtri passa-banda. In seguito, tramite una funzione euristica, il sistema confronta la *pseudo-partitura* trovata con lo spartito reale del brano mettendo in relazione il numero massimo di inizi battute. La funzione euristica è ripetuta ricorsivamente fino a trovare tutti gli inizi delle battute completando quindi la prima fase. La seconda fase prevede di cercare gli eventi all'interno di una misura solo tra l'inizio di battuta trovato e l'inizio della battuta successiva. Questo algoritmo non compie una sincronizzazione completa di tutti gli eventi in partitura, ma sincronizza gruppi di eventi con durata a pari a quella riportata nella segnatura di tempo. La funzione euristica utilizzata per trovare gli istanti di inizio battuta impone che nei brani analizzati non siano presenti cambi di tempo superiori al 30% all'interno del brano. Nella tabella sottostante sono riportati i risultati ottenuti da questo algoritmo. Una battuta errata di 1 (rispettivamente 2) evento indica che rispetto alla sincronizzazione ideale c'è stato uno "spostamento" in avanti o indietro di una (rispettivamente 2) divisione di battuta come riportato nella segnatura di tempo dello spartito.

Questo algoritmo offre delle prestazioni efficaci per effettuare sincronizzazioni dedicate alla visualizzazione coerente di audio e partitura ma risulta di difficile applicazione come tool per la gestione di basi di dati musicali.

Una versione evoluta di questo algoritmo è stata proposta da Galbiati nel 2001[8].

L'algoritmo presentato da Galbiati prevede di valutare non più contemporaneamente le parti presenti nello spartito ma di considerarle singolarmente. Tramite un diverso algoritmo di decisione basato su una divisione in *cluster* dei risultati trovati, si opera una stima sugli attacchi di questi insiemi e sulla loro durata in modo da poterne dedurre i tempi di sincronizzazione finali anche in assenza di una partitura pedissequa e completa.

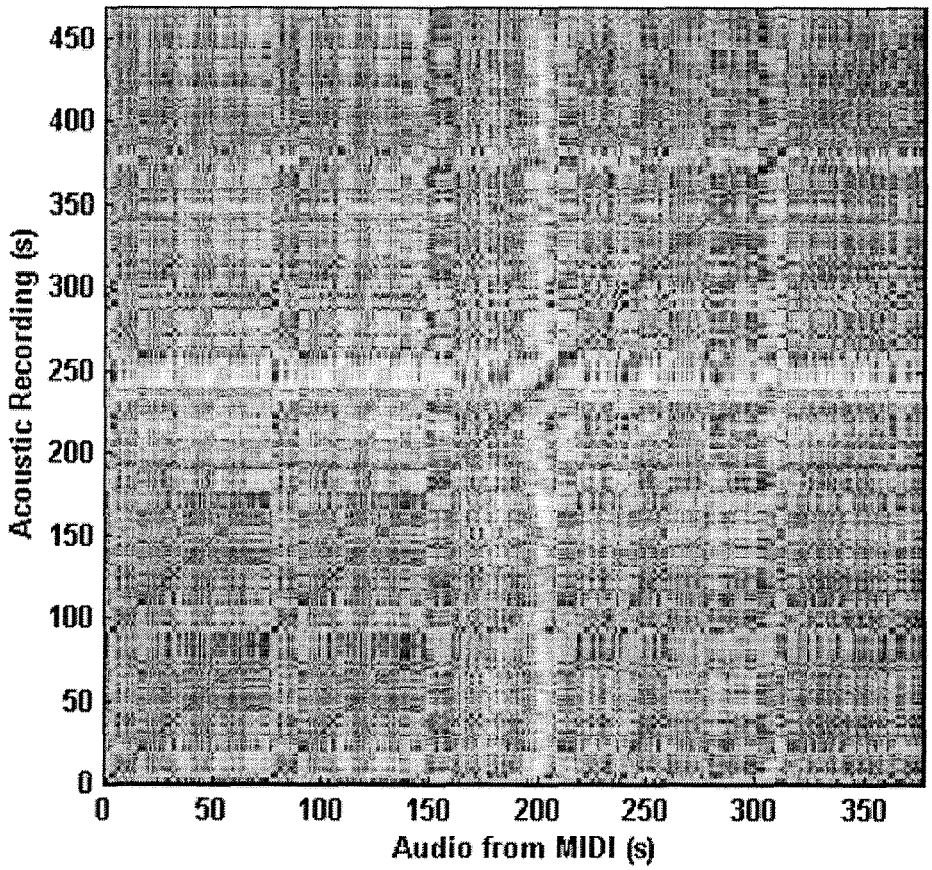
Your scan in attachment

Questo approccio è nato appunto con l'idea di superare il vincolo di un'esecuzione pedissequa della partitura nel file audio. I risultati ottenuti avendo a disposizione una partitura conforme alla reale esecuzione sono in linea e a volte superiori con quanto proposto da D'Onofrio, con partiture incomplete o con un'interpretazione molto personale l'algoritmo riesce a ben sincronizzare una buona parte degli eventi che si discostano dalla partitura. Purtroppo questa evoluzione presentata da Galbiati, lavorando su partiture non perfette, ha necessità di tutta una serie di parametri impostati dall'utente, nel caso in cui queste variabili siano impostate in maniera scorretta il sistema compie delle deduzioni errate portando a dei risultati validi solo parzialmente. I parametri richiesti non sono sempre di facile comprensione e quindi rendono l'algoritmo di difficile uso e non adatto ad un'esecuzione completamente automatica.

Il secondo algoritmo analizzato è stato proposto da Dannenberg nel 2003 [59].

Il prototipo proposto prevede di utilizzare la partitura MIDI per fornire un'esecuzione ideale del brano e successivamente mettere in relazione questa esecuzione con quella che deve essere sincronizzata. La relazione tra le due partiture è realizzata tramite la creazione di *chroma vector*. Questi vettori sono composti da 12 elementi ed ognuno di essi rappresenta una classe di pitch. Tramite una FFT il sistema calcola lo spettro del segnale considerando blocchi da 0.25s, mette in relazione le linee frequenziali con le rispettive classi di pitch sommando i valori assoluti di tutte le ottave che corrispondono ad un'unica classe e inserisce all'interno del vettore il valore assoluto di ogni classe di pitch diviso per la sommatoria di tutti i valori assoluti presenti nello spettro. Una volta ottenuti questi vettori croma, sia dallo spettro generato dal MIDI sia dallo spettro generato dall'esecuzione reale, sono messi in relazione tramite una matrice in cui ogni elemento rappresenta la distanza euclidea tra due diversi vettori croma. In figura 4.13 è rappresentata una tabella di similarità come si può vedere le durata del brano audio è maggiore rispetto al MIDI, questo inconveniente è facilmente risolo tramite un'operazione di proporzionalità. La distanza euclidea tra i vettori è rappresentata dalla tonalità di grigio assunta dai singoli elementi partendo dal nero per una distanza zero e utilizzando grigi sempre più chiari per valori crescenti. Il passo finale dell'algoritmo è la ricerca di un percorso "continuo" che unisca il vertice in basso a sinistra con il vertice in alto a destra e che minimizzi la sommatoria delle distanze tra i vettori croma.

Figura 4.13 Una matrice di similarità [59]



Nella figura 4.13 si nota che uno dei tracciati possibili approssima la diagonale della matrice. Questa caratteristica non è casuale ma si verifica quando il tempo del brano rimane sostanzialmente costante, se il brano in esame presenta invece delle forti variazioni di tempo il percorso si discosterà in maniera sempre maggiore dalla diagonale principale.

Nella tabella sottostante sono mostrati i risultati sperimentali ottenuti da questo algoritmo. Sono riportati gli errori medi tra tutti gli eventi sincronizzati e la loro deviazione standard. Sono stati testati tre brani: il primo movimento della 5th sinfonia di Beethoven, "Let It Be" dei Beatles e una versione con false variazioni di tempo nel MIDI sempre del primo brano.Le imprecisioni sembrano imputabili solo ad errori di quantizzazione

Test Name	Avg. Error	Std. Deviation
Beeth.	0.052s	0.111s
Beeth.-vary tempo	0.034	0.056
"Let It Be"	0.076	0.112

L'ultimo algoritmo preso in esame è stato proposto da Muller nel 2004 [60].

Questo algoritmo prevede una prima fase di analisi di basso livello composta da: un algoritmo di pitch tracking tramite filtri ellittici e un metodo di peak detection. I risultati di queste analisi sono inseriti in due liste S e P. S contiene tutte le note trovate e P contiene tutti i picchi. Queste liste sono messe in relazione tramite una funzione che calcola vari possibiliriferimenti. I risultati sono valutati tramite le informazioni presenti in partitura. La relazione che meglio si adatta allo spartito è la sincronizzazione trovata.
Un prototipo di questo algoritmo è disponibile in rete all'indirizzo:

www-mmdb.iai.uni-bonn.de/download/sync/

dove è possibile scaricare un software che visualizza le sincronizzazioni effettuate dagli autori.

Capitolo 5

Metodi per la sincronizzazione MP3

In questo capitolo sono illustrati i vari approcci sviluppati nel corso di questo lavoro di tesi per risolvere il problema della sincronizzazione su file audio compressi.

Il primo paragrafo descrive le analisi svolte sull'informazione di Block Type. Tramite questa analisi si è verificata la possibilità di sfruttare questa caratteristica per individuare all'interno del file audio degli istanti di particolare interesse quali punti di attacco di note (transienti), ripartenze dopo pause e informazioni ritmiche in generale[45]. Il secondo paragrafo descrive invece il tipo di architettura proposta e le varie implementazioni sviluppate.

5.1 Analisi informazione di Block Type

Lo standard MP3 non fissa alcun parametro relativo all'utilizzo di un particolare tipo di finestratura è quindi possibile avere dei file MP3 privi di blocchi corti o anche file composti esclusivamente da questo tipo di blocchi. Questa considerazione ha imposto di analizzare vari codec MP3:

1. LAME 3.96.1 [64]
2. BLADE [65]
3. TSUNAMI PRO [66]
4. BLAZE [67]
5. FRAUNHOFER [68]

Sono stati utilizzati i seguenti segnali di test:

1. sound1.wav ----------→ 3 impulsi della durata di circa mezzo secondo generati artificialmente tramite Sound Forge composti da un'onda quadra con frequenza di 440Hz MONOFONICO
2. cembalo.wav ----------→ breve partitura monodica eseguita dal digital performer di Finale utilizzando il suono del clavicembalo nello standard MIDI. STEREO
3. ocarina.wav----------→ Stessa partitura eseguita da digital performer partendo dal suono di ocarina nello standard midi. STEREO
4. Primus Wynona.wav-→ estratto polifonico di 10 secondi preso da un CD-DA con batteria molto presente. Origine STEREO reso MONO estraendo solo il canale sinistro
5. Zaburon Intro.wav----→ estratto di circa 10 secondi da cd audio di orchestrazione, sintetizzata in parte tramite MIDI, con 4 attacchi eseguiti in fortissimo. Origine STEREO reso MONO estraendo solo il canale sinistro

Tutti i segnali sono stati trascritti su un file WAV con frequenza di campionamento a 44.100 kHz e con quantizzazione a 16 bit senza aggiunta di metadati ove possibile escluderli.

I test svolti hanno avuto diverse finalità. Inizialmente si è cercato di capire se la presenza di short block fosse effettivamente utile per l'individuazione di eventuali attacchi. In seguito si è analizzata: la qualità con cui i vari encoder gestiscono quest'informazione, il variare dell'informazione di block type aumentando e diminuendo il bit rate, e il comportamento dei vari encoder a seguito di un'operazione di normalizzazione del segnale. L'ultimo esperimento svolto è stato relativo al solo encoder Lame. Questo encoder mette a disposizione dell'utente varie opzioni per la configurazione del modello psicoacustico che controlla anche la scelta della finestratura utilizzata. Si è quindi valutato come le variazioni di questi parametri si riflettessero sull'utilizzo di short block.

Per eseguire i test è stato inizialmente modificato il decoder di riferimento ISO. Purtroppo questo decoder, non essendo utilizzato commercialmente, non supporta alcuni meta dati al di fuori dello standard che sono comunemente inseriti da alcuni encoder all'interno di frame fittizi. Questa caratteristica genera alcuni problemi nell'estrazione dello spettro MDCT e dell'informazione di block type si è quindi scelto di lavorare sul decoder Lame, il cui codice è facilmente reperibile su internet e di libero dominio[64].

Il decoder Lame è stato modificato in maniera tale da rendere visibili: su di un file testo le in formazioni relative allo spettro MDCT, su di un altro file testo il tipo di blocco utilizzato. Queste informazioni sono state date in input ad un piccolo script Matlab che ha generato tutte le figure presenti in questo paragrafo. Sono stati utilizzati file di testo per facilitare la gestione delle informazioni su tutti i software adoperati.

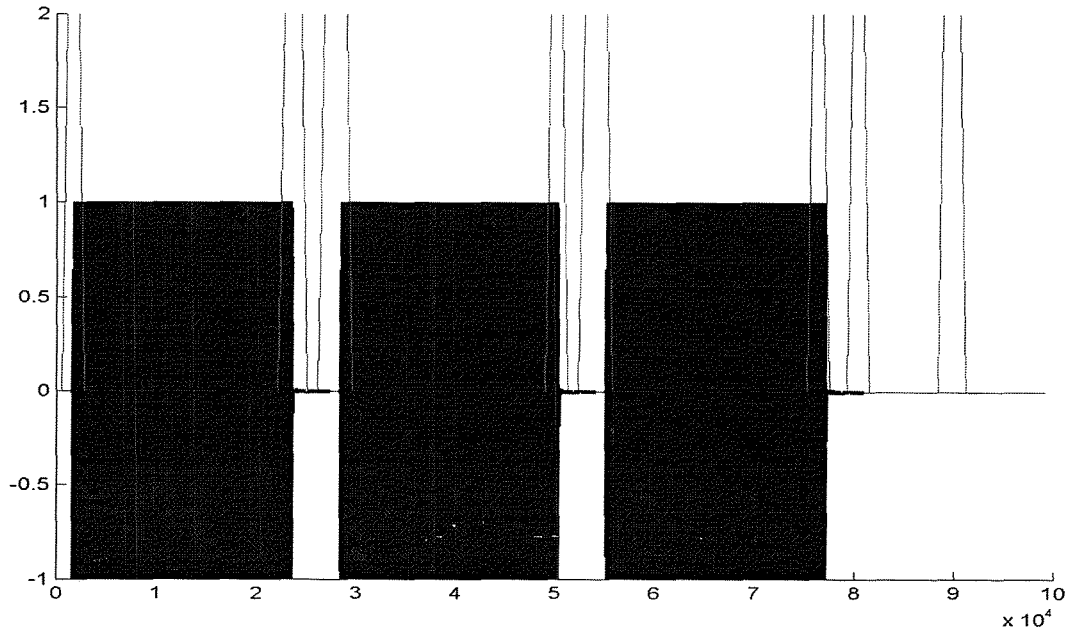
Qualità del Block Type per l'individuazione di eventuali attacchi

È stato codificato il file di test 1 con tutti gli encoder selezionati con bit rate a 320 kbp/s e con eventuali variabili qualitative settate al massimo.

Nelle immagini seguenti la parte più scura indica la forma d'onda analizzata, mentre le linee verticali individuano la presenza di blocchi diversi dal comune blocco lungo, in maniera più formale:

il valore 0 della linea sottile indica un blocco lungo, il valore 2 tutti gli altri tipi di blocco. Le ordinate rappresentano il tempo.

Figura 5.1 Resa del file di test 1 tramite l'encoder Lame con un bit rate a 320 kbit/s



Tutti gli altri encoder testati hanno confermato l'andamento mostrato in figura 5.1. L'unico a mostrare un comportamento "anomalo" è l'encoder Blade che non individua mai i transienti decrescenti.

Questo risultato su di un segnale generato artificialmente e ideale per questo tipo di analisi ha confermato che l'utilizzo di questa informazione non è indipendente dall'encoder utilizzato, ma permette, abbastanza agevolmente, di individuare eventuali transienti [45]

Qualità degli encoder nella gestione degli Short Block

Con questa fase di test si è analizzato quale tra i vari encoder disponibili offrisse una "migliore gestione" dell'informazione relativa al Block Type.

È stato utilizzato il file di test numero 4, perché in questo estratto si sono notati il maggior numero di variazioni sul Block Type.

Ad una prima analisi si può dedurre che maggiore è il numero di Short Block individuati più l'encoder risulta appropriato per un'operazione di sincronizzazione.

Tutti gli encoder sono stati settati con un bit rate a 320 Kbps e con la massima qualità possibile.

