
UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

*Corso di Laurea triennale in*

*Scienze e Tecnologie delle Comunicazione Musicale*


# RICERCA AUTOMATICA

# DI SEQUENZE MELODICHE

# ALL'INTERNO DI UN CORPUS DI FILE IEEE 1599


RELATORE

Prof. Luca Andrea Ludovico

CORRELATORE

Dott. Adriano Baratè
Dott. Alberto Pinto

TESI DI LAUREA DI

Mattia Frenna

Matr. 725712



Anno Accademico 2010/2011

# Indice

# Capitolo 1

## Introduzione

Ogni opera musicale porta con sé un bagaglio di informazioni di varia natura.
Ad alto livello d'astrazione un brano musicale è caratterizzato infatti da informazioni molto generali, come ad esempio il titolo, il nome dell'album in cui è contenuto, il nome del compositore; un'analisi più approfondita di quello che però è il vero e proprio contenuto informativo di un pezzo, svela aspetti simbolici, notazionali ed ovviamente relativi all'audio.
L'informazione musicale, per poter essere descritta nel suo complesso, necessita perciò di uno standard che permetta di gestire contemporaneamente, ed in maniera sincrona, tutti i diversi strati che la caratterizzano.
Lo strumento di cui ci si è serviti per ottenere questo particolare tipo di descrizione è l' IEEE 1599, nel corso della trattazione verranno analizzate ampiamente le caratteristiche e le potenzialità di questo linguaggio.
Lo Scopo principale di questo elaborato è quello di descrivere lo sviluppo di un software di tipo MIR (Music Information Retrieval) in grado di eseguire comparazioni fra sequenze melodiche, ponendo l'accento sul linguaggio utilizzato per la descrizione del materiale musicale (IEEE 1599) e tenendo presente ciò che già esiste in letteratura per quanto riguarda le metriche e gli algoritmi di string-matching.

## 1.1 Sistemi di reperimento delle informazioni

Verranno di seguito descritte le tecniche utilizzate nell'informatica per affrontare il problema del reperimento delle informazioni sia riguardanti la musica che di qualsiasi altro tipo. Nella sezione 1.2 verrà descritto l'Information Retrieval, nella sezione 1.3 i suoi modelli, nella sezione 1.4 verrà definito il Music Information Retrieval, mentre nella sezione 1.5 verrà data una panoramica dei problemi da affrontare nella progettazione di MIR "Multifaceted"; infine nelle sezioni 1.6 e 1.7 verranno descritte rispettivamente le tecniche di Music IR ed alcuni esempi di MIR esistenti.

## 1.2 Che cos'è l'Information Retrieval?

L'Information Retrieval (IR) **[1]** è l'insieme delle tecniche utilizzate per il recupero mirato dell'informazione in formato elettronico. Per "informazione" si intendono tutti i documenti, i metadati e i file presenti all'interno di banche dati o nel web. Il termine è stato coniato da Calvin Mooers alla fine degli anni '40 del Novecento, oggi è usato quasi esclusivamente in ambito informatico.
L'IR si occupa in generale della rappresentazione, memorizzazione e organizzazione dell'informazione, con la finalità di agevolare l'utente nel soddisfacimento dei suoi bisogni informativi. Data ad esempio una collezione di documenti e un bisogno informativo dell'utente, lo scopo di un sistema di IR è di trovare informazioni che potrebbero essere utili, o rilevanti, per l'utente stesso. Rispetto alla teoria classica delle basi di dati (DBMS), l'aspetto principale non è la ricerca di dati ma la ricerca di informazioni. Negli anni '90, la diffusione del Web ha determinato un aumento dell'interesse nei confronti dell'IR, questo perché il Web non è altro che un'enorme contenitore di documenti sui quali gli utenti vogliono ricercare le informazioni.

Per comprendere il funzionamento dei sistemi IR è necessario rifarsi a due concetti molto elementari ma allo stesso tempo molto importanti: query ed oggetto.

- Query: Dato che i linguaggi di interrogazione si basano molto spesso su comandi di tipo testuale possiamo definirle come stringhe di parole che vengono immesse nel sistema da parte dell'utente.

- Oggetto: Entità che mantiene al suo interno informazioni di varia natura.

Perciò una classica ricerca di IR ha per input il comando (la query) inserito dall'utente che viene confrontato con gli oggetti presenti nell'indice costruito dal sistema di IR.

## 1.3 Modelli di Information Retrieval

I modelli classici di IR sono sostanzialmente tre:

- Il modello booleano
- Il modello vettoriale
- Il modello probabilistico

### 1.3.1 Modello booleano

E' un modello basato sulla teoria degli insiemi e sull'algebra booleana. Ogni documento è rappresentato da un insieme di termini e le query si configurano come espressioni booleane, cioè come insiemi di termini combinati fra loro attraverso l'utilizzo di operatori logici. Il criterio decisionale è di tipo binario, non vi è alcuna nozione di grado di rilevanza. Con questa tecnica un documento viene considerato rilevante o non rilevante.

### 1.3.2 Modello vettoriale

In questo modello viene assegnato un peso, indicato con un numero reale, ad ogni termine e ad ogni query. I documenti e le query vengono quindi rappresentati come vettori in uno spazio n-dimensionale, dove n è il numero di termini indicizzati. La ricerca viene svolta calcolando il grado di similarità tra il vettore che rappresenta la query e i vettori che rappresentano ogni singolo documento.
La funzione attraverso la quale si calcola il grado di similitudine fra due vettori è detta metrica di similarità. Grazie all'uso di una metrica di similarità tra la query ed ogni documento è possibile ordinare i documenti in base al loro grado di rilevanza, definendo una soglia al di sotto della quale respingere i documenti. In questo modo la probabilità che i documenti siano rilevanti per l'utente è direttamente proporzionale al grado di similarità fra questi e la query.

### 1.3.3 Modello probabilistico

E' un modello che stima a livello probabilistico la rilevanza di un dato documento a fronte di una specifica query. In funzione della query vengono definite le caratteristiche della risposta ideale, in maniera tale che per ogni singola query vi sia sempre un insieme di documenti che costituisce tale risposta.

## 1.4 Music Information Retrieval

Music Information Retrieval (MIR) **[2]** è la scienza che si occupa del recupero di informazioni musicali. Essa comprende la ricerca delle similarità tra brani musicali avvalendosi della tecnica di pattern matching, l'identificazione automatica e il riconoscimento musicale, la classificazione, il clustering e la modellazione della musica.
Un campo di ricerca che negli ultimi anni si sta sviluppando velocemente, cercando di superare i metodi utilizzati in precedenza basati sulla semplice indicizzazione e ricerca dei metadati testuali. Le difficoltà maggiori riscontrate in questo campo sono imputabili alla complessità della rappresentazione dell'informazione musicale, all'acquisizione dei diritti d'autore e all'assenza di standard di riferimento tra i vari gruppi di lavoro.

L'attenzione della comunità scientifica è orientata sostanzialmente su due filoni:

- Retrieval di file audio: indicizzazione e ricerca con metadati.
- Retrieval di opere musicali generiche: indicizzazione e ricerca mediante le caratteristiche musicali dell'opera (multifaceted challenge).

## 1.5 Multifaceted challenge

La rappresentazione di un'opera musicale è caratterizzata da diversi aspetti intrinsecamente legati alla natura dell'opera stessa. Questi aspetti sono:

| | |
|---|---|
| **Pitch** | Intonazione/altezza della nota data dalla sua frequenza in termini di oscillazioni al secondo. |
| **Temporal** | Durata degli eventi musicali |
| **Harmonic** | Rappresentazione di polifonie |
| **Timbral** | Rappresentazione del timbri sonoro |
| **Editorial** | Rappresentazione grafica degli spartiti |
| **Textual** | Rappresentazione di un'opera musicale mediante il testo cantato |
| **Bibliographic** | Aspetto relativo ai metadati testuali |

Oggigiorno i più diffusi sistemi MIR trattano il solo aspetto bibliografico, quindi si basano solo ed esclusivamente sui metadati testuali, l'odierna sfida (challenge) del MIR è lo sviluppo di sistemi in grado di funzionare sulla base degli aspetti precedentemente elencati. Ovviamente un obiettivo di questo tipo porta con se tutta una serie di problematiche, legate alla rappresentazione dell'informazione musicale e all'organizzazione dei materiali, che possono essere catalogate come segue:

I. *Multirepresentational Challenge*

II. *Multicultural Challenge*

III. *Multiexperential Challenge*

IV. *Multidisciplinarity Challenge*

***I.*** L'utilizzo di vari sistemi per la rappresentazione informatica dell'opera musicale ha impatto significativo sulla facilità di estrazione delle rappresentazioni interne e sullo spazio richiesto per la loro memorizzazione. Ad esempio la memorizzazione di uno spartito richiede bassi costi di immagazzinamento, quella di un file audio è assai più onerosa. La soluzione potrebbe risiedere nell'utilizzo dell'IEEE 1599, che fornisce una rappresentazione omogenea dei contenuti musicali basata su XML.

***II.*** Questa problematica riguarda invece l'aspetto multiculturale che caratterizza la musica. Opere musicali prodotte al di fuori del sistema tonale occidentale (ad esempio opere di musica indiana, tribale africana) potrebbero richiedere lo studio di aspetti specifici per la loro rappresentazione, diversi da quelli elencati in precedenza (pitch, harmonic,...).