Figura 5.2 Resa del file di test 4 tramite l'encoder Lame con un bit rate a 320 kbit/s

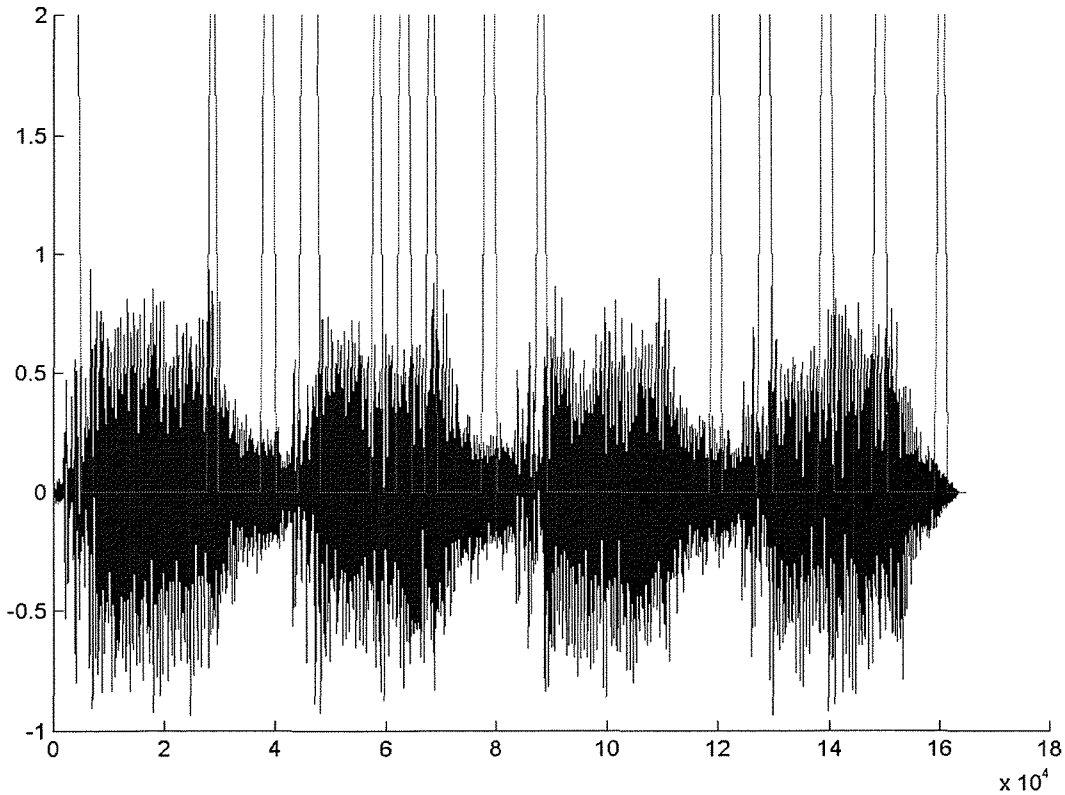
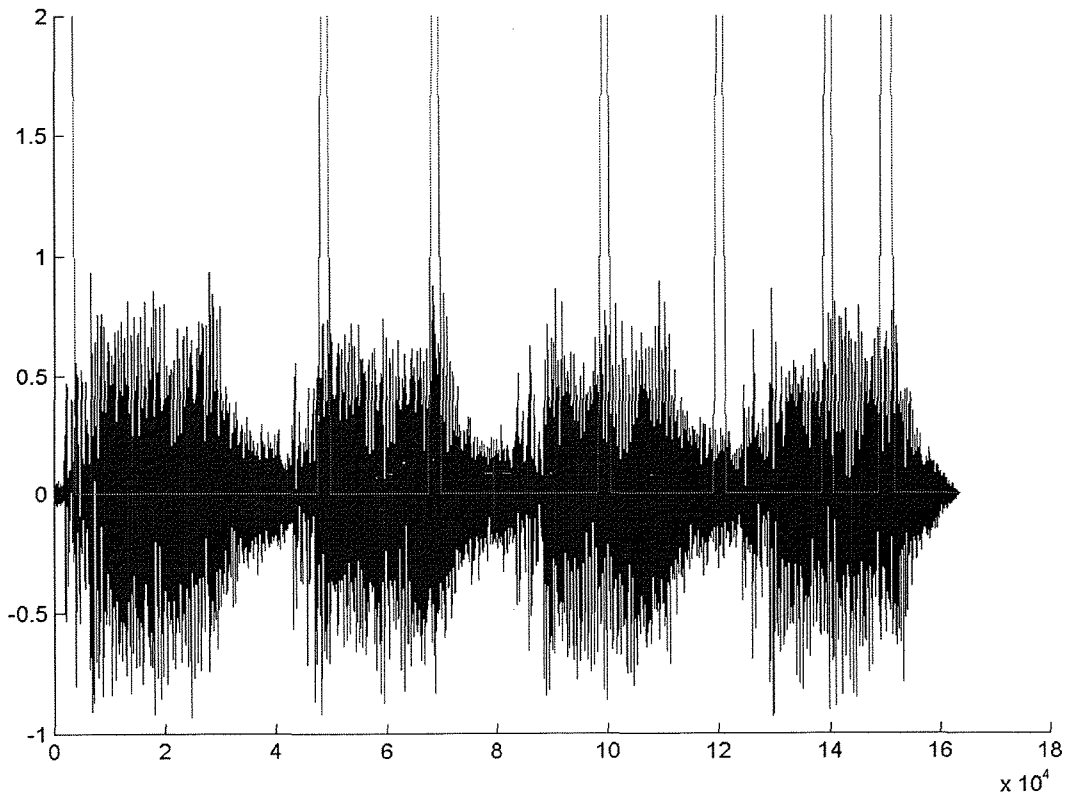


Figura 5.3 Resa del file di test 4 tramite l'encoder Blade con un bit rate a 320 kbit/s



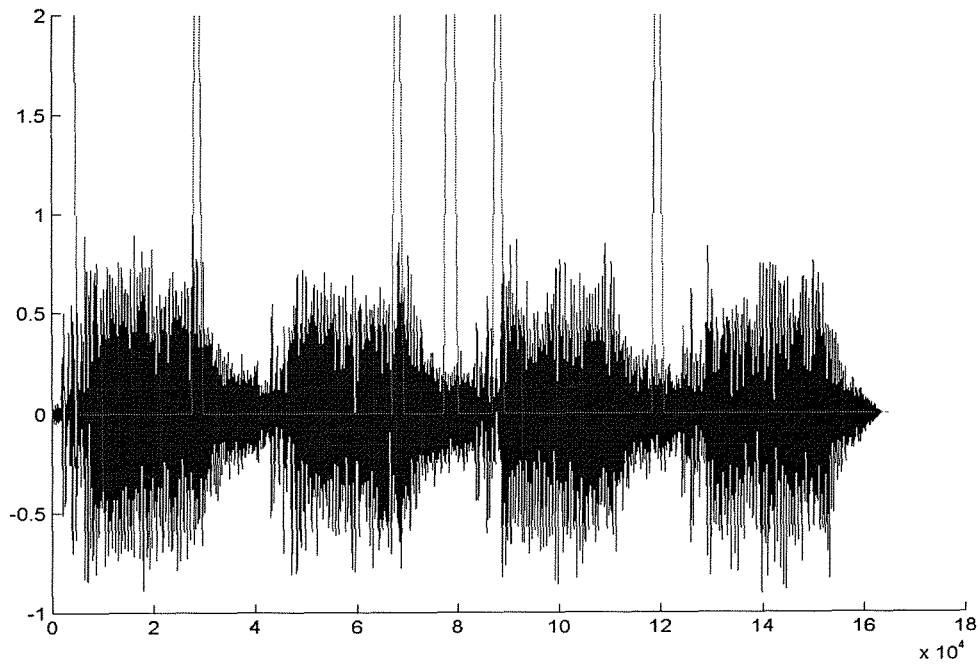
Fraunhofer e Blade hanno identificato meno short block degli altri codec mentre l'encoder Lame è quello che ne ha riconosciuti il numero maggiore. Tsunami e Blaze hanno generato risultati molto simili posizionandosi rispetto al numero di short block individuati in maniera intermedia. È da notare che avere un maggior numero di short block non è necessariamente una cosa positiva in quanto potrebbe creare problemi generando una serie di falsi allarmi[60]. Gli short block sono comunque usati solo negli istanti di tempo in cui si ha un colpo di rullante o un colpo di cassa, questo conferma i risultati ottenuti in [45].

Variazione dell' informazione di Block type variando il Bit Rate

Questa analisi si è resa necessaria in modo da rendere eventuali algoritmi che sfruttino questa caratteristica indipendenti dal bit rate utilizzato in codifica. È stato sempre testato il file 4 per le considerazioni fatte nel punto precedente. Le impostazioni del codec sono state lasciate invariate per ogni encoding modificando il solo parametro del bit-rate richiesto. Tutti gli encoder hanno confermato che la scelta di utilizare o meno short block è indipendente dal bit rate.

L'unica eccezione è rappresentata dall' encoder Lame per cui variando il Bit rate varia anche il numero dei transienti identificati. In Lame il numero degli short block è direttamente proporzionale al bit rate utilizzato come si vede confrontando le figure 5.2 e 5.4

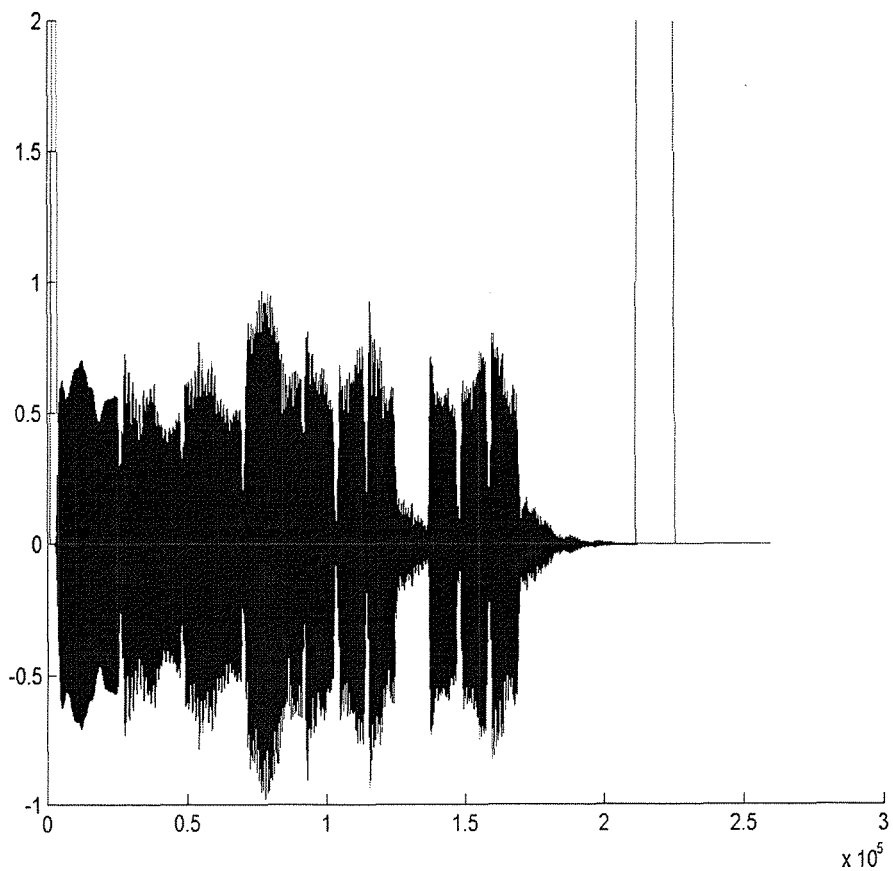
Figura 5.4 Resa del file di test 4 tramite l'encoder Lame con un bit rate a 64 kbit/s



Variazione dell'informazione di Short Block in funzione di un'operatore di normalizzazione

Si è scelto di analizzare l'impatto di un operatore di normalizzazione per verificare se una massimizzazione dei picchi del segnale producesse un aumento nel numero di short block utilizzati. L'esperimento si è svolto osservando prima gli short block individuati dall'encoder Lame sul file 3, successivamente su questo file è stato applicato l'operatore di normalizzazione presente in Sound Forge secondo il template "Maximize Peak Value" [69] che massimizza i picchi del segnale portandoli ai massimi valori possibili . Il segnale ottenuto è stato quindi ricodificato tramite l'encoder Lame e sono state analizzate le variazioni degli short block individuati tra il segnale originale e quello modificato. L'operatore di normalizzazione risulta avere una utilità solo in presenza di silenzi con valori di ampiezza pari a 0. Questo permette eventualmente di riuscire a valutare più correttamente l'inizio o la fine dei brani ma non ha un impatto significativo nell'ambito considerato.

Figura 5.5 File 3 dopo la normalizzazione (nell'originale nessuno short block utilizzato)



Variazione dell' informazione di Block type in LAME variando le opzioni disponibili

LAME è l'unico encoder in cui la scelta di utilizzare uno short block è dipendente dal bit rate richiesto, essendo questo fatto anomalo sia da un punto di vista teorico che nell'eperienza pratica ottenuta con gli altri encoder si è deciso di analizzare le opzioni disponibili e di valutare il loro impatto sulla scelta degli short block.

Sono state considerate le seguenti opzioni:

- --athshort ignora *GPSYCHO* per gli short block
- --athonly ignora *GPSYCHO* completamente, utilizza solo *ATH*
- --notemp disabilita gli effetti temporali di mascheramento
- -q <arg> <arg> = 0...9. Default
  -q 5 -q 0: massima qualità ma molto lento
  -q 9: qualità scadente ma molto veloce

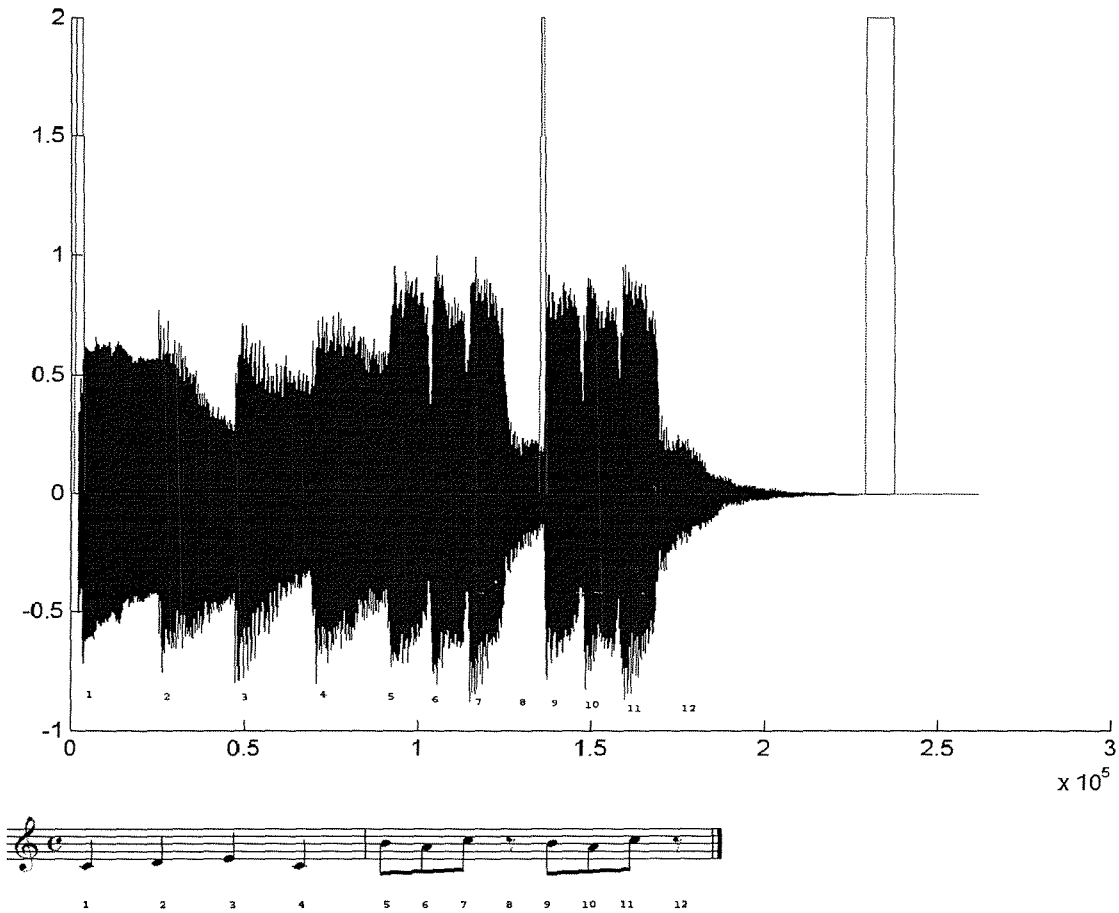
GPSYCHO indica il modello psicoacustico mentre *ATH* la soglia di mascheramento in quiete, l'opzione q seleziona il tipo di modello psicoacustico implementato

Nessuna delle opzioni precedenti influenza, in Lame, la scelta di eventuali short block, combinando le opzioni tra di loro i risultati non subiscono variazioni è dunque sempre confermata la tendenza ad utilizzare un numero minore di short block con bit rate bassi.

Risultati su esempi reali

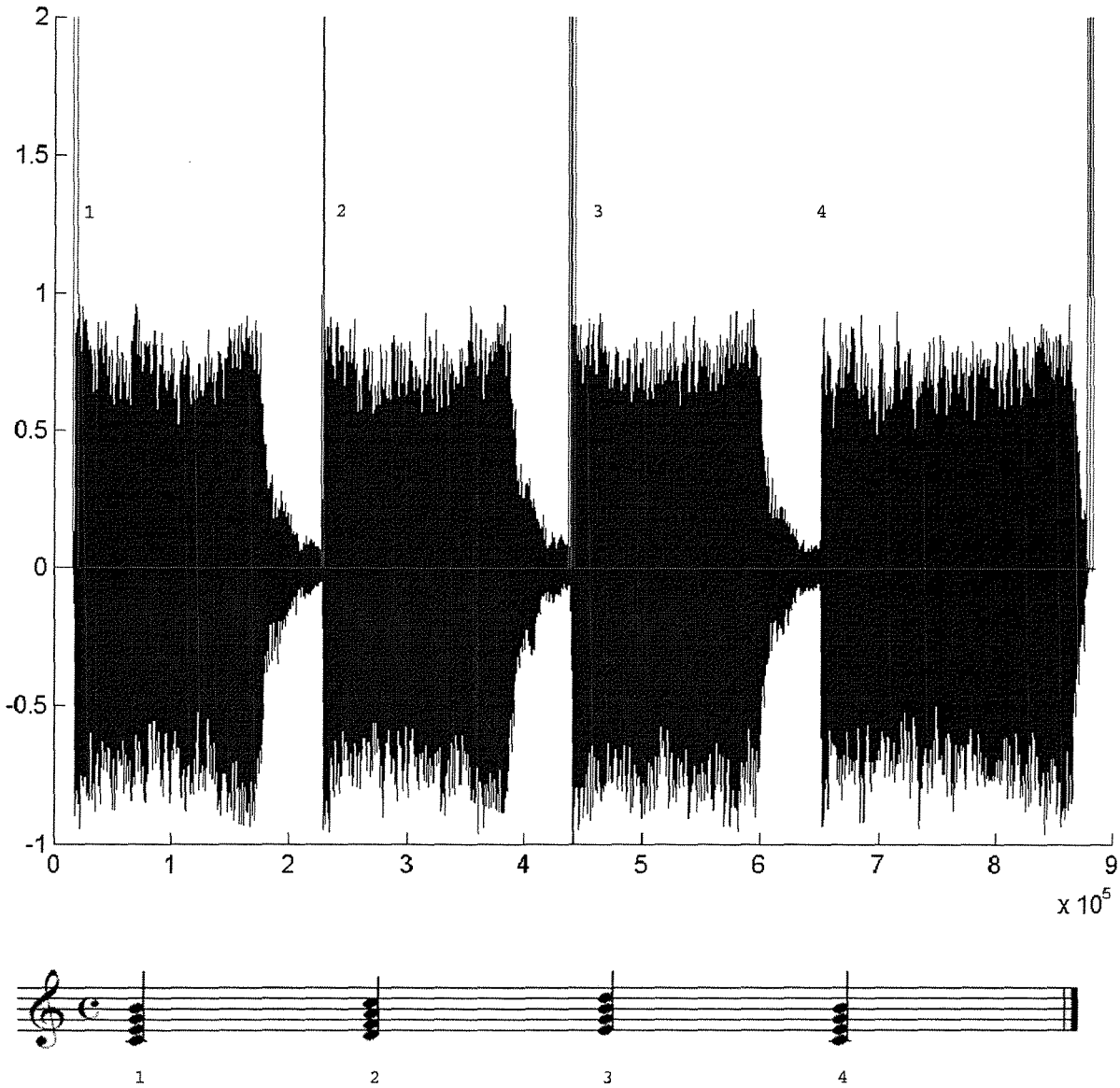
In questa parte si è verificata la reale affidabilità della presenza di short block come indice di un evento significativo nell'audio da sincronizzare in partitura. A dispetto del lavoro [45] i brani utilizzati sono privi di sezione ritmica, come avviene nella maggior parte dei brani di musica classica. Nelle figure seguenti è anche riportata la partitura che ha generato il file audio in esame (per brevità nella seconda immagine è stata riportata una partitura fittizia, corretta ritmicamente ma non completa armonicamente).

Figura 5.6 File 2 con relativa partitura



Dalla figura 5.6 si nota come lo short block permetta di riconoscere l'attacco della nota 9 dopo la pausa. Purtroppo questa caratteristica non si è dimostrata stabile perchè mantenendo inalterata la partitura ma variando lo strumento utilizzato per la sua esecuzione non è stato utilizzato nessuno short block. Questo è dovuto al fatto che il secondo strumento utilizzato è un'ocarina che non ha una fase di attacco particolarmente forte e quindi, la sua codifica, non necessita di short block per evitare artefatti. Nel secondo caso (fig 5.7) lo short block opera bene su una partitura effettivamente suonata ed estratta direttamente da un CD-DA commerciale; riesce ad individuare anche l'inizio di una rullata di timpani posta proprio alla fine dell'estratto nonostante l'operazione di fade out applicata tramite Sound Forge.

Figura 5.7 Nel file 5 l'encoder riconosce gli attacchi dei primi 3 accordi



Conclusioni

Segue un breve elenco delle conclusioni che sono state tratte in relazione all'operazione di finestratura a seguito degli sperimenti descritti in questo capitolo:

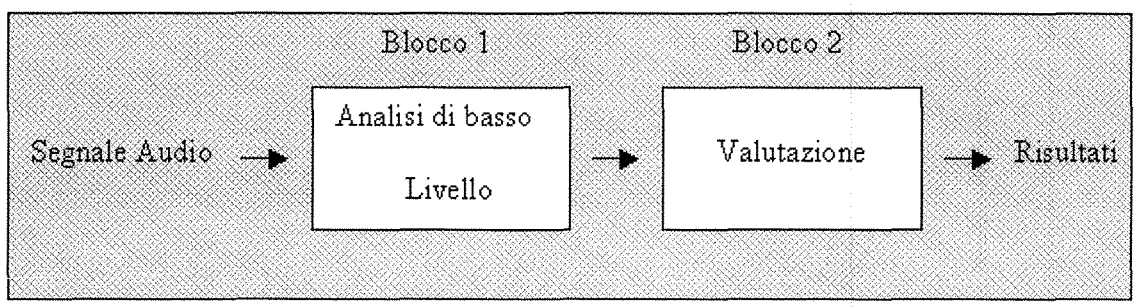
- L' algoritmo che sceglie il tipo di finestratura impiegato è una caratteristica implementativa dipendente dall'encoder, che ha piena libertà nell'utilizzo di questa opzione messa a disposizione dal formato.
- Alcuni encoder danno la possibilità all'utente di forzare un solo tipo di finestratura per l'intero file.
- Nonostante non sia gestita univocamente, questa caratteristica ha mostrato un certa stabilità nei test effettuati nell'individuazione di eventuali sezioni ritmiche. Questo risultato concorda con quanto affermato in [45]. Un suo utilizzo sistematico può, però, generare alcune difficoltà perché è un opzione fortemente dipendente dall'encoder utilizzato.
- La scelta del tipo di finestratura da adottare è un'operazione sostanzialmente indipendente dal bit rate richiesto. Questo dato è molto utile perché permette di valutare questa informazione indipendentemente dal livello di compressione del file MP3
- Un operatore di normalizzazione non ha un particolare impatto sul tipo di finestratura impiegato all'interno del segnale.
- Le variazioni della finestratura evidenziate in principio e alla fine del file audio 3 rispetto ad un'operazione di normalizzazione potrebbero rivelarsi utili per individuare l'inizio e la fine di un brano in quei file MP3 contenenti più pezzi codificati in un unico file.

Purtroppo questa informazione pur avendo delle caratteristiche interessanti non ha mostrato una particolare utilità reale nei problemi di AMS, principalmente perchè non si ha mai la certezza del suo posizionamento rispetto alla partitura simbolica; in altre parole è difficile capire quale evento abbia effettivamente generato uno short block in una partitura priva di sezione ritmica. Questa informazione può però rivelarsi utile nel caso in cui si voglia raffinare il lavoro svolto da un algoritmo di sincronizzazione sfruttando il posizionamento di una finestra di tipo short per ottenere la massima precisione.

5.2 Architettura generale della soluzione proposta

Tutti gli algoritmi di AMS sviluppati in questo lavoro di tesi possono essere divisi in due blocchi logici: il primo effettua un'analisi di basso livello del segnale. Il secondo blocco valuta i risultati ottenuti dall'analisi di basso livello e sceglie coerentemente tra i risultati trovati quelli più attendibili per effettuare la sincronizzazione. Nel sotto paragrafo seguente sono presentati i metodi utilizzati per realizzare una buona analisi di basso livello mentre il sotto paragrafo 5.2.2 presenterà i metodi implementati per la valutazione dei risultati ottenuti dal blocco precedente. È da notare che tutti i metodi proposti e scartati lavorano in maniera sequenziale partendo dal primo evento presente in partitura e dal primo granulo MP3 cercando di sincronizzare tutto in un'unica passata.

Figura 5.8 Schema generale degli algoritmi sviluppati



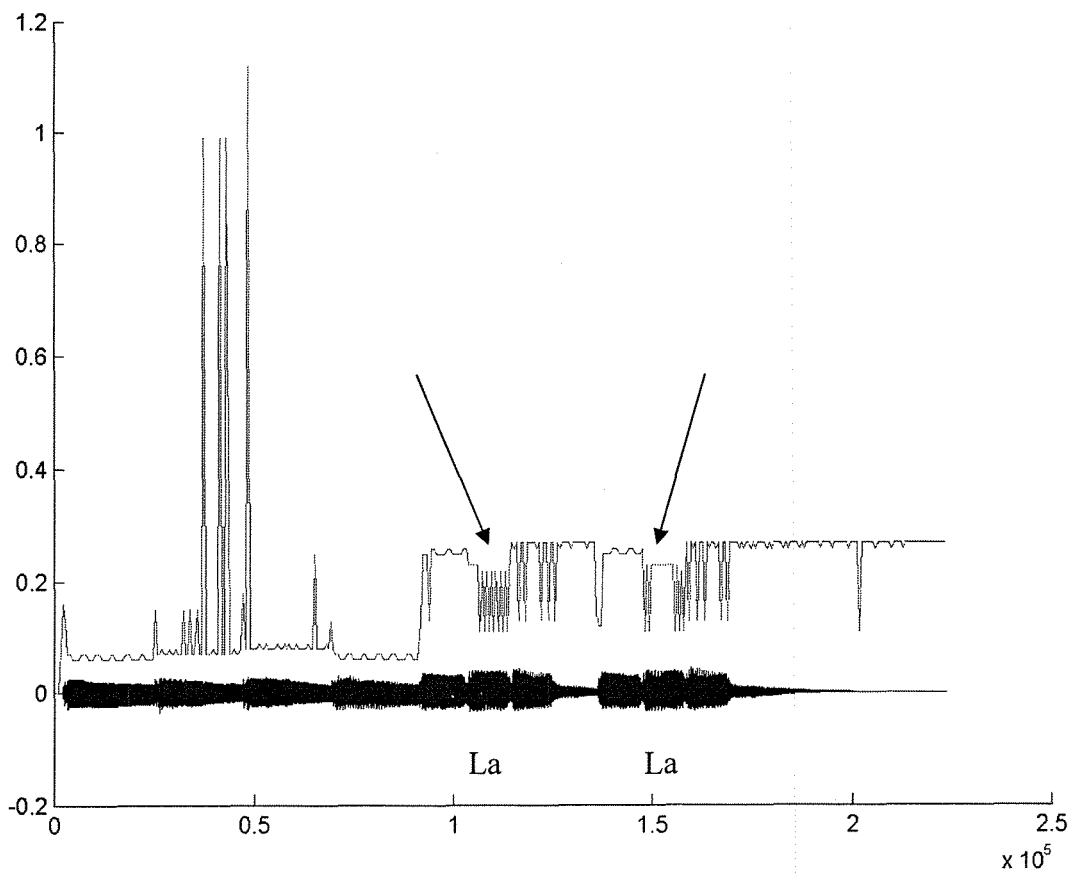
5.2.1 Algoritmi per l'analisi di basso livello

Questa parte del sistema ha il compito di fornire al blocco successivo i migliori dati possibili che riescano ad individuare l'inizio delle note. Per trovare un algoritmo valido è stato applicato un metodo di indagine sistematico applicando sempre sullo stesso file audio (il file 3 del cap 4) l'analisi implementata e realizzando un grafico che sovrapponesse l'andamento del segnale con i risultati trovati dall'analisi e verificando quindi una correlazione tra l'inizio delle note nel segnale e la funzione restituita dall'analisi. Nelle figure seguenti saranno mostrati questi risultati: la parte più scura rappresenta lo sviluppo del segnale rispetto al tempo mentre la linea continua rappresenta la funzione di analisi.

Calcolo della posizione del massimo nello spettro del segnale

Questo metodo prevede di calcolare per ogni granulo di interesse la posizione del massimo e quindi verificare se questo massimo è in una linea frequenziale di una nota presente in partitura. Dalla figura 5.9 si nota subito come la posizione dei massimi nello spettro di un file MP3 mantenga solo una minima correlazione qualitativa con quanto effettivamente presente in partitura. Infatti si nota subito che anche all'interno della stessa nota la posizione del massimo ha un oscillazione molto ampia, questo andamento è dipendente sia dalla nota suonata che da fattori non prevedibili. Le frecce indicano proprio due note uguali (due La) che hanno un andamento molto diverso pur essendo poste tra note uguali. Questo metodo è stato dunque scartato

Figura 5.9 Andamento della posizione del massimo nello spettro MDCT per il file 2

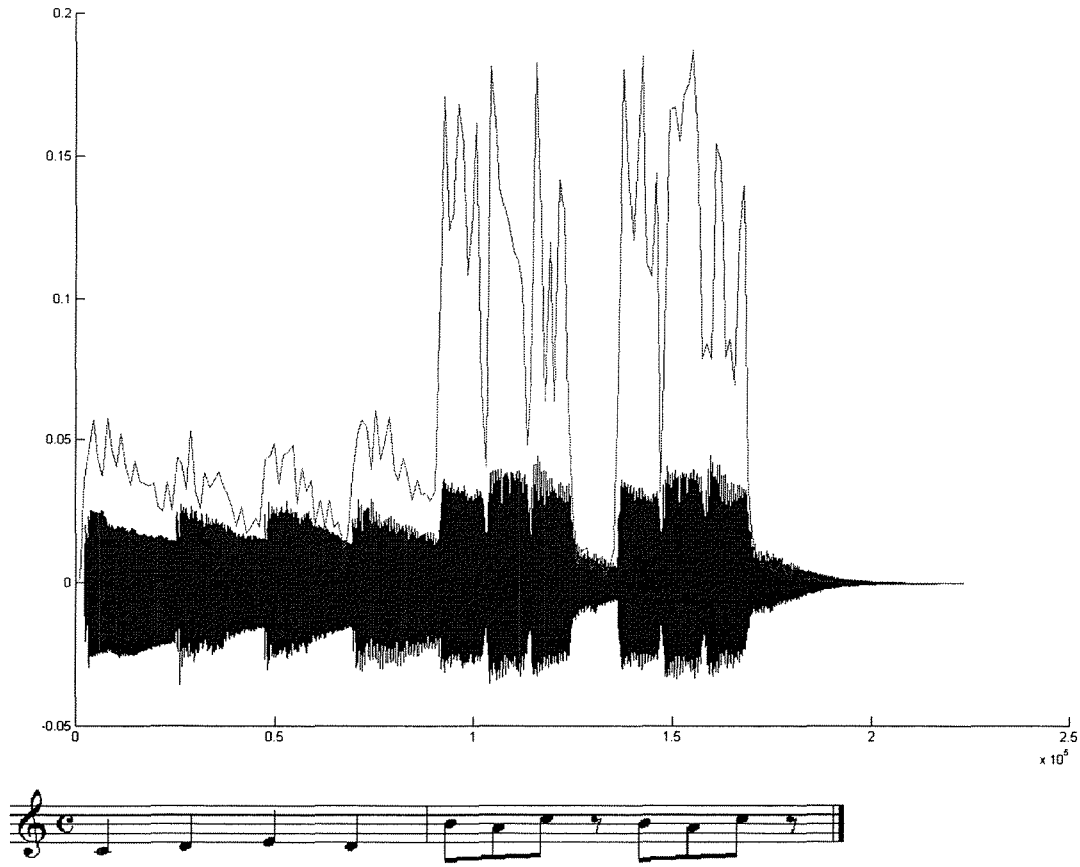


Stima dell'autocorrelazione energetica del segnale

Tramite il calcolo dell'autocorrelazione tra le energie di due granuli adiacenti viene stimato quanto il segnale rimanga stabile tra istanti successivi. In presenza di valori molto bassi potrebbe essere lecito aspettarsi un inizio di una nota. L'andamento del valore di autocorrelazione è determinato dal ritardo con cui il segnale è confrontato con se stesso. Come dimostrato in [70] si può utilizzare il valore dell'autocorrelazione per determinare il pitch delle note cercate utilizzando un ritardo pari al periodo della frequenza fondamentale della nota cercata. Questo è un metodo valido solo per file monofonici, è di conseguenza poco utile per sincronizzare partiture complesse. In [5] è stato esposto come trovare un coefficiente di ritardo appropiato che permetta di osservare gli attacchi delle singole note di una certa durata. Questo metodo ha però il problema di non essere stabile in quei generi dove l'interpretazione varia notevolmente la durata delle note.

Si è scelto quindi di confrontare il segnale con se stesso con il ritardo minimo (circa 13ms, la durata temporale di un granulo) ipotizzando che in una situazione di stabilità del segnale il coefficente di autocorrelazione sia molto elevato (approssimando quasi l'energia) e che risulti invece minimo in presenza di una transizione delle frequenze fondamentali. Come si può notare dalla figura 5.10 questa ipotesi si è rivelata erronea ed in effetti da un punto di vista teorico un periodo di circa 13ms è comunque troppo elevato per un'ipotesi di questo tipo. In MP3 non sia ha la possibilità di abbassare ulteriormente questo ritardo per gli ovvi limiti strutturali, di conseguenza il metodo è stato abbandonato.

Figura 5.10 Andamento del coefficiente di autocorrelazione con ritardo pari ad 1 granulo



Stima dell'energia del segnale

Questo metodo prevede il calcolo dell'energia del segnale in ogni singolo granulo e in presenza di forti variazioni energitiche viene supposto esserci un istante di interesse.

Dalla figura 5.11 si nota che l'energia del segnale è approssimabile anche in MP3. Per brani monofonici con strumenti che hanno attacchi molto forti e fasi di sustain relativamente brevi potrebbe essere anche possibile trovare gli attacchi delle note. Un approccio di questo tipo appare però inapplicabile a situazioni più generali. Si è dunque scelto di passare ad un metodo che non utilizzasse l'energia dello spettro nel suo insieme ma che andasse a considerare solo l'energia contenuta nelle linee frequenziali di interesse della nota trovata. Inizialmente si è utilizzata come frequenza interessante quella subito superiore al valore della frequenza dell'armonica fondamentale. Questo metodo però genera una serie di problemi dovuti alla scarsa risoluzione frequenziale dei coefficenti MDCT. È necessario che note diverse abbiano linee frequenziali diverse per evitare ambiguità, questa condizione, come

mostrato dal grafico 5.1, è violata a partire dalle note subito sotto l'ottava centrale visto che i coefficienti MDCT hanno una risoluzione frequenziale di poco inferiore a 40 Hz. Per ovviare a questo inconveniente si è scelto di considerare solo la prima, la seconda e la terza armonica escludendo quindi la frequenza fondamentale dall'analisi.

Figura 5.11 Andamento dell'energia del segnale

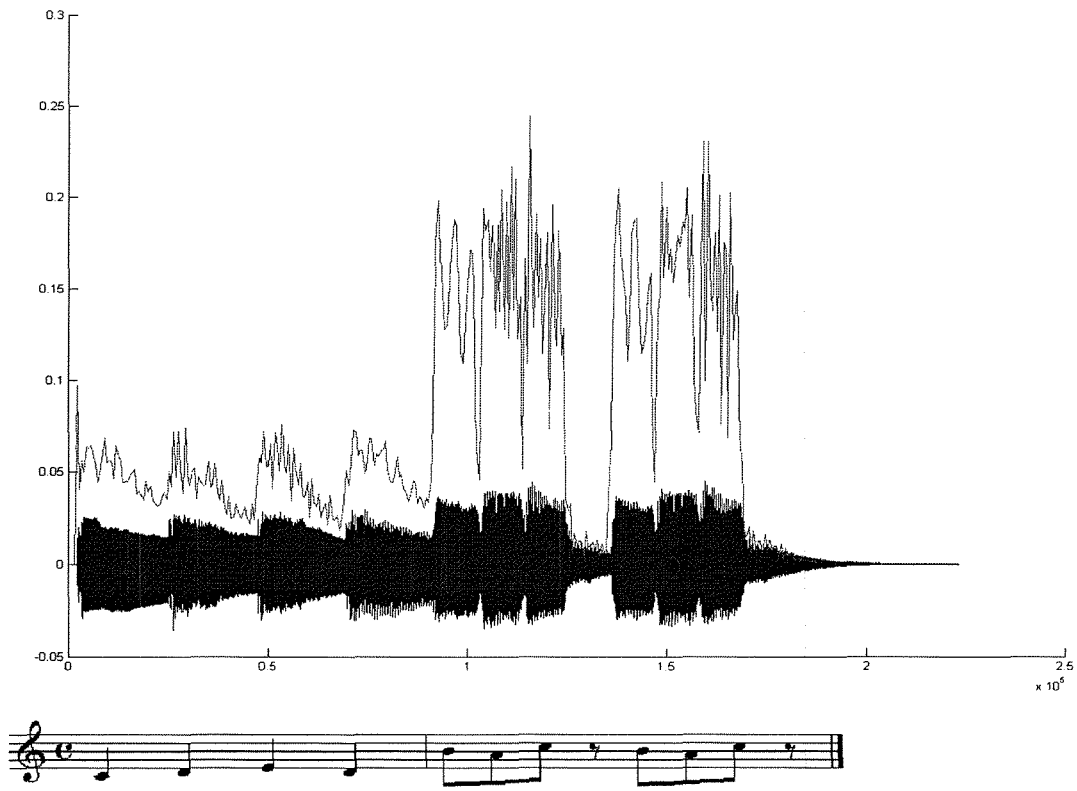
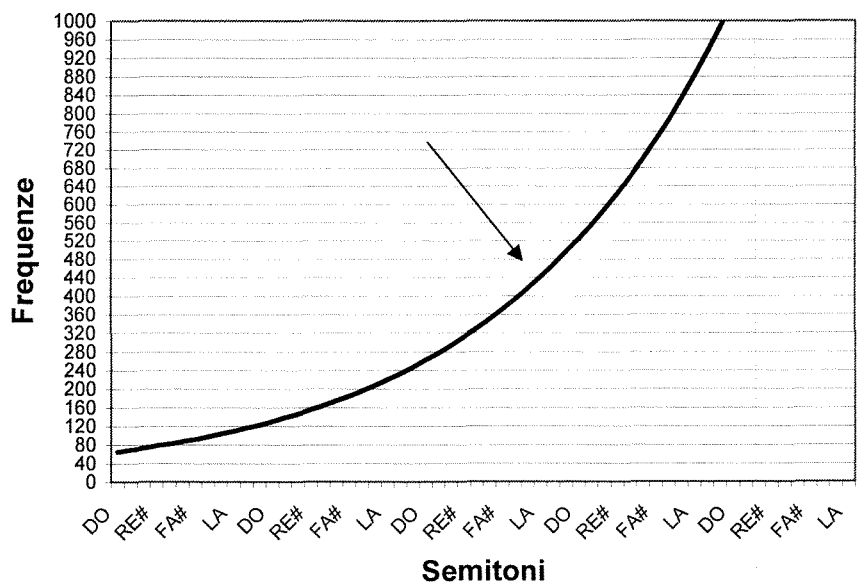


Grafico 5.1 Ogni nota dovrebbe essere in una banda diversa, la freccia mostra il La centrale



Utilizzando una tecnica di sogliatura si è cercato di riconoscere la presenza di una nota della partitura nel file audio, questo metodo si è però rivelato inapplicabile per l'impossibilità di trovare un valore di soglia adeguato a causa delle grosse oscillazioni della funzione di energia per singola nota all'interno dello stesso file e maggiormente tra file diversi.