***III.*** Questo aspetto si riferisce ad un possibile sviluppo futuro dei sistemi MIR che prevederebbe la catalogazione di opere secondo considerazioni psico-acustiche sull'effetto emotivo prodotto nell'ascoltatore (noia, tristezza, allegria,...). In questo caso la problematica è quella della catalogazione delle opere, della definizione del concetto di bisogno informativo (“dammi un brano allegro”), della misurazione delle sensazioni tramite sensori biometrici.

***IV.*** L'ultima problematica, o sfida (challenge), che ci si trova a dover affrontare nella progettazione di MIR che non si basino solo ed esclusivamente sui metadati testuali, è quella relativa all'aspetto multidisciplinare della faccenda. La comunità degli sviluppatori e dei ricercatori è una comunità eterogenea, ne conseguono difficoltà nella definizione degli obiettivi e delle modalità operative di un MIR, difficoltà nell'adozione di uno standard per l'indicizzazione delle opere (spartiti vs. campionamento del segnale), ecc...

## 1.6 Tecniche di Music Information Retrieval

**[3]** I metodi di Music Information Retrieval possono essere suddivisi nei seguenti modi:

- Analytic/Production systems [AP]
- Locating systems [LS]

La differenza sostanziale fra questi due approcci sta nella completezza di rappresentazione, maggiore è il numero di aspetti o Facets scelti per rappresentare le opere musicali, maggiore è la completezza del sistema. Un sistema completo include tutti gli aspetti di rappresentazione, sia in forma audio che simbolica.

### 1.6.1 Metodo analityc/production system (AP)

Questo approccio presenta un elevato grado di completezza rappresentativa, e si rivolge ad utenti specialisti del settore, quali musicologi, compositori e trascrittori che utilizzano tale sistema per specifici compiti di analisi teorica e produzione di brani musicali.
Generalmente la questione della rappresentazione dell'informazione musicale viene ricondotta a problematiche già note per le quali esistono soluzioni consolidate applicabili al Music Retrieval. L'obiettivo è modellare la rappresentazione musicale ad un linguaggio di programmazione, sviluppando appositi linguaggi funzionali, e ricondurre tutto ad espressioni regolari tramite modelli di codifica ad hoc che comunque presentano un' elevata complessità concettuale ed implementativa.

### 1.6.2 Metodo locating system (LS)

Questo procedimento è utilizzato specialmente nei sistemi per la localizzazione e l'identificazione di brani musicali ed è rivolto ad utenti non esperti. L'obiettivo è quello di risolvere semplici interrogazioni del tipo:

- trovare tutte le opere di uno specifico autore
- dato un interprete trovarne tutte le esecuzioni
- dato un testo trovare l'opera che lo contiene
- dato il titolo di un'opera, trovare tutte le informazioni relative

In questo modo l'utente generico può trovare le informazioni che cerca in maniera facile e veloce.
Per la risoluzione di tali query sono sufficienti i metadati e le informazioni testuali generiche a contorno dell'opera. Nel caso in cui la richiesta sia più complessa, del tipo data una linea melodica trovare l'opera corrispondente, la query viene risolta attraverso la tecnica dell'Incipit Index che fornisce una rappresentazione testuale dell'inviluppo melodico (contour melodico) della parte iniziale del tema principale.
Il sistema degli Incipit Index è però incompleto dal punto di vista della rappresentazione, poiché basa la ricerca solo sul tema principale. Nonostante ciò questa tecnica fornisce comunque buoni risultati nell'applicazione pratica per via della ridotta complessità di formulazione delle query da parte dell'utente.

## *1.7 Alcuni esempi di sistemi MIR*

*RISM*- **[4]**Annovera nel suo database circa duecentomila composizioni per ottomila autori. Le ricerche si basano sull'intonazione (pitch) e sull'informazione temporale ed i contenuti musicali vengono descritti mediante Incipit Index. In questo sistema ogni opera viene rappresentata come parte iniziale dell'opera stessa con codifica Plaine and Easy (codici alfanumerici per intonazione e durata). Il problema è che per eseguire le interrogazioni è necessario conoscere esattamente la melodia ed il linguaggio specifico, inoltre, le query sono sensibili alla trasposizione in diversa tonalità e vi è una sostanziale difficoltà, se non impossibilità interpretativa delle rappresentazioni interne.

*MELDEX*- **[5]**E' un sistema MIR completo che gestisce circa centomila file midi e diecimila documenti. Il reperimento dei midi viene effettuato attraverso tecniche di web spidering e per ogni singola composizione sono previste rappresentazioni multiple (spartito, audio, midi). Per quanto riguarda le query la particolarità sta nella possibilità di effettuare le cosiddette "query by humming".

# Capitolo 2

## IEEE 1599

L'IEEE 1599 come è già stato anticipato in precedenza nel corso del primo capitolo, è un linguaggio per la codifica dell'informazione musicale basato su XML.
Ciò che caratterizza questo linguaggio è la sua struttura multi-layer che consente di gestire aspetti simbolici, notazionali e oggetti audio allo stesso tempo; abbiamo perciò a che fare con uno strumento utilissimo nel campo dell'Information Retrieval, in quanto consente di sfruttare a pieno le informazioni strutturali.
Un MIR basato su IEEE 1599, ad esempio, potrà eseguire query in funzione del materiale tematico, differenziandosi dai comuni metodi di recupero delle informazioni che non prendono in considerazione le strutture interne dei temi melodici e le relazioni metriche tra le note.
Ogni **[6]** particolare strato dell'informazione musicale è legato a specifici formati proprietari (come il MIDI, per l'aspetto performance, il NIFF per l'aspetto notational ecc...), nessuno di questi può essere opportunamente applicato agli altri strati. IEEE 1599 fornisce una via effettiva per la rappresentazione dell'informazione multimediale a diversi livelli d'astrazione e per l'integrazione di tali livelli.
Nel prossimo paragrafo andremo ad analizzare nello specifico la struttura multi-layer che caratterizza questo linguaggio.

## 2.1 Struttura Multi-Layer

In IEEE 1599 l'informazione musicale è rappresentata attraverso una struttura multi-layer tenuta insieme da un particolare "collante" detto ***Spine*** .
L'informazione in questione può infatti essere strutturata e rappresentata attraverso un modello di suddivisione a strati (layer), in tale modello ogni specifico livello di astrazione dell'informazione musicale corrisponde ad un particolare layer (general, structural, music logic, notational, performance e audio).

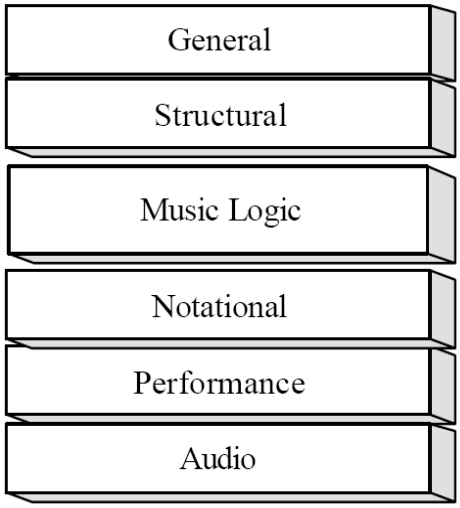



*Fig. 1 – Music information layers*

**[7]** Andiamo ad analizzare i singoli livelli:

- *General*, contiene i metadati relativi al brano in oggetto, tra cui le informazioni catalografiche su titolo dell'opera, autori e genere.

- *Structural,* identifica gli oggetti musicali su cui il brano è costruito e permette di evidenziarne i mutui rapporti.

- *Logic*, vero nucleo del formato, destinato alla descrizione simbolica dei contenuti musicali.

• *Notational*, contiene le differenti rappresentazioni grafiche della partitura, ad esempio riferibili a diverse edizioni o trascrizioni.

• *Performance*, dedicato ai formati per la generazione di esecuzioni sintetiche da parte dell'elaboratore.

•*Audio*, consente di legare al brano in oggetto le esecuzioni audio/video della partitura.

Tali livelli corrispondono a sei sotto-elementi dell'elemento radice( Fig. 2).
IEEE 1599 è in grado di supportare una molteplicità di materiali dello stesso tipo, ad esempio, nel livello audio è possibile inserire più esecuzioni dello stesso brano, come nel livello notational è possibile codificare l'informazione relativa a differenti edizioni a stampa e manoscritte.
Un aspetto particolarmente rilevante per l'approccio appena descritto è il pieno supporto fornito ai formati di codifica già esistenti e comunemente in uso. In questa maniera solo i metadati e le informazioni simboliche sulla partitura vengono esplicitamente descritti in XML, mentre per audio, grafica e video vengono codificati i riferimenti a file esterni.
Questo approccio permette di sfruttare il corpus di oggetti digitali già disponibile e le peculiarità proprie di ciascun formato nella descrizione degli oggetti multimediali.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://www.mx.dico.
unimi.it/ieee1599.dtd">
<ieee1599>
   <general>...</general>
   <logic>...</logic>
   <structural>...</structural>
   <notational>...</notational>
   <performance>...</performance>
   <audio>...</audio>
</ieee1599>
```

*Fig. 2 – Struttura di un documento IEEE 1599*

## 2.2 Lo Spine

Lo Spine è una particolare struttura dati che si trova all'interno del livello Logic e che funge da collante per la struttura multi-layer precedentemente descritta.
All'interno dello spine si trovano elencati tutti gli eventi di interesse per chi effettua la codifica, in maniera tale che questi vengano ordinati mutuamente fra loro ed etichettati in modo univoco (Fig 3b). Le descrizioni e i riferimenti agli eventi contenuti negli altri livelli si basano sempre sull'identificativo univoco dello spine.
Grazie al meccanismo che demanda a file esterni la descrizione multimediale di eventi e che si basa su riferimenti allo spine, aggiungere una traccia audio o le scansioni di un'ulteriore partitura, non comporta la creazione di riferimenti con tutti gli oggetti già mappati, ma solo la determinazione dei punti o degli istanti in cui occorrono gli eventi musicali listati nello spine, all'interno del nuovo file.

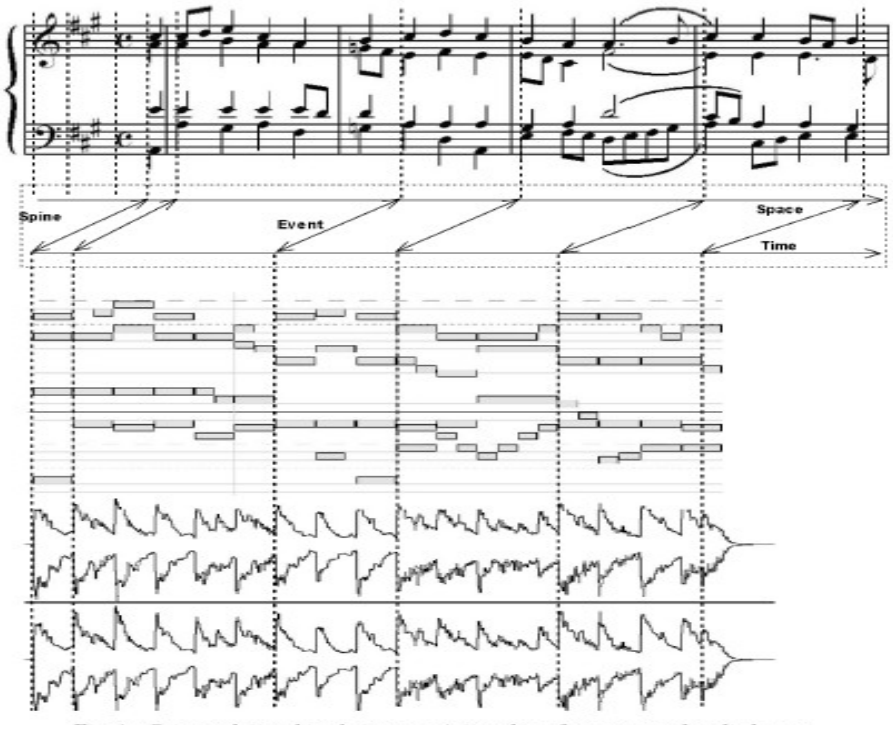



*Fig. 3(a) – Spine: relazioni fra i layer notational, performance e audio.*

```
<ieee1599>
  <logic>
    <spine>
      <event id="e1" timing="0" hpos="0"/>
      <event id="e2" timing="2" hpos="2"/>
      <event id="e3" timing="2" hpos="2"/>
      <event id="e4" timing="1" hpos="1"/>
      <event id="e5" timing="1" hpos="1"/>
      ...
    </spine>
    <los>...</los>
  </logic>
</ieee1599>
```

*Fig. 3(b) – Spine: Lista degli eventi*

## 2.3 Logically Organized Simbols

All'interno del Layer Logic **[8]** troviamo un altro importante strato chiamato *Los* (Logically Organized Symbols) che ha la funzione di descrivere la partitura da un punto di vista simbolico.
Attraverso una struttura gerarchica particolarmente adatta per la rappresentazione dei contenuti musicali (anch'essi fortemente strutturati in modo gerarchico), il *los* fornisce una descrizione astratta dei simboli in partitura.
Di seguito un esempio di come viene descritto un accordo all'interno del sub-layer *Los:*

```
<chord event _ ref="p7v1 _ 69">
    <notehead>
        <pitch step="E" octave="5"/>
        <duration num="1" den="1"/>
    </notehead>
    <notehead>
        <pitch step="G" octave="5"/>
        <duration num="1" den="1"/>
    </notehead>
    <notehead>
        <pitch step="C" octave="6"/>
        <duration num="1" den="1"/>
    </notehead>
</chord>
```

All'interno dell'elemento “chord” (che viene utilizzato per indicare una o più note simultanee con la stessa durata) in questo caso troviamo tre sotto-elementi chiamati “notehead”. Il contenuto di ognuno di questi sotto-elementi sta a rappresentare una singola nota dell'accordo; in particolare l'elemento “pitch”, contenuto in notehead descrive il nome e l'ottava della nota in questione, mentre, l'elemento “duration”, fornisce l'informazione relativa alla durata della nota.
Di seguito un ulteriore esempio che riporta la descrizione di un'intera battuta all'interno del *Los*:

```
<measure number="1">
   <voice ref="Clarinetto _ I _ voice _ 1">
      <chord event _ ref="p5v1 _ 0">
         <notehead>
            <pitch step="B" octave="5" />
            <duration num="3" den="8" />
            <aug _ dot />
         </notehead>
      </chord>
      <chord event _ ref="p5v1 _ 1">
         <notehead>
            <pitch step="B" octave="5" />
            <duration num="1" den="8" />
         </notehead>
      </chord>
      <chord event _ ref="p5v1 _ 2">
         <notehead>
            <pitch step="C" octave="6" alter="1" />
            <duration num="1" den="8" />
         </notehead>
      </chord>
      <chord event _ ref="p5v1 _ 3">
         <notehead>
            <pitch step="D" octave="6" />
            <duration num="1" den="8" />
         </notehead>
      </chord>
   </voice>
</measure>
```

L'elemento “measure” contiene “voice” che attraverso il suo attributo “ref” identifica una voce specifica. A sua volta voice contiene gli elementi chord di cui abbiamo discusso precedentemente.
In questa battuta sono presenti quattro elementi chord, ognuno di essi contiene un unico elemento notehead, questo ci dice che la misura racchiude una sequenza melodica di quattro note singole (NB: più elementi notehead nello stesso chord significa più note simultanee con la stessa durata).

# Capitolo 3

## *Problematiche connesse al pattern matching*

L'analisi musicale computer-assisted **[9]** e le problematiche connesse all'argomento, costituiscono ad oggi un caso di studio di notevole interesse per la comunità scientifica.
Sviluppatasi intorno agli anni '60, la computer-assisted music analysis dispone di strumenti analitici che permettono di risolvere problemi talvolta irrisolvibili senza il supporto del computer, come ad esempio: la creazione di caratterizzazioni complete, statisticamente verificabili di stile, oppure, la verifica della probabilità di paternità di un'opera attraverso il confronto fra il profilo statistico dell'opera ed il profilo statistico dello stile dei compositori.
L'analisi musicale realizzata con il supporto di mezzi informatici contribuisce inoltre allo sviluppo di approcci sistematici nella teoria musicale, supporta la ricerca nel campo dell'acustica e quella sugli aspetti cognitivi e formali della musica.
Le problematiche che nello specifico andremo ad affrontare nel corso di questo capitolo sono quelle relative allo string-pattern-matching.

## *3.1 Panoramica generale sui problemi di pattern matching*

Per parlare **[10]** dei problemi connessi al pattern matching sarà utile innanzitutto introdurre il "gergo" relativo all'argomento. Cominciamo quindi col dare una definizione di *Stringa.*
Una stringa è una sequenza di simboli tratta da un insieme finito di simboli detto *Alfabeto*. I concetti di *Patterns* (modelli) e *Texts* (testi) identificano in generale delle stringhe. Il *Testo* di solito corrisponde ad una partitura o ad un'altra entità musicale, mentre il *Pattern* può essere un motivo fornito dall'utente in forma di sequenza di note.
Alcune problematiche di pattern matching possono comunque non essere legate ad un input fornito dall'utente (pattern specificato dall'utente), ad esempio l'analisi di una partitura, finalizzata a trovare passaggi che si ripetono, non si basa su di un pattern specificato dall'utente.
Una problematica che sta a monte della questione del pattern matching, è quella della rappresentazione delle entità musicali di interesse in un contesto informatico.
Non è difficile capire che una maggiore accuratezza nella rappresentazione del materiale musicale vuol dire maggiore possibilità di ottenere risultati soddisfacenti in fase di matching. Ad esempio per quanto riguarda la rappresentazione del pitch, vi sono quattro livelli di accuratezza che vengono espressi attraverso l'utilizzo di quattro diversi sistemi di rappresentazione: base-7, base-12, base-21 e base-40.
In breve usando il sistema di rappresentazione base-7, l'ottava viene suddivisa in sette parti, abbiamo perciò sette valori per rappresentare le note che stanno in un'ottava
(Fig. 4).

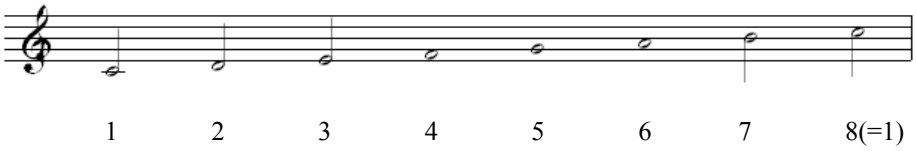
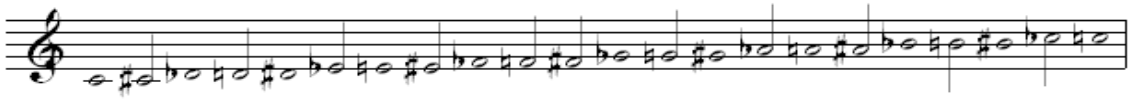



*Fig. 4 – Rappresentazione del pitch in base-7*

Questo tipo di sistema non permette di distinguere intervalli maggiori da intervalli minori, è perciò molto limitante.
Utilizzando il sistema di rappresentazione del pitch base-12 abbiamo invece a disposizione dodici valori per rappresentare le note all'interno dell'ottava, questo comporta un aumento dell'accuratezza.

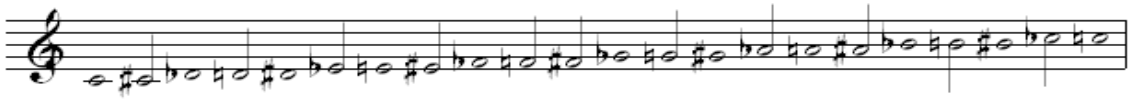


1 2 2 3 4 4 5 6 5 6 7 7 8 9 9 10 11 11 12 13 12 13(=1)

*Fig. 5 – Rappresentazione del pitch in base-12*

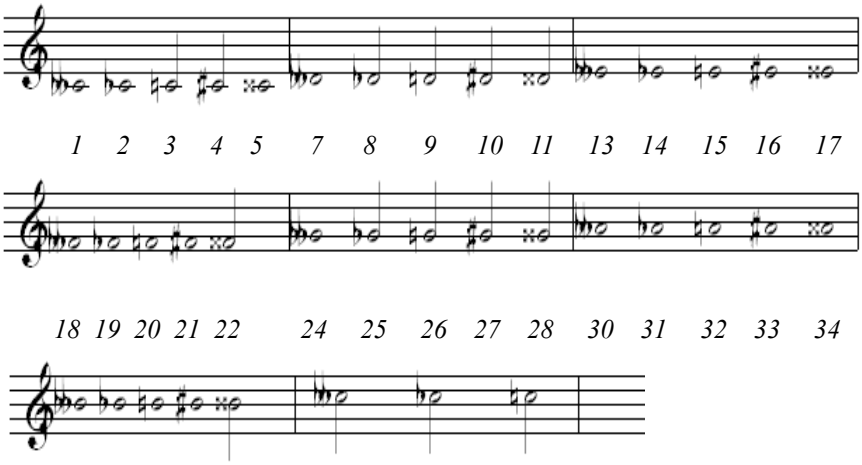
Anche questo sistema presenta delle limitazioni per particolari tipi di applicazioni melodiche, per questo motivo sono stati sviluppati sistemi maggiormente articolati come base-21 e base-40 che utilizzano rispettivamente 21 e 40 valori per rappresentare le note all'interno dell'ottava (*Fig. 6 a, 6 b )*

*(a)*



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22(=1)

*(b)*



*1 2 3 4 5 7 8 9 10 11 13 14 15 16 17*

*18 19 20 21 22 24 25 26 27 28 30 31 32 33 34*

*36 37 38 39 40 (4)1 (4)2 (4)3*

*Fig. 6 – Rappresentazione del pitch in base-21(a) e in base-40 (b)*

Un aspetto particolarmente rilevante della ricerca in questo campo è quello relativo allo sviluppo di sistemi per la rappresentazione multidimensionale dell'informazione musicale (sistemi che tengono conto di tutti gli aspetti qualitativi della musica, come il pitch, la durata, il timbro ecc...).
In ultima analisi, per quel che riguarda la trattazione di partiture polifoniche, è possibile rifarsi a quanto viene suggerito dalla letteratura **[11]**. Una partitura polifonica dovrebbe generalmente essere trattata, o come una collezione di stringhe melodiche, ognuna delle quali etichettata come esplicitamente appartenente ad una certa voce, o come una sequenza di collezioni di note (tralasciando le informazioni relative alle voci) che si presentano contemporaneamente all'interno di un certo intervallo di tempo.
Le informazioni musicali derivate da una partitura che utilizza la notazione musicale convenzionale vengono trattate di solito nella prima maniera, mentre, quelle derivate ad esempio dal MIDI (a cui mancano le informazioni esplicite sulle voci) vengono trattate nella seconda.
Passiamo ora ad un'analisi più specifica delle varie categorie di pattern-match e dei relativi algoritmi.

## *3.2 Categorie di Pattern-Match e algoritmi*

Le questioni che si andranno a considerare nel corso di questo paragrafo sono quelle basate sul modello di rappresentazione bidimensionale (pitch e durata) e riguardano quelle entità musicali polifoniche che detengono informazioni relative alle voci.
Cominciamo col distinguere due tipologie di match: *esatti* e *trasposti*.
Nel primo caso ciò che viene "matchato", confrontato, è la specifica informazione di pitch, mentre nel secondo, viene confrontata l'informazione relativa agli intervalli. Noi ci occuperemo principalmente degli algoritmi relativi al match esatto.

**Exact Matching** – Quando si parla di matching esatto ci si riferisce al problema per cui, data una sequenza di note è necessario verificare se tale stringa ricorre all'interno di una delle voci. Dalla letteratura apprendiamo che tale problema può essere risolto attraverso l'utilizzo dell'algoritmo di Knuth-Morris-Pratt (1977) che permette appunto di trovare le occorrenze di una stringa *P* (pattern) in un testo *S*. La particolarità dell'algoritmo in questione **[12]** risiede nel pretrattamento della stringa da cercare, la quale contiene l'informazione sufficiente a determinare la posizione da cui continuare la ricerca in caso di non-corrispondenza. Questo permette all'algoritmo di non verificare i caratteri che erano stati precedentemente verificati e dunque di limitare il numero di confronti necessari.
Nella pratica il metodo migliore è l'algoritmo di Boyer-Moore (1977) che è

un'estensione del metodo di Knuth-Morris-Pratt. Questo algoritmo ha un funzionamento simile a quello di forza bruta[1] e basa il suo funzionamento sull'utilizzo di due tecniche euristiche: Bad Character e Good Suffix[2]. Se i caratteri del pattern si confrontano con quelli del testo, è stata trovata l'occorrenza, altrimenti viene spostato il pattern di *K* posizioni, dove *K* è il massimo fra i valori proposti dalle due euristiche.
Vale la pena di citare un ulteriore metodo chiamato automa di Aho-Corasick implementato nel comando *grep* nei sistemi **UNIX.** L'automa[3] in questione è una versione estesa della struttura dati suffix-tree[4] ed è particolarmente adatto per quei casi dove, dati più pattern, si vuole verificare se ognuno di essi occorre nel testo.
Per quanto riguarda il tempo di esecuzione degli algoritmi citati fin qui, possiamo dire che nell'algoritmo di Knuth-Morris-Pratt esso non dipende dalla dimensione dell'alfabeto, mentre,questo non accade per Boyer-Moore e Aho-Corasick.
Fin qui è stata trattata la casistica del matching esatto, quello di maggiore interesse per quanto concerne lo sviluppo del software MIR di cui tratta questo elaborato.
Ora, per ragioni di completezza, andremo ad introdurre altre problematiche che vale la pena discutere nella trattazione dell'argomento.

**Matching with Deletions** – E' il problema per cui è necessario verificare se una sequenza occorre in una delle voci, tralasciando le informazioni di durata. Questo problema, essendo abbastanza semplice, potrebbe essere risolto o con algoritmi di string-matching approssimato, o con algoritmi più snelli e veloci, sviluppati appositamente per la risoluzione del caso in questione.

**Repetition identification** – Identificazione delle ripetizioni; Problematica per cui dato un insieme di sequenze di note (voci), è necessario identificare ripetizioni non sovrapposte di un pattern all'interno di queste sequenze.
Per risolvere questo tipo di problematica vi sono diversi metodi, Main and Lorentz ad esempio è un metodo basato sulla failure function, il cui tempo di esecuzione è espresso dalla funzione *O(n log n),* dove *n* indica la lunghezza del testo.

---

1 **Algoritmo di forza bruta:** un algoritmo molto semplice per cui facendo scivolare il pattern *P* sopra il testo *T* , se gli elementi di *P* si confrontano perfettamente con quelli di *T,* l'occorrenza è stata trovata, altrimenti *P* viene spostato a destra di una posizione

2 **Euristica Bad Character**: Quando nella ricerca un carattere del testo crea un mismatch con il carattere relativo del pattern, l'euristica dichiara quel carattere come *bad-character* e fa scivolare il pattern a destra in modo da far coincidere il bad-character con la sua occorrenza più a destra nel pattern.

**Euristica Good Suffix:** Simile all'euristica precedente, ma quando occorre un mismatch, dichiara come *good-suffix* tutti i caratteri che si sono allineati correttamente a P e fa scivolare il pattern di uno spostamento minimo da far coincidere il good-suffix con gli eventuali caratteri di P.

3 **Automa a stati finiti:** Un automa a stati finiti (ASF) **[13]** è un sistema dinamico, invariante e discreto nell'avanzamento e nelle interazioni, nel quale gli insiemi dei possibili valori di ingresso, uscita e stato, sono insiemi finiti. Possiamo vederlo in sostanza come un piccolo dispositivo che mediante una testina legge una stringa di input su un nastro e la elabora, facendo uso di un meccanismo molto semplice di calcolo e di una memoria limitata. L'esame della stringa avviene un carattere alla volta attraverso precisi passi computazionali che comportano l'avanzamento della testina stessa

4 **Suffix-tree:** Un albero dei suffissi (*suffix tree* in inglese) **[14]** è una struttura dati che evidenzia la struttura interna di una stringa in un modo che facilita operazioni comuni come la ricerca di sottostringhe. Gli alberi dei suffissi permettono di risolvere il problema del matching esatto in tempo lineare (al pari di altri algoritmi come Knuth-Morris-Pratt, Boyer-Moore...) ma la loro principale virtù è che essi possono essere usati per risolvere in tempo lineare molti altri problemi più complessi del pattern matching esatto.

**Overlapping Repetition Identification** – Identificazione di ripetizioni sovrapposte; Dato un insieme di voci, identificare pattern ripetuti che potrebbero sovrapporsi in voci differenti o nella stessa.
Per questo tipo di problematica il metodo consigliato è Apostolico and Ehenfeucht, un metodo basato sulla struttura dati Suffix-tree che identifica tutte le posizioni nel testo dove iniziano le ripetizioni e come queste sono mappate nella stessa locazione della struttura dati Suffix-tree.

**Transformed Matching** – Problema per cui, dato un insieme di voci e un pattern, si devono verificare le occorrenze del pattern all'interno delle voci, o in forma originale (problema 1), o in forma inversa, o retrogradato, o in forma di retrogradazione dell'inversione.
In questo caso la soluzione del problema può risiedere nell'applicare tre volte consecutive l'algoritmo di pattern-matching esatto usato per risolvere "problema 1".
L'automa di Aho-Corasick è il metodo migliore per trattare questo tipo di problema, in quanto è in grado di gestire tutti i quattro casi in un passo solo.

**Distribuited Matching** – Il problema in questo caso è quello per cui dato un insieme di voci e un pattern, è necessario verificare se il pattern ricorre all'interno di una sola voce o se si trova distribuito orizzontalmente a cavallo fra voci diverse.
La letteratura non indica alcun metodo specifico per la risoluzione di questo tipo di problema, è possibile comunque utilizzare un sistema basato sull'automa di Aho-Corasick.

**Approximate Matching** – Problematica per la quale dato un insieme di voci e un pattern, è necessario verificare le occorrenze approssimate del pattern in una delle voci.
Algoritmi per la risoluzione di questo problema si trovano in Crochemore and Rytter (1996), Aho (1990) e Ukkonen (1985). In aggiunta alle considerazioni sui problemi legati allo string-matching approssimato, in questo caso si considerano anche tali problemi in presenza di errori.

# Capitolo 4

## *Descrizione del software*

Nel corso dei capitoli precedenti sono stati introdotti tutti gli aspetti necessari per comprendere lo sviluppo del software di Music IR nella sua interezza. Questo ultimo capitolo sarà dedicato alla descrizione del programma focalizzata sul funzionamento dello stesso. Iniziamo con un'analisi ad alto livello della questione.
L'obiettivo iniziale del progetto era quello di produrre un software, scritto in C# **[15]**, capace di effettuare ricerche di sequenze melodiche all'interno di un corpus di file IEEE 1599. Tale obiettivo è stato sostanzialmente conseguito con successo, ma è stato necessario l'apporto di alcune modifiche concettuali e pratiche alla questione.
In fase di sviluppo si è pensato che l'operazione di pattern matching, in questo caso, richiedesse un'analisi preventiva dei file IEEE 1599 costituenti il corpus, la conseguenza naturale di questa analisi è stata la necessità di memorizzare le informazioni ritenute utili. Per questo motivo si è pensato di sviluppare il software suddividendolo in due parti totalmente indipendenti nel funzionamento: la prima parte, il primo software (se vogliamo), orientata all'analisi dei documenti IEEE 1599, all'estrazione di quelle informazioni utili nell'operazione di pattern matching e alla memorizzazione di tali informazioni; la seconda parte, volta invece a ricevere l'input dall'utente e ad eseguire la ricerca per confronto.
Vista la suddivisione fisica e concettuale del problema, svolgeremo nello specifico un'analisi delle singole parti fondamentali di cui si compone il programma, partendo ovviamente dalla prima.

## 4.1 Lettura dei File IEEE 1599

La lettura (analisi) dei file IEEE 1599 è la prima operazione che viene eseguita dal software in questione, ed in particolare da quella parte del programma (la prima parte) che ha la funzione di: analizzare i file, estrarre le informazioni utili per le operazioni di pattern matching e memorizzare tali informazioni.
I documenti IEEE 1599 vengono caricati dall'utente per mezzo di una finestra di dialogo opportunamente predisposta, una volta selezionati i file, il software comincerà ad elaborare singolarmente tutti i documenti caricati.
Quello che viene fatto consiste sostanzialmente nell'accedere alle parti di interesse dei documenti attraverso una serie di cicli annidati. Questo tipo di meccanismo permette di organizzare tutti gli eventi, suddividendoli per parti e per singole voci. Di seguito un esempio di codice che permette l'accesso a tutti gli eventi musicali di una specifica parte.