Energia relativa per una banda di interesse

Si è scelto di limitare l'alta variabilità dei dati sfruttando non più il valore assoluto dell'energia ma un valore relativo di questa grandezza calcolato secondo la formula:

$$energia\ relativa\ (nota) = \frac{\sum_{ottava=1}^{3} x(nota_{ottava})}{\sum_{i=1}^{576} x^2(i)} \qquad (1)$$

Intendendo con x il valore del coefficiente MDCT in una posizione dello spettro delle frequenze; con *nota* la posizione nello spettro delle frequenze della fondamentale di una nota. Il valore di *ottava* indica che sono considerate la prima, la seconda e la terza armonica.
Questa formula offre dei buoni risultati pratici (figura 5.12). In presenza della nota cercata il suo valore è molto elevato, pur avendo ampie oscillazioni e restituisce valori che tendono a zero se la nota cercata non è presente nel granulo analizzato. La scelta di non considerare la frequenza fondamentale della nota presa in esame è dovuta alla bassa risoluzione dello spettro MP3 (vedi capitolo 5.2.1 "stima dell'energia del segnale"). Nelle figure 5.12 e 5.13 si possono osservare gli andamenti di questa funzione relativi alle note Mi e Do, lo spartito in basso indica la posizione delle note all'interno del file audio. È da notare che utilizzando la formula (1) i valori di energia trovati non sono più semplici valori assoluti ma possono essere visti come valori percentuali che indicano la parte di energia totale dello spettro contenuta nelle linee frequenziali di interesse. Di conseguenza eventuali operazioni di sogliatura potranno essere visti come la percentuale di energia dello spettro compresa nelle linee frequenziali di interesse.

Considerati i buoni risultati ottenuti questo metodo è stato scelto come algoritmo finale ed è stato utilizzato per le successive prove dei vari algoritmi di valutazione descritti in seguito.

Figura 5.12 Andamento dell'energia relativa della nota MI

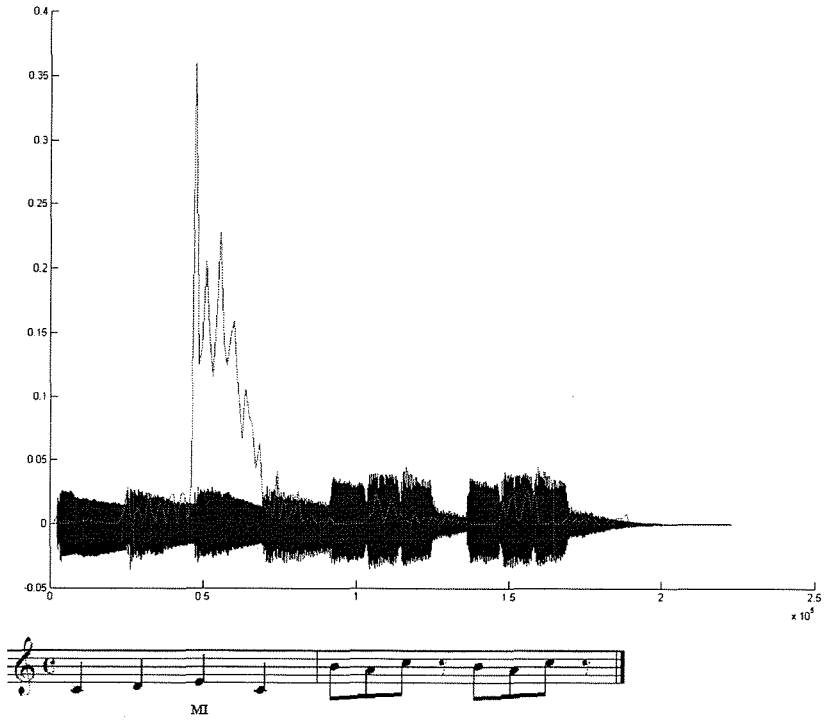
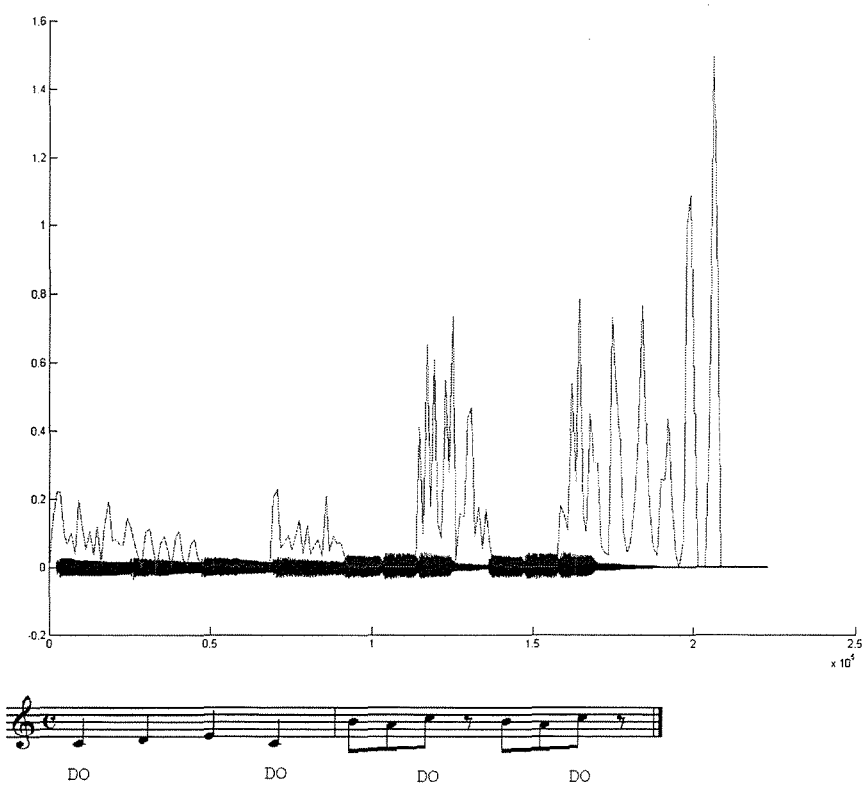


Figura 5.13 Andamento dell'energia relativa della nota DO



5.2.2 Algoritmi di valutazione

In questa sezione sono presentati gli algoritmi di valutazione dei risultati della funzione (1). Con algoritmo di valutazione si intende: primariamente un metodo che permetta di trovare efficacemente ed automaticamente un buon valore di soglia che identifichi l'effettiva presenza delle note e che dia luogo ad un numero minimo di false individuazioni (e che tralasci un numero minimo di note reali); secondariamente un metodo che sia in grado di restringere efficacemente il campo di ricerca delle note.

Il primo algoritmo presentato "mappatura lineare temporale a singola passata" è l'algoritmo più semplice che compia questi passi. Si vedrà come questo algoritmo fallisca in molte situazioni e quindi saranno presentate le strategie che si sono ideate per costruire degli algoritmi più efficaci (5.2.2.1). L'ultimo paragrafo (5.3) è la descrizione del metodo scelto "proporzione globale audio partitura". Questo metodo trova un valore di soglia molto preciso che varia in funzione della nota cercata. L'utilizzo di questi valori di soglia offre dei risultati così validi all'algoritmo di selezione che non si ha più la necessità di effettuare una ricerca ristretta solo in alcuni punti ma permette appunto di agire globalmente su tutta l'esecuzione. Sarà poi la parte dedicata alla selezione di questi risultati a creare il file di sincronizzazione finale scegliendo i risultati ritenuti più validi.

Mappatura lineare temporale a singola passata

Il primo algoritmo di valutazione implementato prevede una soglia fissa sull'energia relativa ed una finestra di ricerca temporale molto stretta (circa 3 decimi di secondo) legata ad una mappatura diretta della partitura nel file audio. Questo tipo di algoritmo fallisce in due casi: quando la nota cercata non è nella finestra di ricerca, ad esempio, quando il tempo relativo del brano non è costante, e quando il valore di soglia impostato non è adatto, evento molto probabile visto che la funzione energia relativa non mantiene valori costanti all'interno dello stesso brano e tanto meno tra brani diversi. Queste considerazioni hanno portato a separare questi algoritmi in due famiglie: una che deve fornire un luogo di ricerca sensato, è inutile cercare la prima nota di un brano verso la fine e viceversa (*algoritmi per la mappatura temporale*), la seconda che ha il compito di fornire dei buoni valori di soglia per cercare di limitare le ampie oscillazioni della formula (1) (*algoritmi per la ricerca dei valori di soglia*).

5.2.2.1 Algoritmi evoluti per la mappatura temporale

Mappatura lineare temporale in due fasi

Questo algoritmo come il precedente si basa su una mappatura diretta della partitura nel file audio separando però questa mappatura in due fasi distinte.

Nella prima fase, il cui compito è mappare solo gli inizi delle battute (di solito gli eventi musicali più facilmente riconoscibili), la durata del file audio è divisa per il numero di battute presenti in partitura e ad ogni battuta è associato un possibile inizio. La ricerca della nota considerata, la prima della battuta, avviene quindi all'interno di una finestra di ricerca centrata nel punto di possibile inizio della misura e ampia circa 2 secondi. La scelta di una finestra di ricerca così ampia è possibile perchè considerando solo gli inizi delle battute non si corre il rischio di avere eventuali sovrapposizioni. Nella seconda fase, invece, si utilizza una finestra più piccola (circa mezzo secondo) per gli eventi all'interno della battuta; in questo caso la mappatura è effettuata partendo dai risultati trovati nella prima fase. Con queste variazioni l'algoritmo non ha migliorato i risultati ottenuti in precedenza perchè nella fase di ricerca delle battute eventuali note uguali poste a cavallo di una battuta rendono impossibile la scelta di un inizio certo per la battuta creando un'ambiguità che si ripercuote su tutte le note successive.

Mappatura adattiva temporale

Vista l'impossibilità di utilizzare finestre di ricerca ampie (creano ambiguità) e visto che non è possibile mantenere finestre di ricerca posizionate staticamente sopratutto se molto strette (non riescono a seguire efficacemente le variazioni di tempo) si è tentato di mantenere un'ampiezza di finestra molto piccola, ma cercando di posizionarla dinamicamente tentando di prevedere la posizione dell'evento successivo stimandola dai risultati ottenuti precedentemente. L'idea è quella di avere un sistema che anche in presenza di cambi di tempo istantanei fallisca in un numero ristretto di note, andando a leggere fasi di sustain come attacchi, mentre "aggiorna" le proprie stime per poi riallinearsi sugli effettivi inizi delle note.

Questo metodo si è rivelato abbastanza affidabile per file di piccole dimensioni, 10-15 secondi, ma ha dimostrato una propensione a fallire su durate più elevate. Questo comportamento è dovuto al fatto che ogni errore, anche minimo, nella sincronizzazione, tende

ad amplificarsi nella stima della posizione degli eventi musicali successivi in una sorta di retroazione positiva che manda il sistema completamente fuori strada.

5.2.2.2 Algoritmi per la ricerca dei valori di soglia

Sogliatura sulla media dell'energia relativa per una banda di interesse

In questo algoritmo si è tentato di limitare l'alta variabilità dei dati disponibili utilizzando come soglia non più un valore energetico ma una forte variazione rispetto all'energia media del segnale sulla banda di interesse. L'energia media è calcolata mediante una lista FIFO di 5 o 10 elementi calcolando la somma dell'energia della banda di interesse nei granuli esaminati. Quando cambia la banda di interesse un'operazione di buffering riempie nuovamente la lista con l'energia della nuova banda considerata nei vari granuli. Un granulo è considerato interessante quando si verifica che la sua energia è superiore di 15 volte rispetto all'energia media considerata. Purtroppo questo approccio non è stato sufficiente a tracciare le variazioni energetiche risultando a volte troppo lento nel seguire le variazioni e a volte troppo influenzato da valori erroneamente troppo elevati.

Sogliatura variabile in base alle note presenti in partitura

In questo metodo si è cercato di stimare il valore di una possibile soglia in base al numero e al tipo di note presenti in partitura utilizzando principi di acustica e fisica delle onde descritti in [15]. Considerando che i valori energetici sono la percentuale di energia dello spettro presente nelle linee frequenziali si è cercato di capire come queste percentuali variassero in funzione del numero delle note presenti in partitura e alla loro disposizione armonica. Si è osservato che effettivamente all'interno di un singolo file è possibile fare delle stime anche relativamente accurate di questi valori. Purtroppo non è possibile stimare queste percentuali genericamente per ogni possibile situazione perchè sono direttamene influenzate da caratteristiche quali: timbro dello strumento, volume di registrazione, dinamica. L'insieme di queste caratteristiche si è rivelato troppo ampio per gestire agevolmente l'impostazione di un valore di soglia.

5.3 Come superare i problemi presentati

Analizzando nel complesso tutti i metodi presentati si può osservare la comune caratteristica di esaminare il problema localmente cioè analizzando ogni singolo evento musicale ed ogni granulo indipendentemente dal complesso del brano e della partitura. Questo tipo di approccio si è dimostrato sostanzialmente errato. Oltre alle problematiche analizzate sopra è da considerare anche che la funzione (1) non è esente da segnalazioni errate come mostrato in figura 5.13 dove l'analisi delle frequenze del Do va a coprire anche il Re successivo. Questo problema non può essere risolto localmente, il sistema non può sapere se in effetti in quel punto è presente un Do o un Re senza avere una visione di insieme della partitura e del brano audio. Questo problema ne genera anche un altro di difficile soluzione senza un'analisi globale. Immaginiamo, ad esempio, che al posto della seconda nota presente nella partitura riportata in figura 5.13 sia presente un accordo composto dal Re e da un ulteriore Do (armonicamente discutibile, ma non impossibile) il sistema avrebbe segnalato il primo Do correttamente ma trovando subito un altro Do da sincronizzare avrebbe segnalato subito, al granulo successivo, un nuovo punto di sincronizzazione sbagliando completamente. Localmente sono possibili due strade per evitare questo problema o si cercano di sincronizzare tutte le note presenti in un accordo, e questa strada appare difficile considerati gli effetti di mascheramento che esistono tra intervalli armonici, o si inserisce un flag che segnala quando una nota è già stata sincronizzata precedentemente e quindi non utilizzabile per sincronizzare l'evento successivo (o gli eventi successivi in caso di variazioni molto rapide). Questa strada ovviamente può portare al vicolo cieco di non avere alcuna nota disponibile per sincronizzare un evento in quanto tutte precedentemente utilizzate. Sarebbe possibile inserire nel sistema un algoritmo che rilevi i transienti del segnale permettendo quindi di individuare le fasi di attacco, purtroppo i metodi più efficienti per risolvere questo problema sono applicati direttamente nel dominio del tempo che non è raggiungibile se non tramite una completa decodifica del file MP3. Queste considerazioni giustificano un approccio globale.

Il paragrafo 5.3.1 "proporzione globale audio partitura", analizzerà l'effettiva soluzione proposta per la parte dedicata alla valutazione dei risultati dell'analisi di basso livello. Questo algoritmo utilizza sempre (1) come funzione per l'analisi di basso livello ma risolve i problemi di *mappatura temporale* e di *ricerca dei valori di soglia* non ragionando più localmente ma analizzando la partitura e il brano audio nel suo complesso, collegando

assieme il problema del valore di soglia e della mappatura temporale così da renderli interdipendenti. Ovviamente un approccio globale richiede molte più risorse in termini di memoria rispetto ad un approccio locale. Questo problema sarà discusso nel capitolo conclusivo di questo lavoro di tesi.

5.3.1 Proporzione globale audio partitura

Come osservato nel capitolo precedente appare molto difficile trovare un metodo valido per risolvere i problemi relativi alla scelta di un buon valore di soglia e di una mappatura temporale valida agendo localmente. Per superare questa difficoltà si è scelto di considerare il brano e la partitura globalmente[5].

Per identificare il valore migliore per la soglia, relativa alla nota considerata, è necessario esaminare la partitura (strato simbolico) e la sua esecuzione complessivamente (strato audio). Per mettere in relazione i due diversi strati considerati si è fatto ricorso ad una proporzione che leghi queste due grandezze. Anzitutto bisogna calcolare quante *divisions* (vedi capitolo 3) del totale di cui è composta la partitura siano effettivamente occupate dalla nota data. Il rapporto tra questo valore e il totale delle *divisions* di cui è composta la partitura del brano può essere messo in relazione con il rapporto tra la quantità di granuli nel file MP3 che contengano la nota considerata e il numero di granuli totali presenti nel file MP3. Da questa relazione si può dedurre facilmente la formula:

$$x = \frac{DuratadelBrano * Duration(nota)}{DivisionsTotali} \qquad (2)$$

Intendendo con *duration(nota)* il numero di *division* che contengono la nota presa in esame, con *durata del brano* il numero dei granuli che compongono il file MP3 e con *divisions totali* il numero di *division* che compongono l'intera partitura. Il valore incognito x è il numero di granuli che dovrebbero contenere la nota data.

Il calcolo del valore di soglia relativo ad una singola nota è effettuato partendo da una soglia iniziale minima (vedremo nel prossimo capitolo cosa si intende per minima) che consideri quasi tutti i granuli del file MP3 come granuli in cui sia effettivamente presente la nota considerata. Il valore minimo è poi incrementato fino a trovare un numero di granuli che

contengano ipoteticamente la nota considerata che sia minore o uguale del numero di granuli *x* calcolati dalla formula (2).

La formula (2) è sempre verificata per brani con una velocità metronomica costante, per brani che invece hanno delle velocità variabili non dà luogo ad errori eccessivi in quanto le variazioni temporali, anche se molto elevate, sono distribuite sulle stime di tutti gli eventi della partitura. Questo permette di ottenere valori di soglia molto affidabili che generano una serie di possibili inizi per gli eventi molto affidabili; di conseguenza non si ha più la necessità di restringere il campo di ricerca sulla posizione degli eventi presenti in partitura nel relativo file audio. Osservando più approfonditamente la formula (2) si intuisce come in realtà essa sottintenda un'operazione di mappatura temporale, molto diversa rispetto a quelle esposte in 5.2.2.1, che è effettuata, non più sulla posizione del singolo evento (mappatura locale), ma sull'insieme degli eventi presenti in partitura (mappatura globale) stimando non gli inizi di ognuno di essi ma la "quantità" di tempo che occupano nel file audio. Vedremo nel capitolo successivo lo schema generale del prototipo realizzato e quindi come siano stati effettivamente utilizzati i risultati trovati da questo algoritmo.

Capitolo 6

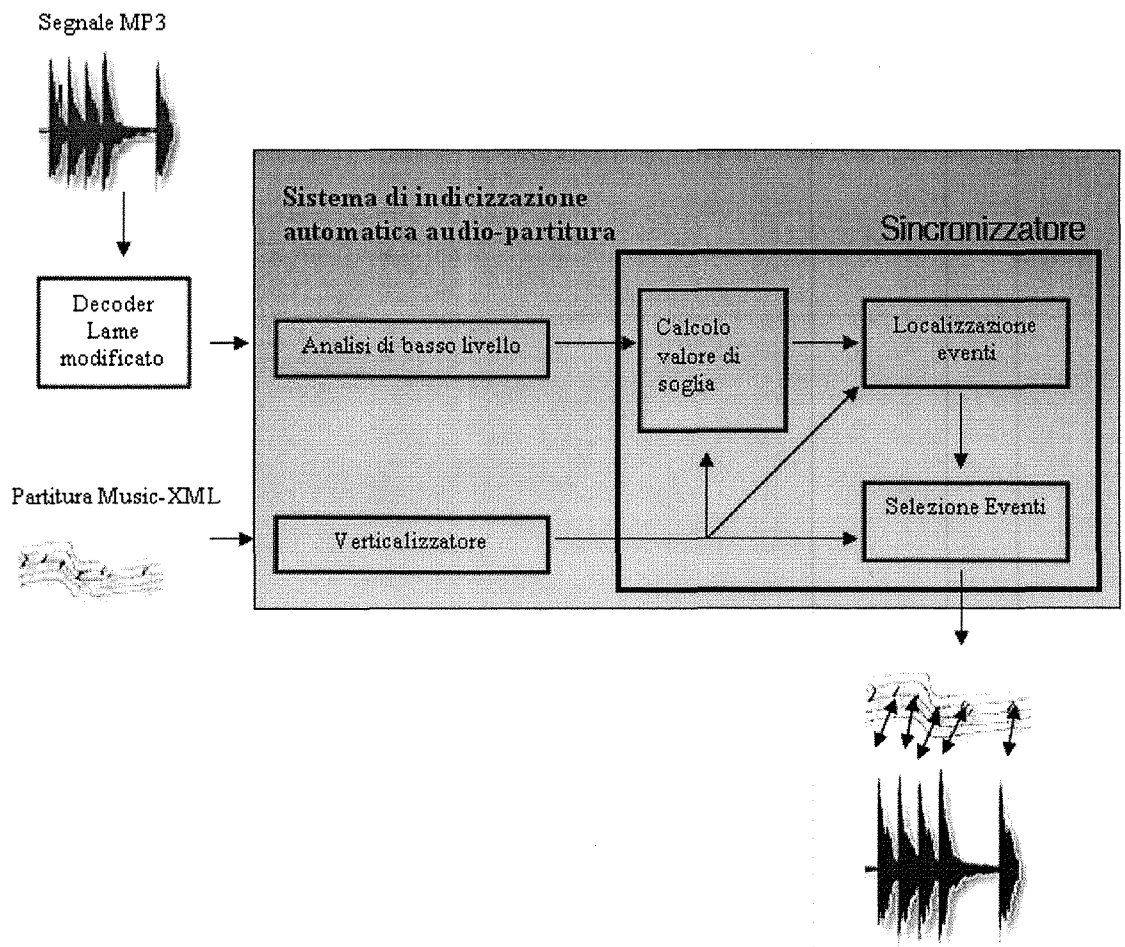
Soluzione finale e prototipo software realizzato

In questo capitolo sarà descritto inizialmente il prototipo proposto nel suo insieme evidenziandone i blocchi più rilevanti. Il paragrafo 6.2 è dedicato ai linguaggi software utilizzati per la sua realizzazione. Il paragrafo 6.3 presenta l'architettura del sistema presentato evidenziandone le strutture dati e i metodi più significativi In ultimo è presentato un breve manuale utente che spiega come far funzionare il prototipo e i vari file di input e di output.

6.1 Descrizione della soluzione proposta

In questo paragrafo è descritta in generale la soluzione proposta che è ben schematizzata dalla figura 6.1. Per ogni blocco presente sarà descritto il suo funzionamento senza soffermarci troppo sui particolari implementativi che saranno analizzati approfonditamente nel paragrafo 6.3.

Figura 6.1 schema generale del prototipo proposto



Come si può osservare subito dalla figura 6.3 il sistema è composto da 2 parti distinte: il decoder Lame modificato e il sistema di indicizzazione automatica audio-partitura vero e proprio. Il decoder Lame ha il compito di decodificare il file MP3 fino ad estrarre i campioni MDCT, le informazioni relative al block type e al mixed block flag. Questi dati sono trascritti su 3 file testo distinti che dovranno essere passati in input al resto del sistema. Il decoder Lame modificato è un eseguibile a se stante che funziona in maniera totalmente indipendente

dal resto del sistema, infatti il decoder, per estrarre i dati, effettua una normale operazione di decodifica portandola comunque a compimento anche se il file risultante non è assolutamente utilizzato. Questa scelta architetturale permette la sostituzione di questo decoder, con altri più efficienti (eventualmente sviluppati ad hoc) che non sprechino tempo eseguendo una decodifica completa non necessaria, senza troppe complicazioni. Di contro è necessario che l'utilizzatore esegua sempre due programmi distinti per realizzare una sincronizzazione: Lame che estrae i dati e il sistema di indicizzazione audio-partitura che li sincronizza con la partitura.

Il sistema di indicizzazione audio-partitura proposto è composto da tre blocchi principali: Analisi di basso livello, Verticalizzatore e Sincronizzatore. Il blocco Verticalizzatore gestisce l'informazione simbolica necessaria al funzionamento del sistema. Questa parte effettua un'operazione di verticalizzazione sull'intera partitura, dividendola nei gruppi verticali che sono gli elementi veri e propri da sincronizzare. Oltre a questo risultato il Verticalizzatore restituisce anche la percentuale di *divisions* (vedi capitolo 3) occupate da una data nota. Quest'informazione è necessaria per applicare la formula (2).

Il blocco che compie l'analisi di basso livello si interfaccia direttamente con Lame. Ha il compito di rendere compatibili i file testo con l'intero sistema caricandoli correttamente in memoria. Tali dati sono utilizzati da questo modulo per calcolare l'energia di ogni granulo e l'energia relativa presente in ogni linea frequenziale dello spettro MDCT. Il macro blocco *Sincronizzatore* è la parte fondamentale dell'intero sistema. Inizialmente il Sincronizzatore calcola il valore di soglia migliore per ogni nota presente in partitura. La soglia è calcolata in accordo con quanto descritto nel paragrafo 5.3.1 utilizzando prima la formula (1), per avere una distribuzione dell'energia delle note all'interno del file audio. Successivamente il sistema applica dei valori di soglia crescenti che partono da un valore minimo. Il valore di soglia non è più incrementato quando il numero di granuli selezionati dall'operazione di sogliatura risulta minore o uguale al numero di granuli trovati dalla formula (2). Una volta che le soglie sono state definite il blocco Localizzazione eventi applica queste soglie ai risultati della formula (1). L'ultimo blocco di *Sincronizzatore* scorre tutte le note presenti negli eventi rilevati dal verticalizzatore. Per ogni nota considerata cerca il primo granulo che dovrebbe contenerla. Una volta trovato questo granulo lo considera come punto di sincronizzazione per la nota data, scrive i dati nel file di output e passa alla nota successiva ricominciando ad analizzare i granuli partendo dall'ultimio granulo considerato.

6.2 Linguaggi utilizzati

Per implementare il prototipo proposto in 6.3 è stato utilizzato il linguaggio di programmazione C#. C# è un linguaggio orientato agli oggetti semplice, elegante e indipendente dai tipi, sviluppato da Microsoft per l'implementazione di vari tipi di applicazioni. C# fornisce meccanismi intrinseci di verifica del codice per un elevato livello di protezione, garbage collection e controllo dei tipi. Inoltre offre un supporto nativo ad XML tramite il .NET Framework. Questo impone ovviamente che la soluzione proposta necessita di .NET Framework per essere eseguita, considerando che si tratta di un prototipo questa richiesta non appare eccessivamente vincolante.

La modifica dell'encoder Lame è stata fatta invece in C++ perchè il codec stesso è scritto in questo linguaggio. Durante la fase di implementazione e di test è stato utilizzato l'ambiente Matlab per la visualizzazione dei risultati.

6.3 La soluzione proposta

Il prototipo sviluppato è composto da tre moduli differenti: Verticalizzatore, SpectrumAnalysis e Sincronizzatore. Nel paragrafo 6.3.1 saranno descritti i metodi principali e le più importanti strutture dati utilizzate da questi metodi. Nel capitolo 6.3.2 sarà descritto il main del prototipo e il suo funzionamento.

6.3.1 Strutture dati utilizzate e metodi principali

Verticalizzatore

Questo modulo legge in ingresso un file MusicXML formattato secondo il DTD *score-partwise* e ne estrae le parti che compongono la partitura. Verifica che tutte le parti abbiano lo stesso tipo di divisione della battuta e che queste abbiano lo stesso numero di *divisions* se così non è il sistema calcola dei fattori moltiplicativi che normalizzano tutte le *duration* degli eventi rispetto alla divisione maggiore riportata nelle segnature di tempo e rispetto al numero massimo di *divisions* trovate. La classe crea una tabella(*TableOut*) che ha un numero di righe

pari al numero di voci presenti in partitura (*voci*) e un numero di colonne pari a al numero di divisions che compongono l'intero brano (*atomi verticali*). Ogni elemento all'interno di questa tabella è una lista di note che rappresentano gli accordi nelle singole voci. Le note singole sono considerate accordi degeneri. Dunque ogni accordo (nota o pausa) è inserito un numero di volte pari alla sua duration normalizzata. Ogni accordo (nota o pausa) ha un flag (*IsNew*) che è impostato a true nell'*atomo verticale* in cui è inserito per la prima volta e a false per tutti gli inserimenti successivi. Le note sono rappresentate come una quintupla composta da : *nome*, *ottava*, *alterazione*, *black list*, *chiave*. Il campo *black list* è un flag che indica se questa nota non è adatta ad essere sincronizzata, il campo *chiave* è stato inserito in previsione di un interfacciamento con MX. Questo campo permette di risalire velocemente ed univocamente ad una nota in MusicXML per poi confrontarla con la rispettiva in MX. Le *chiavi* sono inserite incrementalmente di conseguenza per scorrere tutte le note presenti in partitura è sufficiente scorrere le chiavi. Gli accordi contengono oltre che la lista delle note presenti: un carattere *accento* che non è stato utilizzato ma che eventualmente potrebbe contenere il tipo di accento che ha l'accordo, un flag *IsAChord* che individua se si tratta di un accordo o meno e come già detto il flag *IsNew*.

Partendo dalla tabella *TableOut* il sistema ne costruisce una versione "compatta" che ha un numero di righe sempre pari alle voci presenti in partitura (*voci*), ma con un numero di colonne variabile pari al numero degli *eventi musicali verticalizzati*. Con *eventi musicali verticalizzati* si intendono tutti gli *atomi verticali* in cui si ha l'inizio di un accordo (nota o pausa). Questa tabella finale mantiene quindi tutte le informazioni di verticalizzazione che descrivono le note presenti in un gruppo verticalizzato. Le durate di questi gruppi sono inserite in un array di interi. Un metodo calcola successivamente: il numero di divisioni assolute occupate da ogni nota, tramite la tabella degli *eventi musicali verticalizzati*, e l'array delle durate relative.

Spectrum Analysis

Questo modulo ha il compito di caricare in memoria i file generati dal decoder Lame modificato, contenenti: lo spettro del segnale (`HybridINVerticale3.txt`), l'informazione di block type (`BlockType.txt`) e l'informazione di mixed block flag (`MixedBlockFlag.txt`) (è possibile che alcuni granuli in MP3 abbiano una codifica "mista" dei coefficienti MDCT). È composto da due classi: `SpectrumFiles` e `SpectrumBuffer`.

La classe `SpectrumBuffer` dichiarara e amministra tutte le strutture dati per la gestione delle informazioni contenute direttamente nel file MP3. Le strutture dati utilizzate sono: una tabella composta da due righe e 576 colonne chiamata `freq` che ha il compito di caricare in memoria i coefficienti MDCT di un granulo o di un intero frame, coerentemente con quanto richiesto dai metodi che vi accedono, e 2 liste di interi che contengono il valore di block type e di mixed block flag estratti da tutto il brano MP3.

`SpectrumFiles` dichiara un oggetto di tipo `SpectrumBuffer` chiamato `Spectrum` tramite il quale questa classe accede alle strutture dati dichiarate `in SpectrumBuffer.`

`SpectrumFiles` ha il compito di leggere i file testo generati dal decoder Lame, di renderli compatibili con le strutture dati di `SpectrumBuffer` e di inserirli correttamente. Ha inoltre l'incarico di costruire i dati grezzi per l'analisi di basso livello effettuata dal modulo `Sincronizzatore`. I dati grezzi calcolati da questa classe sono: l'energia totale di ogni singolo granulo tramite il metodo `LongBlockEnergy` (che sono inseriti nell'array `Energy`) e l'energia relativa di tutte le linee frequenziali dello spettro MDCT, tramite il metodo `CriticalBandEnergy`, che vanno a sostituire i precedenti valori contenuti in `freq`.

Sincronizzatore

Questa modulo è la parte fondamentale del sistema proposto ha il compito di unire i risultati delle classi *Verticalizzatore* e *Spectrum Analysis* e di costruire il file testo contenente il risultato finale.

Il modulo `Sincronizzatore` è composto da due classi: `TabellaPitchToBanda` e `TimeSync`.

La classe `TabellaPitchToBanda` contiene due tabelle per mettere in relazione l'altezza simbolica delle note con la relativa frequenza fondamentale e quindi con la rispettiva linea frequenziale dello spettro MDCT. La prima tabella (`TabFreq`) è una matrice con le colonne che rappresentano le ottave e le righe che rappresentano la posizione della nota nella notazione americana. Questa tabella contiene le frequenze fondamentali di tutte le note. La seconda tabella (`TabBande`) è invece una tabella hash che mette in relazione la posizione della nota (ottava e nome) con la rispettiva linea frequenziale dello spettro MDCT.

I valori contenuti in questa tabella sono generati ed inseriti utilizzando il metodo seguente:

```
public void FillTabellaPitchToBanda()
{
  for (int i = 0; i < ottave; i++)  //OTTAVE È IL NUMERO DI OTTAVE CONSIDERATE,11
    for (int j = 0; j < note; j++)  // NOTE È IL VALORE 12

      TabBande[ (i * note) + j ] = (int)(TabFreq[i,j] / 38.28125) + 1;

          // i * note + j NON FA DIFFERENZA FRA NOTE NATURALI E ALTERATE
          // 38.28125 È LA DIFFERENZA FRA DUE FREQUENZE DELLO SPETTRO MDCT
          // L'OPERAZIONE "/" CON IL CAMBIO DI TIPO IN (int) DIVENTA UNA
          // DIVISIONE INTERA
}
```

Questo metodo calcola la posizione di una nota in uno spazio lineare moltiplicando il numero di ottava della nota per 12 (base) e sommando la posizione derivata dal nome della nota (spiazzamento). In questa posizione inserisce il risultato della divisione intera tra la frequenza della nota considerata e il valore `38.28125`, questo valore rappresenta la differenza minima che due frequenze devono avere per essere inserite in due diverse linee frequenziali MDCT. Il valore `38.28125` è ottenuto dividendo la massima frequenza contenuta in un file MP3 campionato a 44,1 kHz per il numero di linee frequenziali presenti in MDCT.
Sono state sviluppate due funzioni:

```
public int FindPosNotaDaNota ( char nome, int alter )
public int FindLineaDaNota ( int ottava, char nome, int alter )
```

Rappresentando le note di un ottava come un array da 12 posti. La prima funzione restituisce la posizione di una nota in base al suo nome e ad eventuali alterazioni all'interno di questo array; la seconda funzione, invece, dati in input: il nome, l'ottava e le alterazioni eventualmente presenti restituisce la rispettiva linea frequenziale all'interno dello spettro MDCT contenuta nella tabella `TabBande`.

La classe `TimeSync` implementa l'algoritmo di sincronizzazione vero e proprio.

Un oggetto `TimeSync` è composto da un oggetto `MusicXml` chiamato `StratoSimbolico`, da un oggetto `SpectrumFiles` chiamato `CodecOut`, da un oggetto `TabellaPitchToBanda` chiamato `Bande`, da un contatore intero per i granuli chiamato `GranuloCounter`, da un numeo `double` chiamato `meanenergy` che contiene la media dell'energia di tutti i granuli presenti nel file MP3, e le seguenti strutture dati:

- `TabellaPicchi`
- `ArrayEnergiaCritica`
- `GranuloOutOutSync`

`GranuloOutOutSync` è l'array di interi che mantiene in memoria i risultati finali dell'algoritmo, l'indice indica l'elemento sincronizzato e il valore contenuto il granulo con cui è stato sincronizzato.

`ArrayEnergiaCritica` è un array di numeri reali di dimensione pari al numero di granuli presenti nel file MP3. Questo array contiene i valori della funzione (1) relativi ad una specifica nota e a tutti i granuli del file MP3. Lo stesso array è riutilizzato per tutte le note.

`TabellaPicchi` è una matrice di booleani con dimensioni pari a:

(numero note fondamentali) * (numero granuli nel file MP3 + 1), intendendo con numero note fondamentali le note contenute nell'ottava base cioè 12. Questa tabella contiene il valore true in una colonna N nel momento in cui il valore contenuto nell'`ArrayEnergiaCritica`, relativo ad un certo granulo (N-1) e ad una certa nota, supera il valore della soglia. La prima colonna di `TabellaPicchi` è un flag che se settato a true permette al controllo di non verificare ancora la nota associata in quanto già analizzata, permettendo così di risparmiare molte iterazioni inutili.