```
foreach (XElement node in
Document.XPathSelectElements("logic/los//part"))
{
   XAttribute a = node.Attribute("id");
   StreamWriter sw = new StreamWriter(a.Value + "'" + nome_brano +
     "'.mir", true, Encoding.ASCII);

foreach (XElement nodo in
Document.XPathSelectElements("logic/los/part[@id='" + a.Value +
"']//voice"))
```

Il primo ciclo permette di estrarre tutti i nodi chiamati **"part"** dal documento corrente, per ogni nodo part (che contiene tutti gli eventi musicali relativi ad una specifica parte) viene memorizzato l'attributo "**id"** (che lo identifica) all'interno di una variabile in grado di ricevere tale "valore" (in realtà l'accesso al valore vero e proprio dell'attributo lo si ottiene attraverso la proprietà **Value**), successivamente il valore viene utilizzato all'interno del secondo ciclo per accedere a tutte le voci di quella specifica parte identificata dall'id.
E' da notare la riga di codice che fin qui è stata tralasciata:

```
StreamWriter sw = new StreamWriter(a.Value + "'" + nome_brano +

"'.mir", true, Encoding.ASCII);
```

Questa riga inizializza un nuovo oggetto della classe **StreamWriter,** questo significa che per ogni nodo **part** e quindi in linguaggio naturale, per ogni parte, verrà creato un documento di testo che prenderà il nome dal valore dell'attributo **id** e dal nome del brano che si sta analizzando; il documento avrà estensione "**.mir"**.
Ad esempio: part_1 'Inventio 1'.mir

### *4.1.1 L'estensione .mir*

L'estensione **.mir** (pensata appositamente per i nostri scopi) sta ad indicare un particolare formato di file per la rappresentazione del contenuto melodico di un brano. E' un formato molto semplice, dove ogni riga rappresenta una nota o una pausa e relativo numero di battuta all'interno del brano.
All'interno di un file con estensione **.mir** si trova tutto il contenuto melodico relativo ad una specifica parte. Ad esempio, in un brano per canto e piano ci saranno due parti, la rappresentazione del contenuto melodico dell'intero brano si troverà all'interno di due file **.mir** che conterranno rispettivamente, la parte del canto e quella del piano.
Un altro aspetto importante è quello relativo alla suddivisione delle voci. All'interno di un file di questo tipo le voci sono suddivise da un trattino"-", nel caso in cui in una parte vi sia un' unica voce, il trattino si troverà solo alla fine del documento.
Di seguito viene riportato un esempio:

```
1/000000000000
1/100000000000
2/001000000000
```

Leggendo la prima riga da sinistra verso destra, troviamo il numero "1" seguito da uno "/", questo ci dice che la nota rappresentata si trova nella prima battuta. Perciò l'informazione di battuta si trova a sinistra delle righe, separata, attraverso il simbolo "/", dalla parte della riga che indentifica la nota specifica.
Come abbiamo già detto le voci sono separate fra loro da un trattino "-". Di seguito un esempio:

```
38/001000010000
38/001000010000
-
29/001000010000
29/001000010000
```

Il formato non prevede comunque un sistema di identificazione delle voci.
Passiamo ora alla descrizione del meccanismo di rappresentazione delle note.
Prendiamo in considerazione la riga seguente:

```
1/000000000000
```

I caratteri per la rappresentazione delle note possono essere zeri o uno. Leggendo da sinistra verso destra (dopo lo "/") ogni carattere corrisponde ad una nota, partendo da DO. Ad esempio, il primo carattere dopo lo "/", se fosse "1", starebbe ad indicare la presenza di un DO e così via fino ad arrivare a SI (Do, Do#, Re, Re#, Mi. Fa, Fa#, Sol, Sol#, La, La#, Si). Siccome tutti i caratteri all'interno della stringa sono uguali a zero, questa linea sta ad indicare una pausa.

Nella riga seguente abbiamo invece due note che formano un intervallo armonico:

```
38/001000010000
```

Questa linea ci dice che a battuta "38" si presenta un evento musicale corrispondente ad un accordo, formato da un RE e da un SOL . Perciò se all'interno di una linea vi sono più caratteri uguali ad "1", significa che quella riga rappresenta un accordo e non una singola nota.

I file .mir vengono utilizzati per la memorizzazione delle informazioni utili alle operazioni di string-matching.

## *4.2 Scrittura dei file con estensione .mir*

Vista la struttura di un file **.mir** possiamo passare alla descrizione di come vengono creati questi file.
Abbiamo già detto che il codice che definisce la prima parte dell'applicazione è costituito sostanzialmente da una serie di cicli annidati**,** attraverso tali cicli è possibile scendere "in profondità" nella lettura dei file IEEE 1599. Le informazioni di interesse estratte dai file vengono a questo punto memorizzate in strutture adatte a contenerle. Ad esempio una nota viene rappresentata da un oggetto di questo tipo**:**

```
struct note

  {
     public int? Octave;
     public char? Step;
     public int num;
     public int den;
     public accidental Accidental;
  }
```

Il campo "Octave" contine l'informazione relativa all'ottava; il campo "Step" contiene il nome della nota in inglese; i campi "num" e "den" esprimono la durata dell'evento come frazione, dove "num" è il numeratore e "den" il denominatore; il campo "Accidental" contiene l'informazione relativa alle eventuali alterazioni della nota.

Questi oggetti sono contenuti in liste che a loro volta sono contenute in una lista che a sua volta è contenuta in un'altra lista. Tutto questo serve ad ordinare gli eventi in voci e le voci in liste di voci.
Ogni nota viene rappresentata in due modi, attraverso una struttura e attraverso un array. Per ogni evento musicale (anche se formato da più note simultanee), viene creato un array di bool, di dimensione 12. Partendo dalla posizione zero dell'array che corrisponde a DO, ogni posizione successiva corrisponde alla nota successiva (esempio: la posizione 1 corrisponde a DO# e così via). Se la posizione "0" dell'array, ad esempio, viene inizializzata a false, significa che nell'evento a cui si riferisce l'array non c'è un DO e viceversa. Di seguito un esempio di inizializzazione di un array:

```
for (int i = 0; i < 12; i++)

  {
     Array[i] = false;
  }

    listaArray.Add(Array);
```

In questo caso l'array chiamato "Array" viene inizializzato tutto con il valore false, sta perciò a rappresentare una pausa.
Al termine dell'inizializzazione, l'array viene caricato in una lista chimata "listaArray" e nel documento chiamato "sw", che è un documento .mir creato precedentemente con questo codice:

```
StreamWriter sw = new StreamWriter(a.Value + "'" + nome_brano +

"'.mir", true, Encoding.ASCII);
```

viene scritta (memorizzata) la riga corrispondente a questo evento (in questo caso sarà una sequenza di dodici zeri preceduta dal numero di battuta corrispondente).

```
1/000000000000
```

Il codice che permette la scrittura è il seguente:

```
sw.Write(numero_battuta + "/");
for (int i = 0; i < 12; i++)

  {
     sw.Write('0');
  }

  sw.WriteLine();
```

Questo codice è quello utilizzato per scrivere gli eventi “pausa” all'interno del file .mir.
Nel caso in cui l'evento musicale corrisponda ad una nota o ad un accordo, l'array, verrà prima inizializzato tutto a false, ed in seguito, verranno modificati i valori di quelle posizioni che corrispondono alle note dell'evento:

```
switch (nome_nota)
{

      case 'C': Array[(12 + controllo_alterazione) % 12] = true; break;
      case 'D': Array[(14 + controllo_alterazione) % 12] = true; break;
      case 'E': Array[(16 + controllo_alterazione) % 12] = true; break;
      case 'F': Array[(17 + controllo_alterazione) % 12] = true; break;
      case 'G': Array[(19 + controllo_alterazione) % 12] = true; break;
      case 'A': Array[(21 + controllo_alterazione) % 12] = true; break;
      case 'B': Array[(23 + controllo_alterazione) % 12] = true; break;

}
```

Questo codice verrà reiterato tante volte quante sono le note dell'accordo.

La scrittura della riga corrispondente al contenuto dell'array, in questo caso, avverrà attraverso il codice seguente:

```
sw.Write(numero_battuta + "/");
foreach (bool valore in Array)

{
    if (valore == true) sw.Write('1');
    else sw.Write('0');
}
sw.WriteLine();
listaArray.Add(Array);
```

Questo codice oltre ad effettuare la scrittura di una riga del file .mir, esegue la conversione bool to char (quando legge true scrive “1” e quando legge false scrive “0”). Ogni file .mir può essere visto come un supporto per la memorizzazione di informazioni musicali, in quanto, mantiene l'intera informazione melodica relativa ad una specifica parte di uno specifico brano.

## 4.3 Ricerca di sequenze melodiche

Passiamo ora alla seconda parte del software, vero e proprio cuore del progetto.
Questa parte, dedicata all'operazione di pattern matching, ha la funzione di ricevere in input dati dall'utente e di elaborare tali dati. Allo stesso tempo si occupa della lettura dei file .mir precedentemente creati e dell'organizzazione delle informazioni in liste di strutture, per finire poi con l'esecuzione dei confronti.

### 4.3.1 Elaborazione dei dati inseriti dall'utente

Partiamo con la descrizione di quanto accade ai dati inseriti dall'utente.
Nel momento in cui viene selezionata una nota attraverso l'interfaccia grafica, viene invocato il metodo seguente:

```
private void radioA_Checked(object sender, RoutedEventArgs e)
{
     rb = (RadioButton)sender;
     radio_doubleFlat.IsEnabled = true;
     radio_flat.IsEnabled = true;
     radio_doubleSharp.IsEnabled = true;
     radio_natural.IsEnabled = true;
     radio_sharp.IsEnabled = true;
     radio_sharp.IsChecked = false;
     radio_natural.IsChecked = false;
     radio_flat.IsChecked = false;
     radio_doubleSharp.IsChecked = false;
     radio_doubleFlat.IsChecked = false;
     Array = new bool[12];
     for (int i = 0; i < 12; i++)
     {
         Array[i] = false;
     }
}
```

Attraverso questo codice viene tenuta memoria della nota selezionata e vengono abilitati tutti i radiobutton che permettono di selezionare le alterazioni; viene anche creato ed inizializzato un array di bool, chiamato "Array", con valori uguali a false.
Selezionata la nota è necessario, perchè questa venga aggiunta alla sequenza da ricercare, scegliere un'alterazione fra quelle disponibili (doubleFlat, flat, natural, sharp, doubleSharp). Una volta deciso per una di queste, verrà invocato un metodo specifico in funzione di quella selezionata. Ad esempio:

```
private void radio_doubleFlat_Checked(object sender, RoutedEventArgs e)
{
   listBox1.Items.Add(rb.Name + " double flat");
   controllo_alterazione = -2;
   switch (rb.Name)
   {
       case "C": Array[(12 + controllo_alterazione) % 12] = true;break;
       case "D": Array[(14 + controllo_alterazione) % 12] = true;break;
       case "E": Array[(16 + controllo_alterazione) % 12] = true;break;
       case "F": Array[(17 + controllo_alterazione) % 12] = true;break;
       case "G": Array[(19 + controllo_alterazione) % 12] = true;break;
       case "A": Array[(21 + controllo_alterazione) % 12] = true;break;
       case "B": Array[(23 + controllo_alterazione) % 12] = true;break;
   }

   lista_noteUtente.Add(Array);
   radio_doubleFlat.IsEnabled = false;
   radio_flat.IsEnabled = false;
   radio_doubleSharp.IsEnabled = false;
   radio_natural.IsEnabled = false;
   radio_sharp.IsEnabled = false;
   radio_sharp.IsChecked = false;
   radio_natural.IsChecked = false;
   radio_flat.IsChecked = false;
   radio_doubleSharp.IsChecked = false;
   radio_doubleFlat.IsChecked = false;
   A.IsChecked = false;
   B.IsChecked = false;
   C.IsChecked = false;
   D.IsChecked = false;
   E.IsChecked = false;
   F.IsChecked = false;
   G.IsChecked = false;

}
```

Questo è il metodo corrispondente al radio button "radio_doubleFlat". Questa funzione aggiunge ad una listbox il nome della nota precedentemente selezionata, seguito dalla stringa "double flat". Attraverso la variabile "controllo_alterazione" ed il ciclo che segue, viene impostata a true la posizione dell'array (Array) corrispondente alla nota selezionata.
A questo punto l'array viene caricato in una lista destinata a contenere "le note inserite dall'utente" e chiamata lista_noteUtente. Il codice restante serve a disabilitare e a deselezionare nuovamente i radio button per l'inserimento delle alterazioni e per la scelta delle note.

### 4.3.2 La ricerca

Per prima cosa viene attuata una semplificazione delle informazioni contenute in lista_noteUtente. Viene infatti creata una nuova lista di interi, chiamata "l_int", destinata a contenere solo i numeri di posizione degli elementi degli array (contenuti in lista_noteUtente) impostati a true.
Pertanto la lista "l_int" conterrà i nomi delle note inserite dall'utente, sottoforma di numeri (esempio: zero per indicare DO, uno per indicare DO#, ecc...).
Di seguito il codice che crea ed inizializza la lista l_int:

```
List<int> l_int = new List<int>();
  foreach (Array array_nota in lista_noteUtente)
  {
      i = 0;
      foreach (bool posizione in array_nota)
      {
          if (posizione == false)
          {
              i++;
              continue;
          }
          else
          {
              l_int.Add(i);
          }
      }

  }
```

Per ogni file caricato dall'utente, il programma estrae l'informazione melodica relativa ad ogni singola voce e la memorizza in strutture di questo tipo:

```
struct struttura
{
      public string n_battuta;
      public bool[] n_nota;
      public struttura(string string_val, bool[] boolean)
      {
          n_nota = new bool[12];
          n_battuta = string_val;

          for (int d = 0; d < 12; d++)
          {
              n_nota[d] = boolean[d];
          }
      }

}
```

Attraverso questo blocco di codice:

```
List<struttura> Lista_Note = new List<struttura>();
while (line != "-")
{
    bool[] arr = new bool[12] { false, false, false, false,

                                false, false, false, false,

                                false, false, false, false, };
    struttura nota = new struttura("0", arr);
    while (line[cont] != '/')
    {
        cont++;
    }

    nota.n_battuta = line.Substring(0, cont);
    for (int l = cont + 1; l < line.Length; l++)
    {
        if (line[l] == '0')
        {
            nota.n_nota[(l - cont - 1)] = false;
        }
        else if (line[l] == '1')
        {
            nota.n_nota[(l - cont - 1)] = true;
        }
    }
    Lista_Note.Add(nota);
    line = sr.ReadLine();
    cont = 0;

}
```

Queste strutture vengono a loro volta memorizzate in una lista di strutture chiamata Lista_Note.

Attraverso l'algoritmo seguente verrà quindi eseguito il confronto fra la sequenza inserita dall'utente e l'informazione melodica relativa ad ogni singola voce memorizzata in Lista_Note:

```
foreach (struttura strutt in Lista_Note)//questo viene eseguito per
ogni voce...Lista_Note è una voce
            {
                for (int f = 0; f < 12; f++)
                {
                    if (strutt.n_nota[f] == true)
                    {
                        b = f;
                        if (b == l_int[g])
                        {
                            g++;//fa scorrere l'indice della lista l_int
                            matched++;
                            numeri_battute.Add(strutt.n_battuta);
                            if (matched < l_int.Count) break;
                            else
                            {
                                listBox2.Items.Add("Found in " + nome +

                                                   " measure ");

                                foreach (string st in numeri_battute)
                                {
                                    listBox2.Items.Add(st);
                                }
                                verifica = true;
                                f = 0;
                                g = 0;
                                matched = 0;
                                numeri_battute.Clear();
                                break;
                            }


                        }
                        else
                        {
                            f = 0;
                            g = 0;
                            matched = 0;
                            numeri_battute.Clear();
                            break;
                        }
                    }