Il metodo `FillArrayEnergiaCritica(nota)` ha il compito di calcolare la funzione (1) relativamente ad una nota data e di inserire questi valori in `ArrayEnergiaCritica`.
Il suo codice è :

```
public void FillArrayEnergiaCritica ( Verticalizzatore.Note nota)
{
  CodecOut.OpenSpectrumFile(); // INIZIALIZZA IL FILE CONTENENTE LO SPETTRO
  for(int i = 0; i < CodecOut.NumberOfFrame; i++)
  {
    CodecOut.PutSpectrumToBuffer(i); //CARICA L'MDCT DI UN GRANULO
    CodecOut.CriticalBandEnergy(i);//I VALORI ASSOLUTI MDCT DIVENTANO RELATIVI
    ArrayEnergiaCritica[i] = EnergiaCritica(nota);
  }
  CodecOut.CloseSpectrumFile();
}
```

Questo metodo apre il file dello spettro MDCT (`OpenSpectrumFile`), carica in memoria i valori relativi ad un solo granulo e li eleva al quadrato trasformandoli in energie (`PutSpectrumToBuffer`). Tramite il metodo `CriticalBandEnergy` trasforma i valori di energia assoluti contenuti in `freq` (acceduti tramite `spectrum`) in valori relativi dividendoli per l'energia totale del granulo, il passo successivo calcola il valore della funzione (1) con le seguenti operazioni contenute nel metodo `EnergiaCritica(nota)` :

```
primaarmonica = NoteToFreq(nota , 1);
secondaarmonica = NoteToFreq(nota , 2);
terzaarmonica = NoteToFreq(nota, 3);

energiacritica += CodecOut.Spectrum[0,primaarmonica];
energiacritica += CodecOut.Spectrum[0,secondaarmonica];
energiacritica += CodecOut.Spectrum[0,terzaarmonica];
```

Questa funzione accede allo spettro del segnale nelle posizioni di interesse, relative alla nota presa in esame, tramite la funzione `NoteToFreq`, e restituisce la sommatoria di questi valori. Il metodo `FillArrayEnergiaCritica` inserisce il risultato di questa sommatoria in `ArrayEnergiaCritica`.

6.3.2 Descrizione del metodo main e funzionamento generale

Il `main` del prototipo anzitutto inizializza lo strato simbolico richiamando tramite `StratoSimbolico` i metodi sviluppati nel progetto `Verticalizzatore` e contenuti nella DLL "Verticalizzatore.dll". Successivamente il `main` inizializza lo strato audio richiamando tramite `CodecOut` i metodi sviluppati nel progetto `SpectrumAnalysis` e contenuti nella DLL "SpectrumAnalysis.dll".

Dopo queste fasi il programma ha a disposizione i dati relativi al risultato dell'operazione di verticalizzazione sulla partitura (vedi cap 6.2.1 Verticalizzatore) e i dati relativi allo strato audio (vedi cap 6.2.1 SpectrumAnalysis)

Terminate queste fasi di preparazione il main richiama i metodi contenuti in TabellaPitchToBanda tramite l'oggetto Bande creando così le tabelle necessarie alla gestione delle relazioni tra note, frequenze e valori MDCT(vedi cap 6.2.1 Sincronizzatore).
Dopo questa lunga fase preparatoria il main compie i passi seguenti:

- tramite un ciclo su tutti gli eventi in partitura calcola la funzione (2)
- trova i valori di soglia tramite (1) che rispettino i risultati di (2)
- applica questa soglia ai valori trovati
- inserisce i risultati nella tabella picchi
- applica un filtro mediano che sopprime le rilevazioni isolate in tabella picchi
- seleziona i risultati migliori per effettuare la sincronizzazione.

Andremo ora ad illustrare queste fasi nello stesso ordine in cui sono state elencate, nella pagina seguente è presentato il codice che descrive il ciclo principale dell'applicazione.

```
for ( int i = 0; i < NumberOfEvents; i++ ) // I INDICE DEGLI EVENTI TROVATI DA VERTICALIZZATORE
{
  for ( int j = 0; ( j < Test.StratoSimbolico.NumberOfVoices ); j++ ) // J INDICE DELLE VOCI IN PARTITURA
  {
    Element = ( Verticalizzatore.ElementOut )Test.StratoSimbolico.TableCompressed.tabella[j][i]; //ATOMO VERTICALE
    for ( int k = 0;( k < Element.Chord.Count ); k++ ) // K INDICE DELLE NOTE PRESENTI NEGLI ACCORDI
    {
      nota = ( Verticalizzatore.Note )Element.Chord[k]; // NOTA CHE SARÀ ESAMINATA

      if ( ( nota.NoteName != 'R' ) & ( !nota.Blacklisted ) ) // SCARTA LE PAUSE,  R => REST
        if ( !Test.TabellaPicchi[ Test.NoteToPos( nota ), 0 ] ) // SE TRUE NOTA GIÀ ESAMINATA
        {

          BlockTrue = 100; // BLOCKTRUE È UNA PERCENTUALE INIZIALIZZATA  A 100

          Test.FillArrayEnergiaCritica( nota ); //CARICAMENTO ENERGIA RELATIVA (NOTA)  IN MEMORIA

          Test.TabellaPicchi[ Test.NoteToPos( nota ), 0] = true; // FLAG SETTATO => NOTA ANALIZZATA

          MaxBlockTrue = Test.StratoSimbolico.NumberOfDivisionRelativePerNotes[Test.NoteToPos( nota )]*100;

          for ( double Soglia = SogliaMin; ( BlockTrue >= MaxBlockTrue ) && Soglia < 1; Soglia += 0.00005 )

                //BLOCKTRUE È UN VALORE PERCENTUALE DEL NUMERO DI GRANULI
                                                    CHE SUPERANO LA SOGLIA

            { // SOGLIA È L' ENERGIA PERCENTUALE MINIMA AFFINCHÈ UN BLOCCO SIA
                                    CONSIDERATO INTERESSANTE

              BlockTrue = 0;
              Test.GranuloCounter = 0;  // INDICE DEL GRANULO ESAMINATO

              for ( ;Test.GranuloCounter < Test.CodecOut.NumberOfGranuli; Test.GranuloCounter++ )
              {
                if ( Test.CodecOut.energy[ Test.GranuloCounter ] * 10 > Test.meanenergy )
                        //SCARTA I GRANULI CON ENERGIA MINORE DI 1/10 DEL VALORE MEDIO
                {
                  IsPicco = ( Test.ArrayEnergiaCritica[Test.GranuloCounter] > Soglia );
                                                    // INDICA CHE IL GRANULO CONTIENE UNA NOTA
                  Test.TabellaPicchi[ Test.NoteToPos( nota ), Test.GranuloCounter + 1 ] = IsPicco;

                  if ( IsPicco )
                    BlockTrue ++;  // SE  TROVATA  NOTA IN UN GRANULO BLOCKTRUE++

                }
              }

              BlockTrue = ( BlockTrue / Test.CodecOut.NumberOfGranuli ) * 100;// BLOCKTRUE PERCENTUALE

            } // FINE CICLO SU Soglia;    RELATIVO ALLA PERCENTUALE DI GRANULI SCARTATI
          } // ENDIF;    FALSE SE NOTA GIA ' ANALIZZATA
      } // FINE CICLO SU K RELATIVO ALLE NOTE DI UN ACCORDO
    } // FINE CICLO SU J DELLE VOCI IN PARTITURA
  } //FINE CICLO SU I DEI GRUPPI IN PARTITURA
```

Le frecce all'interno del codice individuano la struttura iterativa che scorre tutte le note presenti in ogni evento generato dal localizzatore, selezionando prima l'evento poi la voce ed eventualmente la nota all'interno di un accordo. La nota identificata è inserita nella variabile di appoggio `nota`. Per ogni nota trovata il sistema verifica che la nota non sia una pausa o una nota non sincronizzabile, dopodichè verifica che la relativa linea di `TabellaPicchi` non sia

stata già riempita e quindi la nota già esaminata. `BlockTrue` è impostata a 100. Questa variabile indica il numero di granuli per cui la funzione (1) restituisce un valore superiore alla soglia. Si caricano in memoria i risultati della funzione (1) relativi alla nota in esame su tutti i granuli del file MP3 e si setta il flag di `TabellaPicchi` che tiene conto delle note già analizzate. `MaxBlockTrue` è impostato al valore percentuale di *divisions* che contengono la nota considerata, coerentemente con la formula (2). Il ciclo successivo gestisce il valore di soglia cercandone il valore ottimale: il primo valore considerato è `sogliamin` che è impostato ad 1/576, questo valore è la media della distribuzione uniforme, si presume che un granulo contenente una certa nota abbia un valore maggiore nella rispettiva linea frequenziale rispetto ad una distribuzione uniforme di un generico segnale. Il valore di soglia è incrementato di 0.00005, un valore formalmente più corretto pretenderebbe una frazione del valore iniziale, ma sperimentalmente si è osservato che questo valore velocizza notevolmente l'algoritmo senza influire troppo sui risultati. Il ciclo termina e la `soglia` trovata è considerata ottimale se il numero percentuale di blocchi che superano questa soglia è minore o uguale della percentuale contenuta in `BlockTrue`. All'interno del ciclo appena descritto il sistema analizza tutti i granuli presenti nel file MP3. Scarta i granuli che hanno un'energia assoluta minore di 1/10 dell'energia media del file MP3. Questa condizione è necessaria in quanto la formula (1) ha proprio l'energia totale del granulo al denominatore, di conseguenza se l'energia totale del granulo assume valori che tendono a zero il rapporto tende ad infinito. Questa caratteristica è molto evidente nella figura 5.13 dove si ha una serie di picchi molto elevati alla fine del brano, quando il segnale è energicamente quasi nullo. Per i granuli energicamente interessanti è calcolata la variabile booleana `IsPicco` che assume un valore di verità se la formula (1) è maggiore di `soglia` per il granulo e la nota in esame. Il valore della variabile `IsPicco` è copiata in `TabellaPicchi`. Se il granulo preso in esame contiene la nota cercata (`IsPicco` è vera) la variabile `BlockTrue` è incrementata. Terminato il ciclo che scorre tutti granuli si trasforma in percentuale il valore di `BlockTrue` rispetto al totale dei granuli presenti e si verifica che questo valore sia minore del numero massimo di *BlockTrue* possibili (sempre in accordo con la formula (2). In questo caso il ciclo termina restituendo il risultato dell'ultima operazione di sogliatura effettuata.

La fase successiva applica a `TabellaPicchi` un filtro mediano.

Il filtro mediano definito dal metodo `MedianFilter010To000` elimina i picchi isolati restituiti dall'operazione di sogliatura e "fonde" i picchi separati da un solo granulo. Il filtro mediano è un filtro non lineare utile per la soppressione del rumore di tipo impulsivo; è

comunemente utilizzato in elaborazione delle immagini per eliminare i cosiddetti rumori "sale & pepe" [7]. Il suo funzionamento è basato sullo scorrimento di una finestra. Sono letti i valori compresi nella finestra e in uscita viene sovra scritta la "mediana" dei valori letti nella posizione centrale della finestra. La mediana di una sequenza discreta a1,a2,...,aN è l'elemento tale che (N-1)/2 elementi hanno valore minore/uguale ad esso, mentre i rimanenti (N-1)/2 elementi hanno valore maggiore/uguale [6]. In questo caso se associamo a "true" il valore 1 e a "false" il valore 0 i valori considerati sono sequenze composte da 0 e 1. La finestra considerata ha ampiezza 3 di conseguenza ponendo in ordine crescente le possibili sequenze di valori si ottengono i seguenti insiemi: {0,0,0}, {0,0,1}, {0,1,1}, {1,1,1}. In base alla definizione il filtro mediano lascia inalterate le sequenze composte da soli 0 e da soli 1. Restituisce il valore 0 in caso di sequenze composte da {0,0,1} questo comportamento cancella i picchi isolati modificando la sola sequenza {0,1,0} in {0,0,0}. Il filtro restituisce valore 1 con le sequenze {0,1,1} fondendo eventuali picchi separati da una sola rilevazione negativa modificando le sole sequenze {1,0,1} in {0,0,0}. Questo filtro è applicato su tutta la tabella `TabellaPicchi` e quindi elimina le rilevazioni positive che hanno come vicini rilevazioni negative e fonde rilevazioni positive separate da una sola rilevazione negativa. Questo filtro rende più affidabili i risultati in `TabellaPicchi` in quanto appare poco probabile la presenza di note che abbiano durate di un solo granulo.

L'ultima fase è rappresentata dal metodo `Select`. Questo metodo scorre tutte le note presenti negli eventi rilevati dal verticalizzatore secondo lo schema identificato dalle frecce nell'estratto di codice precedente. Per ogni nota questa funzione scorre la relativa linea di `TabellaPicchi` (che indica per ogni granulo la presenza o meno della nota data), cerca il primo granulo in cui si ha una rilevazione positiva e lo seleziona come punto di inizio dell'evento considerato. Al momento di una selezione scrive nel file testo di output (`OutSync.txt`): l'indice dell'evento considerato, l'istante di inizio rilevato e il granulo associato a questo istante. Quando il ciclo passa all'evento successivo la ricerca del punto di sincronizzazione prosegue dal granulo successivo a quello identificato come punto di sincronizzazione dell'evento precedente. Mentre scorre tutti i granuli il metodo `Select` scrive su di un file testo (`Experimental.txt`) il valore 1 in caso si abbia un punto di sincronizzazione in quel granulo, 0 per i granuli che non sono punti di sincronizzazione. Questo file è stato utilizzato per generare le figure presenti nel capitolo 7.

Capitolo 7

Test, conclusioni e sviluppi futuri

7.1 Test

Per valutare correttamente un algoritmo di AMS è necessario avere a disposizione un gran numero di partiture di cui siano noti gli istanti di sincronizzazione con la relativa esecuzione [3]. Lo scopo di questa fase di test, però, non è tanto dimostrare l'accuratezza dell'algoritmo presentato, quanto evidenziare la validità dell'innovativo metodo di analisi di basso livello proposto nei capitoli 5.3.1 e 5.2.1 e sintetizzato dalle formule (1) e (2). Non saranno quindi presentate percentuali inerenti il numero di note sincronizzate correttamente o il numero dei brani testati su cui l'algoritmo è andato a buon fine. Piuttosto, in questo paragrafo, utilizzando i risultati relativi ad un solo esempio, ma significativo di tutti i test effettivamente svolti, valuteremo la qualità degli algoritmi proposti e daremo una spiegazione per gli errori generati. Nelle immagini seguenti la parte più scura rappresenterà la forma d'onda del segnale analizzato mentre le linee più chiare saranno relative ai punti di sincronizzazione.

7.1.1 Tipologia brano scelto

Il brano esaminato è "Prélude sur un thème de Edvard H. Grieg" composto da Luca Andrea Ludovico. È un brano a 4 voci per solo pianoforte contenente 24 battute da 4/4, una variazione di tonalità e due tempi differenti: il primo "sereno" e il secondo "più mosso e deciso". Sono state esaminate due differenti esecuzioni di questo brano: la prima, utilizzata nei test descritti in 7.1.1.1 è stata generata tramite un sequencer MIDI, estraendo dalla partitura in esame solo la prima voce e utilizzando il timbro di pianoforte 1 nello standard general MIDI; la seconda interpretazione è invece una registrazione in presa diretta di una esecuzione pedissequa della partitura, poiché è stata registrata in presa diretta, l'esecuzione presenta un rumore di fondo non trascurabile.

7.1.1.1 Test di sincronizzazione

Il test si è svolto codificando l'esecuzione MIDI del brano in esame tramite l'encoder Lame e decodificandolo con la sua versione modificata. La figura 7.1 mostra la forma d'onda completa del brano nella parte più scura, mentre le linee più chiare mostrano i punti di sincronizzazione selezionati per gli eventi presenti in partitura. Purtroppo non è stato possibile inserire la partitura nella stessa immagine per evidenti ragioni grafiche. Come si osserva da questa figura il prototipo offre dei risultati errati sopratutto nella parte centrale del brano, dove non sono evidenziati punti di sincronizzazione. È però interessante notare il comportamento del sistema nelle battute iniziali, dove sono presenti vari punti di sincronizzazione. L'immagine 7.2 rappresenta un ingrandimento della prima parte dell'immagine 7.1. Questa immagine (7.2) permette di evidenziare la correttezza del sistema nelle prime battute del brano considerato.

Figura 7.1 I risultati del prototipo sul file di test scelto

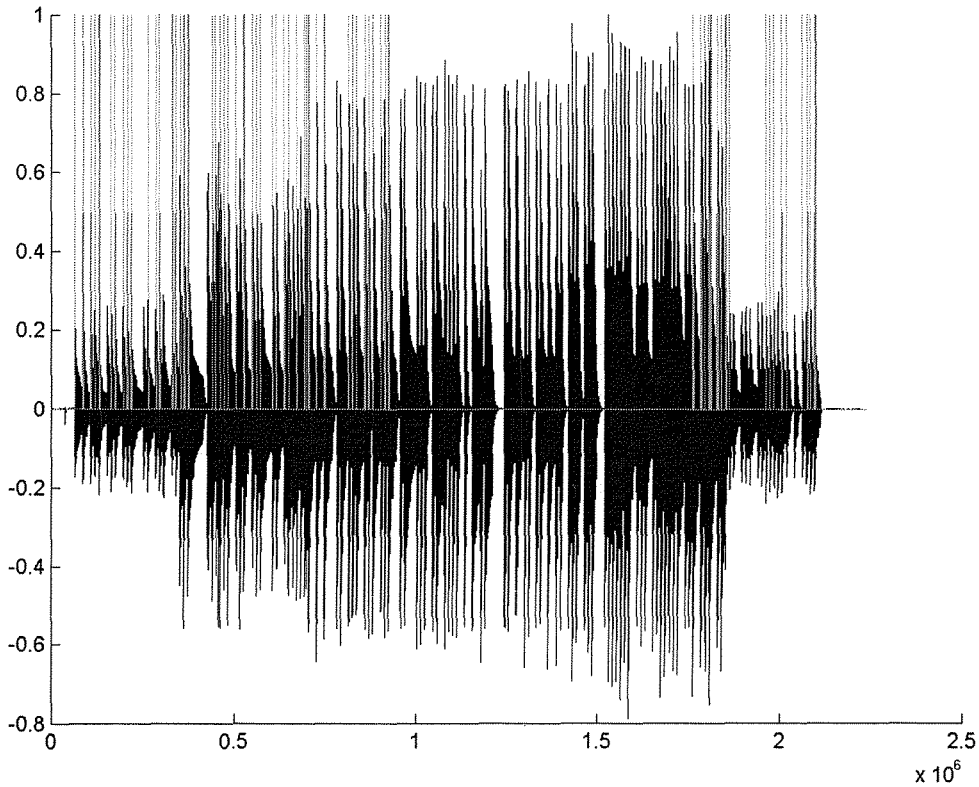
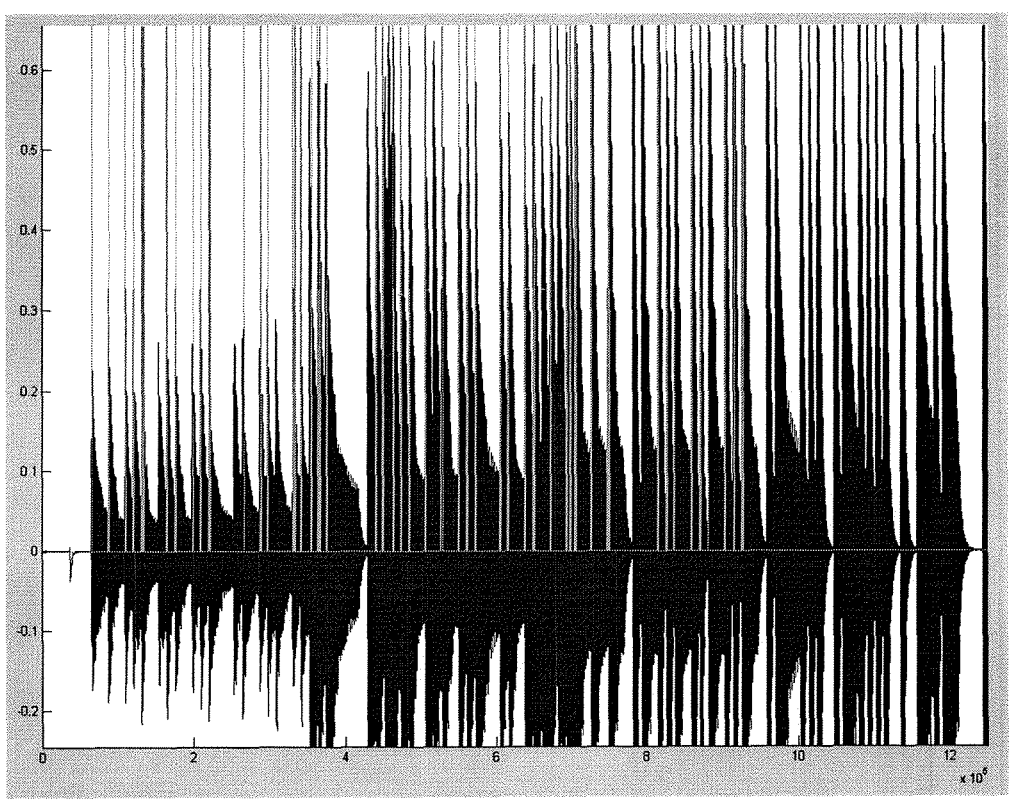


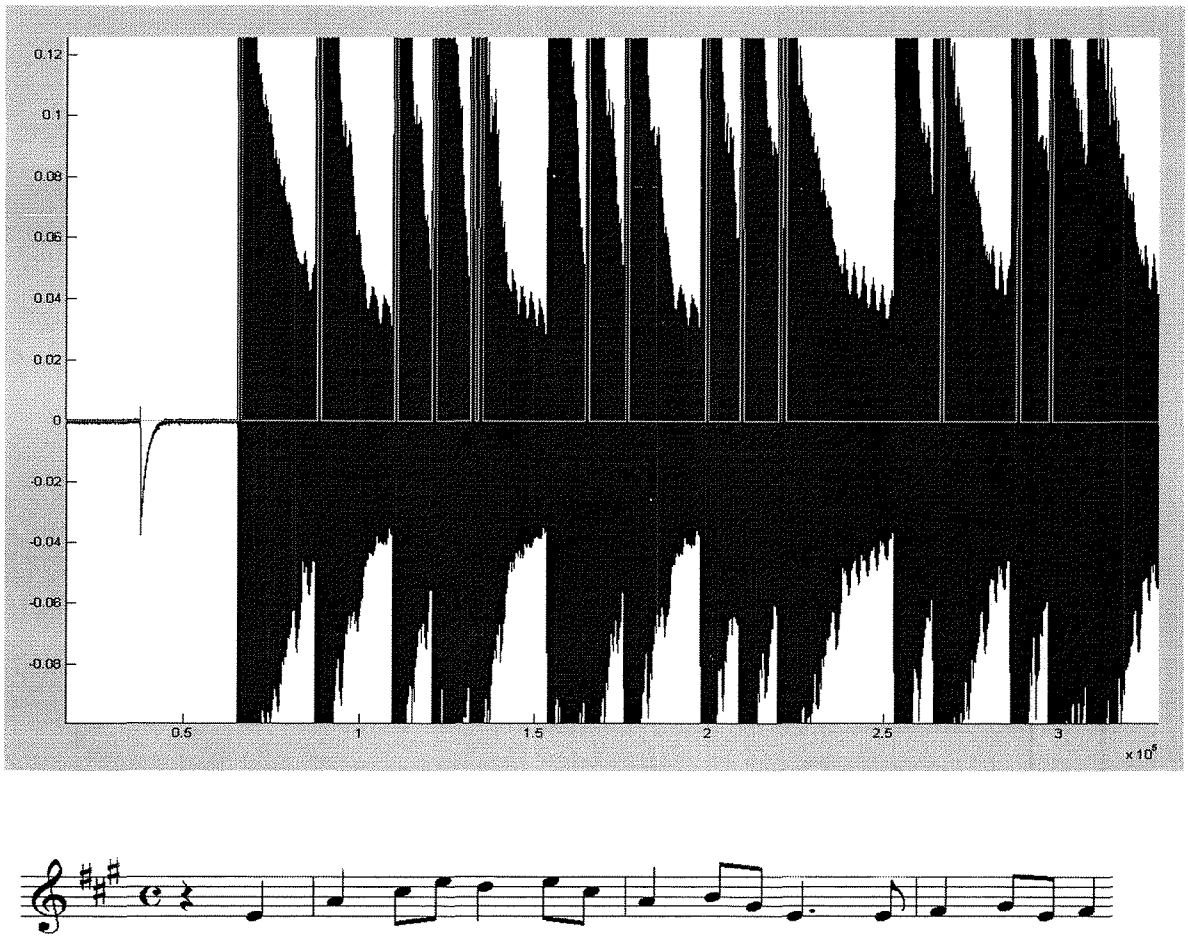
Figura 7.2 Ingrandimento della parte iniziale della figura 7.1



Il degradamento delle prestazioni del sistema, osservato in figura 7.1, nell'individuazione dei punti di attacco in funzione dell'aumentare del tempo è imputabile all'assenza di una vera parte di "valutazione intelligente" dei risultati trovati. Questa mancanza fa scegliere, infatti, punti di sincronizzazione palesemente errati, come evidenziato in figura 7.3. Analizzando più approfonditamente il comportamento dell'algoritmo si osserva che la perdita di qualità dei risultati è causata dalla scelta sequenziale dei punti di sincronia e non dalla poca accuratezza dell'analisi di basso livello. Una spiegazione di questo comportamento è fornita dall'esempio seguente: supponiamo che il sistema ponga l'ancora di sincronizzazione di un evento i in un istante di tempo errato. Questa imprecisione si ripercuoterà sulla ricerca dell'evento i+1, perché il sistema inizierà a valutare istanti di tempo palesemente errati. Questa situazione potrebbe far rilevare, ad esempio, la presenza dell'evento i+1 in un istante di tempo, successivo a quello reale, semplicemente perché nel file audio è effettivamente presente la nota considerata in i+1 proprio nell'istante trovato, solo che la presenza di questa nota potrebbe, in realtà, essere dovuta ad un evento successivo presente in partitura che ha semplicemente la stessa altezza. Di conseguenza, in caso di errore, il sistema può generare una serie di false sincronizzazioni a catena, in cui ogni evento sincronizzato malamente fa aumentare la probabilità di errore per l'evento successivo. Questa retroazione positiva dell'errore porta in confusione il sistema ed è la causa principale delle imprecisioni mostrate in figura 7.1. Inoltre, come evidenziato nel paragrafo 7.1.1.2, l'analisi di basso livello non è esente da errori perché tende a trovare più note rispetto a quelle realmente presenti in partitura generando dei falsi allarmi che possono confondere un sistema che scelga gli istanti di sincronizzazione senza alcun tipo di controllo su queste scelte.

Un ulteriore limite è rappresentato dai casi di ambiguità imputabili all'MDCT (vedi capitolo 6) in quanto il sistema non ha modo di valutare il risultato più attendibile. I possibili algoritmi di valutazione dei risultati che permetterebbero di risolvere i problemi esposti sono presentati nel paragrafo 7.2. Le frecce in figura 7.2 mostrano due ancore di sincronia che sono in evidente conflitto; è infatti improbabile che due attacchi siano separati da soli due granuli. Un algoritmo di valutazione può facilmente risolvere il problema. Si è scelto di non inserire una fase valutativa per ottimizzare al meglio i metodi relativi all'analisi del segnale di basso livello. Questo per avere una solida base su cui poi implementare i vari metodi di valutazione, sfruttando, eventualmente, anche algoritmi noti che sono normalmente utilizzati in PCM [3][9][59][60]. La fase di test successiva è dedicata all'analisi di basso livello e mostra i buoni risultati offerti dalla soluzione implementata per questa parte del problema.

Figura 7.3 Inizio del brano di esempio con relativa partitura

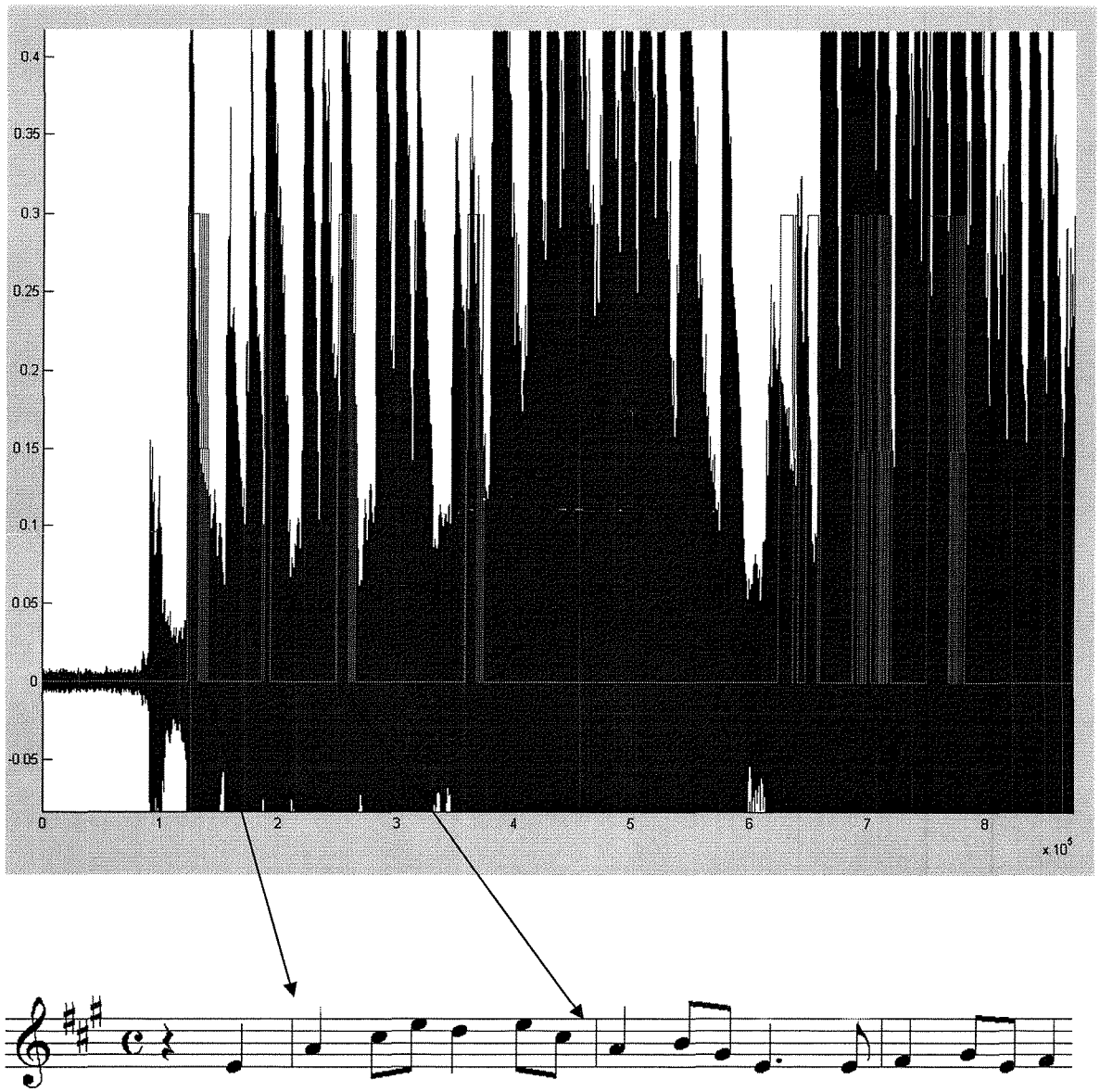


7.1.1.2 Test analisi di basso livello

Questa serie di Test si è svolta partendo da un esecuzione reale del brano in esame. Lo scopo di questi test è mostrare che l'andamento dell'operazione di soglia implementata riesca effettivamente ad identificare le note presenti in partitura. Nella figura 7.4 la linea più chiara indica con valori pari a 0,3 la presenza della nota analizzata, con zero la sua assenza. Per ovvi motivi grafici è stata riportata solo la voce numero uno e non la partitura completa.

Questa funzione rappresenta il contenuto della matrice `TabellaPicchi` dopo l'esecuzione di tutti i passaggi descritti in 6.3.2 ad esclusione del metodo `select`. È presentato come esempio il risultato delle operazioni svolte relativo alla nota La.

Figura 7.4 Contenuto di `TabellaPicchi` relativo alla nota La



Come si osserva dalla figura 7.4 la presenza della nota La è sempre identificata correttamente, i falsi allarmi presenti sono imputabili a due categorie di problemi distinte: la prima relativa alla partitura in esame, la seconda attribuibile al formato MP3. Nella prima categoria rientrano i problemi specifici della partitura. In questo caso il brano presenta un La come nota di riferimento per l'armonia in tutte le battute presenti in figura 7.4, e questo genera un innalzamento dei valori energetici della nota considerata portando a riconoscimenti errati. La seconda categoria è invece derivata direttamente dal formato MP3, precisamente dalla bassa risoluzione spettrale utilizzata dall'MDCT che crea delle sovrapposizioni molto frequenti tra note adiacenti. Bisogna inoltre considerare i problemi dovuti al mascheramento di note in relazione armonica come descritto in [15] che sono sfruttati, e quindi aumentati, dal

formato MP3 come descritto nel capitolo 2. In sostanza l'algoritmo di valutazione di basso livello implementato individua praticamente sempre la corretta posizione della nota in esame, ma oltre a questi risultati reali, genera un numero di falsi allarmi pari al numero delle note cercate. La generazione di falsi allarmi è un problema tipico delle fasi che si occupano dell'analisi di basso livello per fare AMS[3][9][59][60]. Confrontando la quantità di false individuazioni prodotte dall'approccio proposto con l'algoritmo presentato in [60] si osserva un comportamento favorevole alla soluzione esposta in questo lavoro. Di conseguenza è lecito ipotizzare che un algoritmo di valutazione dei risultati ottenuti riesca a ben dominare i falsi allarmi presenti, permettendo al sistema di effettuare delle sincronizzazione valide.

7.2 Conclusioni e sviluppi futuri

Come è facilmente intuibile dal capitolo 4, l'analisi dei segnali MP3 è ancora ad uno stadio embrionale. Il primo problema che si presenta utilizzando questo standard in un contesto dedicato all'analisi del segnale, è l'assenza pressoché totale di funzioni che permettano di estrarre dei dati validi dalle informazioni contenute in questo formato. La mancanza di tool di analisi è imputabile a due fattori distinti: la relativa giovinezza di questo formato, che è divenuto uno standard ufficiale ISO da meno di 15 anni, e la sua particolare rappresentazione del segnale audio, che rende inapplicabili buona parte delle funzioni comunemente usate nel trattamento dell'informazione PCM. In MP3, infatti, non si ha la possibilità di applicare filtri; eventuali trasformate dal dominio del tempo a quello delle frequenze sono inutili essendo il dominio del tempo irraggiungibile a meno di effettuare una decodifica di buona parte del file considerato.

Di conseguenza, questa situazione impone a chiunque intenda fare analisi sulle informazioni audio contenute in questo formato, di gestire i dati di un'unica trasformata MDCT, che ha parametri imposti dallo standard che non permettono alcuna variazione. I dati dell'MDCT devono essere adattati forzatamente allo scopo preposto anche se, come nel caso di algoritmi che svolgono AMS, male si adeguano agli obbiettivi da raggiungere.

Gran parte di questo lavoro di tesi è stato, dunque, dedicato proprio alla creazione degli strumenti necessari per valutare efficacemente i dati del formato MP3 cercando di fornire strumenti per l'analisi di base che offrano dei risultati tecnicamente validi. Osservando gli algoritmi proposti nel capitolo 4 [3][8][59][60] appare subito chiaro come la reale innovazione portata dagli algoritmi citati sia quasi completamente racchiusa nella parte dedicata alla valutazione dei risultati offerti dall'analisi, appoggiandosi quest'ultima su conoscenze ben note in ambito DSP. In un contesto legato ad MP3 l'innovazione è già nella fase di gestione dei dati, nella scelta e nell'implementazione di funzioni che siano effettivamente valide per fornire dati per risolvere il problema studiato. Con questo lavoro di tesi si è quindi cercato di fornire degli strumenti validi per l'analisi di questo formato per i problemi di AMS. Questo obbiettivo è stato complessivamente raggiunto, come evidenziato anche dallo studio dei test. Dalla fase di prova del prototipo appare subito chiaro che i risultati non brillanti sono legati al primitivo algoritmo di selezione implementato e indipendenti dai risultati offerti dalle formule (1) e (2). Anzi, è proprio grazie alla validità di queste formule che il prototipo riesce a generare dei risultati validi anche in assenza di un vero e proprio

algoritmo intelligente di controllo dei dati raccolti. Da queste considerazioni è lecito proporre una fase di studio successiva che parta dai risultati raccolti in questo lavoro e arrivi ad un algoritmo di sincronizzazione valido, implementando un metodo di valutazione dei dati di buon livello. Si propongono 2 possibili approcci per questa futura implementazione: il primo algoritmo potrebbe essere impostato su un adattamento del lavoro svolto dal D'Onofrio [3] in PCM. Questo algoritmo dovrebbe trovare delle ipotesi, adatte al formato in esame, per stimare l'affidabilità delle ancore di sincronizzazione trovate, permettendo, quindi, di suddividere il problema in tanti sotto problemi più semplici che potrebbero essere risolti ricorsivamente. È da notare che la formula che calcola l'indice di affidabilità di un istante di sincronizzazione utilizzata in [3] non è direttamene applicabile sui coefficienti MDCT a causa dell'alta variabilità dei dati e della bassa risoluzione spettrale che la trasformata utilizzata in MP3 offre. Un secondo approccio, molto diverso dal precedente, potrebbe prendere in considerazione i risultati raccolti dal prototipo nella tabella dei picchi e considerare questa matrice nel suo insieme generando dei possibili istanti di sincronizzazione relativi a tutte le note possibili e non più legati alla partitura. Nella fase successiva basterebbe implementare un algoritmo di *string matching* tra i dati presenti in partitura e i dati raccolti che darebbe di scartare le ancore errate. Entrambi gli algoritmi proposti permetterebbero di superare i problemi evidenziati nella fase di test rendendo l'algoritmo complessivo competitivo con le soluzioni proposte in ambito PCM. Gli algoritmi di AMS in formato non compresso normalmente lavorano analisi frequenziali molto strette, circa 5 Hz, di conseguenza un La centrale accordato, ad esempio, a 420 o a 460 Hz porta gli algoritmi di analisi a non dare più una risposta massima nelle frequenze fondamentali delle note provocando una serie di mancate rilevazioni. Un algoritmo che lavori su MP3 non avrebbe alcuna conseguenza pratica da un'accordatura imprecisa perché le linee frequenziali dell'MDCT hanno una risoluzione in frequenza così bassa da non risentire particolarmente di leggere variazioni nelle frequenze fondamentali delle note.