                }



            }
```

## 4.4 Manuale Utente

Una breve descrizione delle modalità di utilizzo del software.
Per effettuare ricerche di sequenze melodiche è necessario prima di tutto codificare i file IEEE 1599 (sui quali si vuole eseguire la ricerca) in file .mir. Avviando l'applicazione "1" e cliccando **Browse** si aprirà una finestra di dialogo attraverso la quale sarà possibile selezionare i file da analizzare e codificare.

*a)*


*b)*

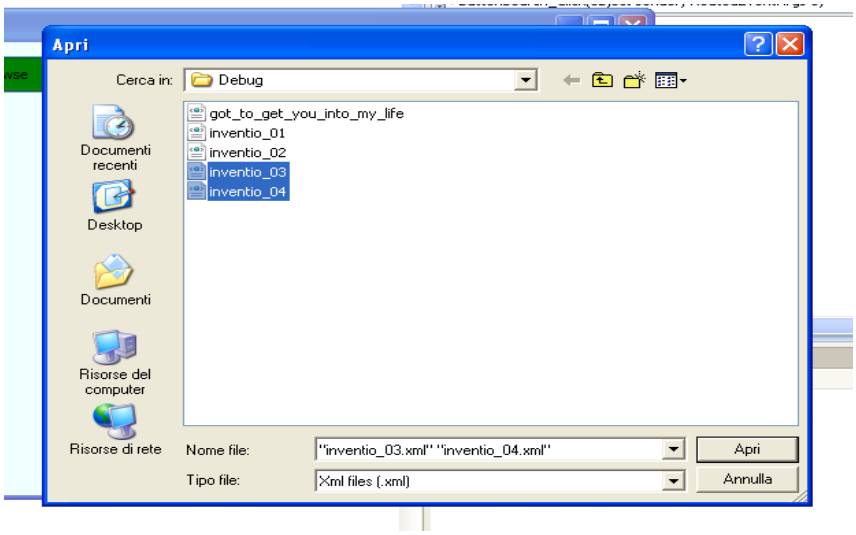



*Fig. 7 – Interfaccia grafica 7(a), finestra di dialogo 7(b)*

Terminata la creazione dei file .mir, comparirà la scritta "finished executing" nella listbox dell'interfaccia grafica; i file prodotti si troveranno quindi nella cartella corrente. Per eseguire ricerche bisognerà avviare quindi l'applicazione "2" che si presenta come in figura:

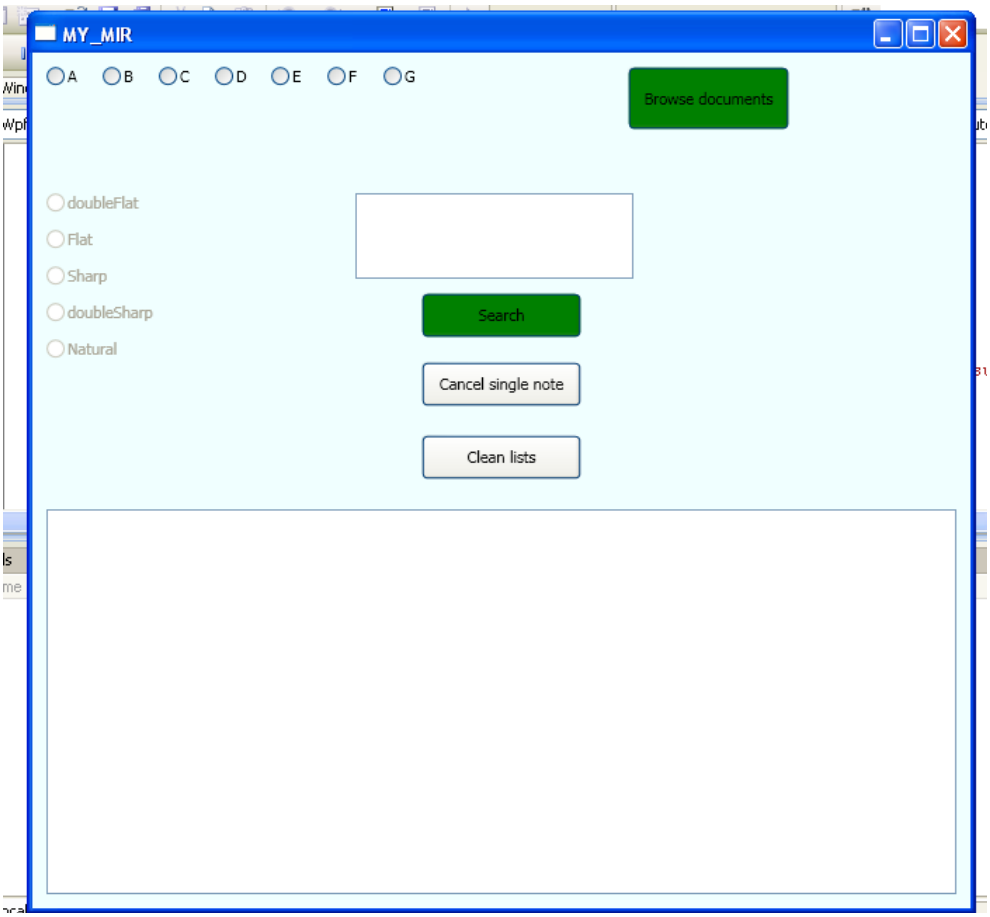



*Fig. 8 – Interfaccia grafica applicazione 2*

La selezione di una nota abilita i radio button per la scelta dell'alterazione e solo dopo aver selezionato una fra le alterazioni disponibili, comparirà la nota col suo nome nella listbox più piccola (Fig 9).

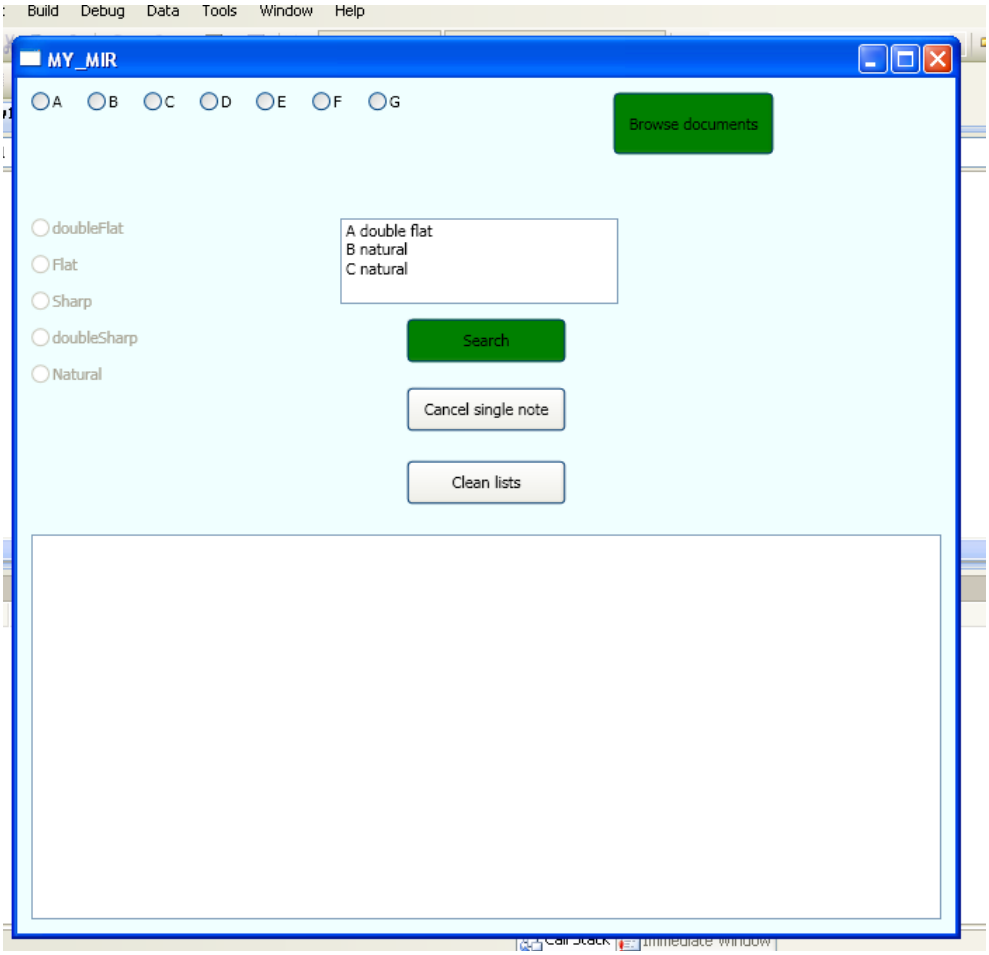



*Fig. 9 – Interfaccia grafica applicazione 2, caricamento sequenza.*

A questo punto, dopo aver caricato i documenti sui quali verrà effettuata la ricerca attraverso il tasto **browse documents**, sarà possibile far partire il match cliccando il tasto **search**.

L'output sarà del tipo mostrato in figura 10:

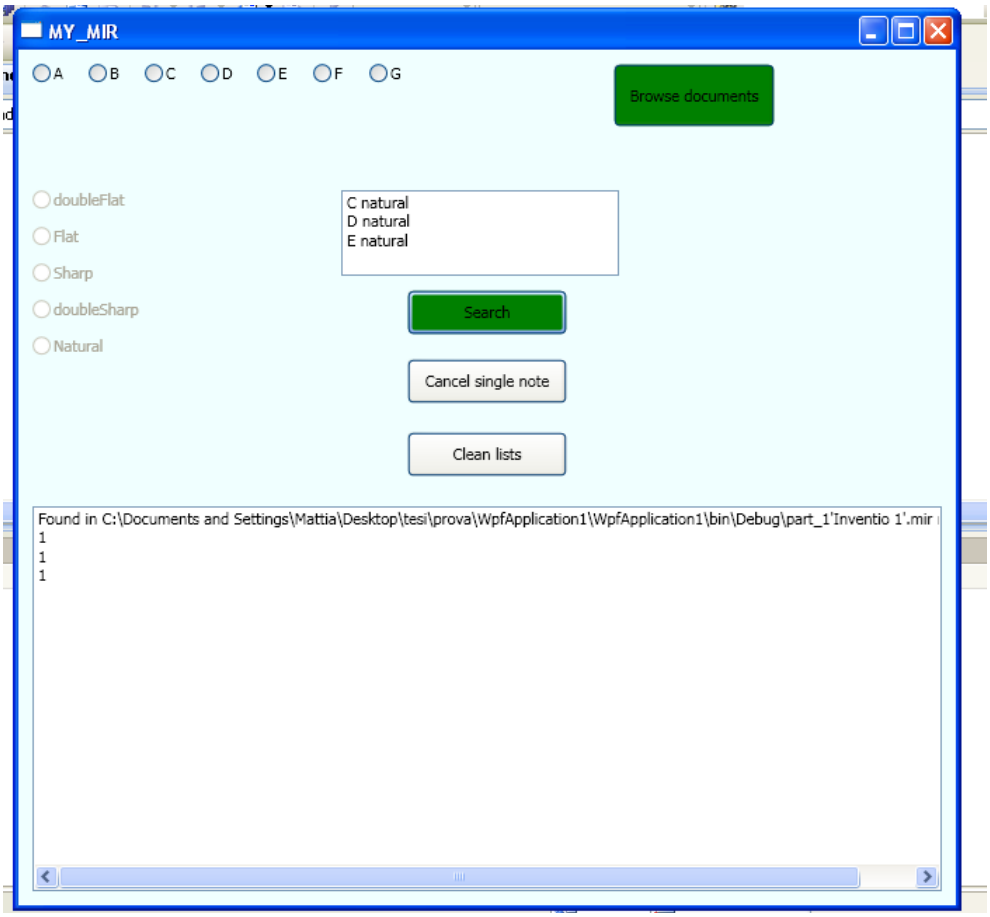



*Fig. 10 – Interfaccia grafica applicazione 2, l'output.*

Nella listbox più grande verrà riportata la stringa:

*"Found in /"percorso del file"/ measure"*

seguita, a capo, dai numeri di battuta di ogni nota della sequenza trovata.

Nel caso in cui si vogliano pulire entrambe le listbox, basterà cliccare il tasto **Clean lists**. Per cancellare una singola nota della sequenza è disponibile invece il tasto **Cancel single note**.

# Conclusioni

L'obiettivo di questo elaborato è stato quello di sviluppare un software in grado di effettuare ricerche di sequenze melodiche all'interno di un corpus di file IEEE 1599.
In fase di progettazione si è pensato di modificare leggermente tale obiettivo, si è optato infatti per lo sviluppo di un formato in grado di memorizzare unicamente l'informazione melodica di un brano e i numeri di battuta degli eventi musicali. Questo particolare formato, identificato dall'estensione .mir, è stato utilizzato quindi come supporto per la memorizzazione delle informazioni utili per i confronti.
Il software sviluppato perciò, non effettua ricerche direttamente su file IEEE 1599, ma su file .mir (che vengono comunque costruiti attraverso l'analisi dei file IEEE 1599).
Il lavoro si è articolato in due fasi: sviluppo del software per l'analisi dei file IEEE 1599, sviluppo del software di pattern matching vero e proprio.
La prima fase è stata quella che ha richiesto più sforzi in termini di tempo e competenze da acquisire. E' stato necessario infatti un lavoro preventivo, volto a capire quali sarebbero potuti essere gli strumenti utili e necessari per lo sviluppo del software in questione e quale sarebbe dovuta essere la logica per l'analisi dei file (IEEE1599).
Nella seconda fase ci si è concentrati principalmnte sullo sviluppo di un buon algoritmo di pattern matching.
Uno sviluppo futuro del progetto potrebbe essere quello di implementare un algoritmo di pattern-matching con approssimazione, in grado di affrontare le problematiche discusse nel terzo capitolo.

# Bibliografia


[1] R. Baeza-Yates, B. Riberiro-Neto
"Modern Information Retrieval"
Addison-Wesley, 1999

[2] URL: http://www.music-ir.org

[3] J. Stephen Downie, University of Illinois at Urbana-Champaign
"Music Information Retrieval"

[4] URL: www.RISM.harvard.edu/RISM/Welcome.html

[5] URL : www.nzdl.org/musiclib

[6] A. Pinto, G. Haus
"A Novel XML Music Information Retrieval Method Using Graph Invariants"
ACM Transactions on Information Systems, Vol. 25, No. 4, 2007

[7] Adriano Baratè, Luca A. Ludovico, Davide A. Mauro
"Tecnologie basate su XML per la fruizione avanzata dei contenuti musicali"

[8] Adriano Baratè, G. Haus, Luca A. Ludovico
"Music Representation of Score, Sound, MIDI,
Structure and Metadata All Integrated in a Singol Multilayer Environment
Based on XML"

[9] URL: https://www.msu.edu/user/schuler4/diss msu.html

[10] Eleanor Selfridge- Field
"Melodic similarity concepts, procedures, and applications"
Computing musicology 11
Hewlet, 2000

[11] Eleanor Selfridge- Field
"Music Query"
Computing in Musicology, Vol 13
MIT Press, 2005

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
"Introduction to Algorithms"
MIT Press and McGraw-Hill, 2001.

[13] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
"Automi, Linguaggi e Calcolabilità"
Pearson Education Italia, 2009

[14] Dan Gusfield
"Algorithms on strings, trees, and sequences: Computer science and Computational biology"
Cambridge University Press, 1997.

[15] Simon Robinson, Christian Nagel, Jay Glynn, Morgan Skinner, KarliWatson, Bill Evjen
"Professional C#" Third Edition
Wiley Pub