I buoni risultati evidenziati dalle formule (1) e (2) rendono plausibile la proposta di un loro utilizzo in ambito PCM. È sensato immaginare che queste formule se applicate sui risultati di una trasformata tempo frequenza studiata appositamente per gli algoritmi di AMS dovrebbero fornire dei risultati migliori. Questa ipotetica trasformata dovrebbe avere sicuramente molte più linee frequenziali (almeno 2048) per superare i limiti imposti dall'MDCT dello standard MP3. L'algoritmo di analisi proposto in questo lavoro di tesi fa riferimento alle frequenze fondamentali delle note basandosi sempre sull'ottava della prima nota trovata. Questa implementazione è stata scelta per limitare il massimo numero di note

analizzabili a 12 (l'altezza delle note indipendentemente dall'ottava di riferimento). Questa scelta permette di contenere il numero massimo di esecuzioni del pesante ciclo presentato nel capitolo 6.3.2 a 12. Questa caratteristica ovviamente limita la qualità dell'analisi di basso livello, sarebbe quindi interessante ottimizzare il codice per eliminare questo vincolo rendendo l'analisi indipendente per ogni ottava.

Una breve considerazione sul modulo Verticalizzatore. I dati generati da questo modulo possono avere anche una utilità pratica se decontestualizzati dal problema della sincronizzazione. Ad esempio, tra i vari output è possibile visualizzare il tempo di partitura occupato dalle note con un certo nome, ad esempio, potrei dire che il 20% dei quarti (se il tempo prevede una suddivisone in quarti) è occupato da Mi oppure che il 16% degli ottavi (se il tempo è suddiviso in ottavi) è occupato dalla nota La. Informazioni di questo tipo possono risultare utili per valutare, ad esempio, i diversi stili compositivi tra i vari autori. Potrebbe, quindi, essere interessante cercare di completare ed espandere il modulo verticalizzatore con il preciso scopo di compiere analisi della partitura fine a se stessa.

Appendice A

Breve manuale utente

Per gestire correttamente questo prototipo è anzitutto necessario capire quali siano i file di input e di output. I file di input per il prototipo realizzato sono:

- `HybridINVerticale3.txt`
- `HybridINVerticale3eng.txt`
- `BlockType.txt`
- `MixedBlockFlag.txt`

I primi 2 file contengono i valori dello spettro MDCT. Ogni valore è inserito su una nuova linea e i granuli sono separati da una serie di caratteri di escape. I due file si differenziano per la gestione del punto e della virgola fornendo il primo valori con la notazione italiana (la virgola identifica i decimali) e il secondo con la notazione inglese (i decimali sono preceduti da un punto).

Il file `BlockType.txt` contiene il tipo di finestratura utilizzato per ogni granulo. Anche in questo caso ogni valore è inserito su una nuova linea, non ci sono caratteri di escape. I valori possibili presenti in questo file sono 4: 0 per i long block, 2 per gli short block , 1 e 3 per i blocchi di transizione rispettivamente 1 per i long-to-short e 3 per gli short-to-long.

Il file `MixedBlockFlag.txt` contiene l'informazione di codifica dei coefficienti MDCT, il valore 0 indica la codifica standard il valore 1 indica una codifica mista.

I seguenti file sono invece generati dal programma:

- `outsync.txt`
- `Experimental.txt`
- `Nota.txt`

Il primo file contiene i risultati della sincronizzazione mostrando per ogni linea il numero dell'evento sincronizzato, il tempo di sincronizzazione in secondi e il granulo in cui è stato trovato l'evento.

`Experimental.txt` contiene valori 0 e 1 pari al numero dei granuli presenti nel file MP3. il valore 0 indica i granuli che non sono stati scelti come istanti di sincronizzazione di nessun evento viceversa il valore 1 identifica i granuli che sono stati legati ad un evento come istanti iniziali.

Il file `Nota.txt` contiene invece l'andamento della funzione (1) rispetto ad una nota selezionata dall'utente.

I file `Nota.txt` e `Experimental.txt` sono stati utilizzati per produrre i grafici presentati nella fase di test (cap 7.1) e alcuni dei grafici presentati nel capitolo 5.

Tramite Matlab, infatti, sono stati scritti dei semplici script che integrano il prototipo fornendo la possibilità di visualizzare il segnale audio rispetto a: la funzione (1), l'energia, i punti di sincronizzazione, il risultato successivo al filtro mediano ecc. Questi dati sono contenuti di volta in volta in `nota.txt`. Utilizzando invece i file di input del sistema è possibile visualizzare il block type utilizzato in ogni granulo, e il valore assunto dall'informazione di mixed block flag.

Per selezionare quale dato visualizzare si deve modificare direttamente il nome del file nel codice di questi script utilizzando `nota.txt` per i valori calcolati da Sincronizzatore e i rispettivi file di input per i valori generati dal Lame.

Il contenuto del file `nota.txt` è impostato direttamente dal progetto Verticalizzatore. La selezione del tipo di contenuto può essere fatte direttamente all'interno del codice abilitando o disabilitando alcuni flag.

I file di input sono generati automaticamente dal Lame tramite la seguente riga di comando:

```
lame --decode [nomefile.mp3]
```

questa è la comune riga di comando per decodificare i file MP3 in WAV tramite Lame [38].

Il progetto Verticalizzatore non necessita di alcun parametro di input in quanto i file di input devono essere presenti nella directory di esecuzione e il nome del file MusicXML contenente la partitura è richiesto dopo l'avvio dell'esecuzione.

Bibliografia

[1] MPEG Requirements Group, "MPEG-7 Context, objective and technical roadmap", Doc. ISO/MPEG N2861, International Organization for Standardization, Vancouver Luglio 1999.

[2] J. Stephen Downie "Music Information Retrieval Annotated Bibliography Website Project, Phase I" Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, 2001

[3] A. D'Onofrio "Metodi e prototipazioni software per il trattamento digitale integrato di processi musicali audio e testuali temporizzati" Tesi Di Laurea, Università degli Studi di Milano, 1999

[4] Henrique B. S. Leão, Germano F. Guimarães, Geber L. Ramalho and Sérgio V. Cavalcante "Benchmarking Wave-to-MIDI Transcription Tools" VII Brazilian Symposium on Computer Music, Curitiba, Brazil, 2000

[5] J. D. Reiss and M. B. Sandler "Nonlinear time series analysis of musical signals", Proc. of the 6th Int. Conference on Digital Audio Effects (DAFX-03), London, UK, 2003.

[6] A.M. Mood, F.A. Graybill, D.C. Boes "Introduzione alla Statistica", McGraw-Hill editori 1991

[7] A. R. Weeks "Fundamentals of Electronic Image Processing", SPIE & IEEE Press 1998.

[8] S. Galbiati "Metodi e strumenti software per l'individuazione automatica di corrispondenze temporali tra parti strumentali e audio orchestrale" Tesi Di Laurea, Università degli Studi di Milano, 2001

[9] WIDISOFT "widisoft a WAV to MIDI converter" www.widisoft.com

[10] IMS "Intelliscore a WAV to MIDI converter" www.intelliscore.net

[11] S. Vidili "Musica Digitale: la codifica del segnale audio secondo lo standard MP3" Monografia di diploma universitario, Politecnico di Torino, 1999

[12] D. Bagni "Teoria dell'Informazione: codifica dell'informazione multimediale (MIT: Multimedia Information Theory), 2001

[13] G. Tzanetakis G. Essl P. Cook "Automatic Musical Genre Classification Of Audio Signals" International Symposium on Music Information Retrieval 2001, Indiana University, Bloomington, Indiana, USA, 2001

[14] G. Vercellesi "Metodi e prototipi software per l'elaborazione diretta di informazione audio in formato compresso Mp3" Tesi Di Laurea, Università degli Studi di Milano, 2003

[15] A. Frova "Fisica nella musica" Zanichelli Editore,1999

[16] K. Salomonsen, S. Søgaard, E. Proft Larsen, S. Ditlev Sørensen "Design and Implementation of an MPEG / Audio Layer 3 Bitstream Processor", Aalborg University – Institute of Electronic System – Department of Communication Technology, 2002

[17] M. Sieler, R. Sperschneider "MPEG Layer 3 Bitstream Syntax and Decoding – no Multichannel audio, no Multilingual audio"

[18] S. Kim, Yi Li, H. Kim, H. Choi, Y. Jang "Real Time MPEG-1 Audio Encoder and Decoder Implemented on a 16-bit Fixed Point DSP", DSP Team, Micro Device Business, Semiconductor Division, Samsung Electronics Co., Ltd, 1999

[19] M. Luise, G. M. Vitetta, "Teoria dei Segnali" Mc Graw-Hill, Milano, 1999

[20] C. Marves, G. Ewers, "A Simple Approach to Digital Signal Processing" Wiley April 1996

[21] B.G. Lee, "A new Algorithm to Compute the Discrete Cosine Transform", *IEEE Trans. ASSP* ASSP-32, 6, 1984

[22] Th. Sporer, Kh. Brandenburg, B. Edler "The use of multirate filter banks for coding of high quality digital audio", 6th European Signal Processing Conference (EUSIPCO), Amsterdam, Vol, 1 pages 211 – 214, June 1992

[23] F. Visciotti "Tecniche di Compressione Audio: Evoluzione dello Standard MPEG", Università degli Studi di Bologna – Facoltà di Ingegneria, 2001

[24] D. Yen Pan "Digital Audio Compression", Digital Technical Journal Vol. 5 No.2 Spring 1999

[25] K. Brandenburg, T. Sporer "NMR and Masking Flag: Evaluation of Quality Using Perceptual Criteria", 11th AES Int. Conf. Portland, 1992, pp. 169-179, 1992

[26] F. Baumgarte, C. Ferekidis, H. Fuchs, "A Nonlinear psychoacoustic Model Applied to the ISO MPEG Layer 3 Coder", Institut für Theoretische Nachrichtentechnik und Informationsverarbeitung Universität Hannover, Germany, 1995

[27] T. Painter, A. Spanias "A Review of Algorithms for Perceptual Coding of Digital Audio Signals", proceedings of the IEEE, vol. 88, no. 4, April 2000

[28] David J M Robinson, Malcolm O J Hawksford, "Psychoacoustic Models and Non-Linear Human Hearing", AES 109th convention, Los Angeles, USA, September 22-25, 2000

[29] L.A. Ludovico "Manuale MX, Versione 1.5", 2005
http://www.lim.dico.unimi.it/didatt/materiali/ManualeMX.doc

[30] "MusicXML 1.0 Tutorial", 2005 http://www.recordare.com/xml/musicxml-tutorial.pdf

[31] G. Zoia, Ruo-hua Zhou, M. Matavelli "MPEG Audio Coding and XML: Samples, models, Descriptors", MAX 2002 Music Application using XML, September 19-20, 2002

[32] G. Haus, M. Longari, "Towards a Symbolic / Time Based Music Language Based on XML", MAX 2002 Music Application using XML, September 19-20, 2002

[33] Elide Rusty Harold, "XML Bible", IDG Books, 2001

[34] Raccomandazione del W3C "Extensible Markup Language (XML) 1.0", http://www.w3.org/XML.

[35] W. B. Hewlett "MuseData: multipurpose representation. In Beyond Midi: the Handbook of Musical Codes", E. Selfridge-Field, Ed. MIT Press, Cambridge, MA, 402-447, 1997

[36] D. Huron "Music Research Using Humdrum: A User's Guide. Stanford, California: Center for Computer Assisted Research in the Humanities", 1999
http://dactyl.som.ohio-state.edu/Humdrum/guide.toc.html

[37] L. Rossi "Teodia Musicale" Edizioni Carrara, 1987

[38] J. S. Downie (2001a). "The music information retrieval annotated bibliography project, phase I". Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR 2001), 2001

[39] Downie, J. Stephen. "Music information retrieval (Chapter 7)". In Annual Review of Information Science and Technology 37, ed. Blaise Cronin, 295-340. Medford, NJ: Information Today, 2003.

[40] E. Allamanche, J. Herre, O. Hellmuth, B. Fröba, T. Kastner, M. Cremer "Content based identification of audio materials using MPEG-7 low level description". Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR 2001), 197–204, 2001

[41] J. P. Bello, G. Monti, M. Sandler "Techniques for automatic music transcription". Proceedings of the 1st Annual International Symposium on Music Information Retrieval (ISMIR 2000), 2000

[42] R.E. Prather "Harmonic analysis from the computer representation of a musical score" Commun. ACM 39, 12es, 239, December 1996

[43] E. Chew "Towards a Mathematical Model of Tonality" Ph.D. thesis, Massachusetts Institute of Technology, 2000

[44] E. Chew, Yun-Ching Chen "Real-Time Pitch Spelling Using the Spiral Array" Computer Music Journal, Vol. 29, Issue 2 - Summer 2005

[45] Y. Wang, M. Vilermo "A compressed domain beat detector using MP3 audio bitstreams". In Proceedings of the Ninth ACM international Conference on Multimedia (Ottawa, Canada, September 30 - October 05, 2001). MULTIMEDIA '01, vol. 9. ACM Press, New York, NY, 194-202, 2001

[46] W.A. Sethares, R.D. Morris, J.C. Sethares "Beat tracking of musical performances using low-level audio features" Speech and Audio Processing, IEEE Transactions on Volume: 13, Issue: 2, pp: 275- 285, March 2005

[47] M. Goto "An Audio-based Real-time Beat Tracking System for Music With or Without Drum-sounds" Journal of New Music Research Vol. 30, No. 2, pp. 159–171, 2001

[48] S. Dixon "Automatic Extraction of Tempo and Beat From Expressive Performances" in Journal of New Music research, Volume 30, Number 1, pp. 39-58(20) , March 2001

[49] N.Scaringella, G. Zoia "A real-time beat tracker for unrestricted audio signals" Signal Processing Institute EPFL, Lausanne, CH-1015 Switzerland, 2004

[50] R.B. Dannenberg, N. Hu "Pattern Discovery Techniques for Music Audio" In Journal of New Music research , Volume 32, Number 2, pp. 153-163, June 2003

[51] M. Noll "Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate". In Proceedings of the Symposium on Computer Processing ing Communications, pp. 779–797. Polytechnic Institute of Brooklyn, New York, USA, 1969

[52] A. Master "Speech spectrum modeling from multiple sources". Master's thesis, Cambridge University, Engineering Dept., Cambridge, England, 2000

[53] H. Kobayashi, T. Shimamura "A weighted autocorrelation method for pitch extraction of noisy speech". In Proceedings of the Acoustical Society of Japan,
pp. 343–344. Saitama University Urawa, Japan, 2005

[54] L.P. Clarisse, J.P. Martens, M. Lesaffre, B.De Baets, H.De Meyer, and M. Leman, "An auditory model based transcriber of singing sequences," in Proceedings of ISMIR, 2002, pp.171–174, 2002

[55] A. Pertusa ,J. M. Iñesta "Polyphonic music transcription through dynamic networks and spectral pattern identification" in Pattern Recognition Letters, 2004

[56] L. Gustavo, A. Martins, J. S. Ferreira "PCM to MIDI Transposition", In 112th AES-Convention, Munich, May 10-13, 2002.

[57] J. Forsberg "Automatic conversion of sound to the MIDI format" Speech, Music and Hearing, 1998

[58] K.N. Kim, U.P. Chong , J.H. Choi "CONVERSION FROM CD-DA FORMAT TO MIDI FORMAT MAINTAINING A SOUND QUALITY" In Science and Technology, 1999. KORUS '99. Proceedings. The Third Russian-Korean International Symposium,1999

[59] N. Hu R.B. Dannenberg, G. Tzanetakis "Polyphonic audio matching and alignment for music retrieval" In Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop 19-22, pp. 185- 188, 2003

[60] M. Muller, F. Kurth, T. Roder "TOWARDS AN EFFICIENT ALGORITHM FOR AUTOMATIC SCORE-TO-AUDIO SYNCHRONIZATION" in Proceedings of ISMIR 2004, 5th International Conference on Music Information Retrieval Audiovisual Institute, Universitat Pompeu Fabra Barcelona, Spain October 10-14, 2004

[61] E. Weinstein, M. Feder, A.V. Oppenheim "Multi-channel signal separation by decorrelation" In Speech and Audio Processing, IEEE Transactions on Volume 1, Issue 4, pp. 405 – 413, Oct 2003

[62] P. Bofill, M. Zibulevski "Blind separation of more sources than mixtures using sparsity of their short-term Fourier transform" In Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation. 19-22 June, Helsinki, 2000

[63] G Tzanetakis, G Essl, P Cook "Automatic Musical Genre Classification of Audio Signals" Proc. Int. Symposium on Music Information Retrieval, ISMIR2001, 2001

[64] LAME Project: http://sourceforge.lame.net

[65] BLADEENC Project: http://bladeenc.mp3.no

[66] Tsunami Codec & filter pack: http://www.lewebdejamy.com/l/Multimedia/Codecs/Tsunami-Codec-et-Filter-Pack-v-3-9-3-Mini.html

[67] Blaze Media Pro: http://www.blazemp.com/

[68] Fraunhofer MPEG Layer3: http://www.iis.fraunhofer.de/amm/techinf/layer3/

[69] Sound Forge 8.0 Manual: http://www.sonymediasoftware.com/download/step2.asp?DID=565

[70] P. de la Cuadra, A. Master, C. Sapp "Efficient Fitch Detection Techniques for Interactive Music" Proc. Int ICMC 2001, 2001

[71] Definizione della finestra di Hanning: mathworld.wolfram.com/HanningFunction.html

[72] G. Middleton "Pitch Detection Algorithms" work produced by The Connexions Project. http://cnx.rice.edu/content/m11714/latest/

[73] V. Arifi, M. Clausen, F. Kurth, M. Mueller "Automatic Synchronization of Musical Data: A Mathematical Approach" CCARH and The MIT Press Editori, 2004

[74] ISO/IEC International Standard IS 11172-3 "Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbits/s - Part 3: Audio"

[75] ISO/IEC International Standard IS 13818-3 "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio"