Projet SY16

Identification automatique de fichiers audio.

Projet réalisé par :
Linda CALVARIN Johan MATHE
Dernière version : 10 janvier 2008

Table des matières

Table des figures

Liste des algorithmes

Problématique - définition des objectifs

L'objectif de ce projet est de mettre en place un système de reconnaissance automatique de chansons par rapport à leur contenu. Il s'agit plus précisément de trouver un moyen d'effectuer une empreinte d'un fichier audio indépendamment de la fréquence d'échantillonage, de la qualité, et du format de compression de cet dernier.

Chapitre 1

Bibliographie

Nous allons dans cette partie aborder les domaines que nous avons explorés avant de finalement choisir une solution viable.

1.1 Modèles auto régressif

Dans une première approche, nous nous sommes orientés vers des estimations de coefficients auto régressifs. L'idée était en effet d'estimer des coefficients AR sur des fenêtres de taille fixe.

1.1.1 Calcul des modèles AR

Le modèle AR(p) est donné par l'équation :

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t.$$

Il est basé sur les paramètres φ_i où $i = 1, \ldots, p$. Ces paramtères peuvent être estimés en utilisant un méthode des moindres carrés ou bien par les équations de Yule-Walker

It is based on parameters where i = 1, ..., p. Those parameters may be calculated using least squares regression or the Yule-Walker equations:

$$\gamma_m = \sum_{k=1}^{p} \varphi_k \gamma_{m-k} + \sigma_\varepsilon^2 \delta_m$$

où m = 0, ... , p, ce qui donnen p + 1 equations.

γ_m est l'autocorrélation de la fonction X, σ_ϵ est l'écart type du bruit, et $\delta_m$est la fonction delta de Kronecker.

Etant donné que la dernière partie de l'équation est non nulle si et seulement si m=0, l'équiation est habituellement résulue grâce à la matrice pour m>0

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} \gamma_0 & \gamma_{-1} & \gamma_{-2} & \cdots \\ \gamma_1 & \gamma_0 & \gamma_{-1} & \cdots \\ \gamma_2 & \gamma & \gamma_0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \vdots \end{bmatrix}$$

qui résout tous les φ.

Pour m=0, on obtient :

$$\gamma_0 = \sum_{k=1}^{p} \varphi_k \gamma_{-k} + \sigma_\varepsilon^2$$

ce qui nous permet de déterminer σ_ϵ^2.

1.1.2 Avantages/Défauts de cette approche

L'avantage de cette méthode est incontestablement sa simplicité algorithmique par rapport aux méthodes temps/fréquence que nous aborderons plus loin dans cette bibliographie. En effet, les équations de Yule Walker permettent de trouver très simplement les coefficients AR sur des petites fenêtres.

A l'inverse, le gros défaut de cette méthode réside dans le fait qu'évidemment nos signaux ne sont pas stationnaires et que l'approximation de dire que notre signal sera stationnaire sur une fenetre n'est bien évidemment pas vraie dans une écrasante majotité des cas. De plus, les tests montrent que cette méthode est très sensible aux bruits. On peut voir dans le graphique 1.1.3

1.1.3 Essais effectués

L'approche retenue pour les essais avec les coefficients AR était de former une empreinte avec les coefficients d'un extrait de musique d'une dizaine de secondes. Les coefficients ont été estimés grâce à la méthode de Yule Walker sur des fenêtres successives qui ne se chauvauchent pas d'une durée de 100ms.

Les coefficients ainsi estimés forment un vecteur de dimension N qu'il est aisé de comparer avec une base de données d'autres coefficients. Il suffit en effet de calculer l'erreur quadratique et de choisir la chanson dans la base de données qui minimise cette erreur quadratique.

Algorithm 1 Construction du vecteur de coefficients AR

```
[sig,Fe] = lire_fichier('audio.wav');
Ws = 0.1; Taille de la fenetre en secondes
Ncoefs = 20 Nombre de coefficients AR à estimer
coefs = 0; Initialisation du vecteur de valeurs
for i=300:400
concater_coefs(coefs,estime_coefs_ar(sig(floor(i*Ws*Fe:(i+1)*Ws*Fe)),Ncoefs)];
end
```

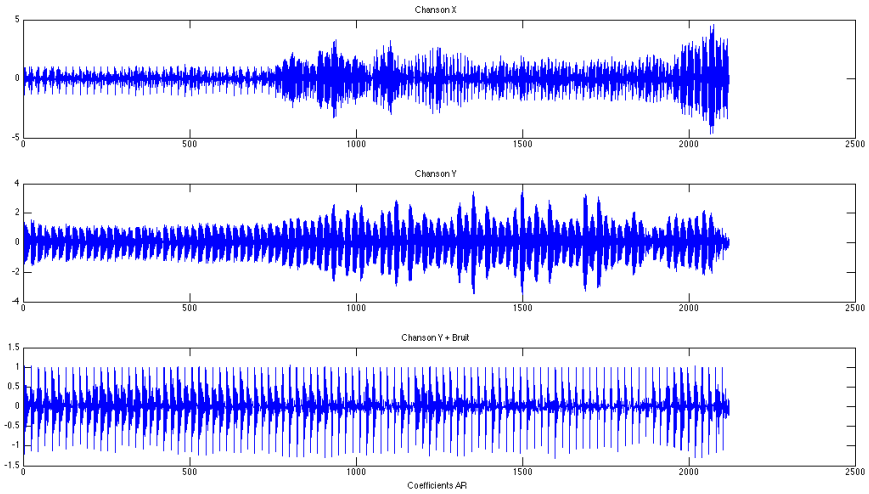

FIG. 1.1 – Vecteurs de coefficients AR

Ajout d'un bruit

En vue de faire des tests de bon fonctionnement on ajoutera un bruit blanc centré à notre signal audio original. Modifier la puissance de ce bruit nous permet de tester la robustesse de notre estimation et ainsi de simuler une dégradation liée à une recompression ou autres.

Algorithm 2 Ajout de bruit

```
Pnoise = 0.01;
y = wgn(1,length(sig),0.001).*PNoise;
sig_bruit = sig + y'; % Ajout du bruit au signal original
```

Comparaison des vecteurs

Une fois les vecteurs obtenus, il faut trouver un critère de comparaison entre ceux ci. Comme dit précédemment, il s'agit ici d'une facon assez simple de minimiser l'erreur quadratique de toutes les composantes du vecteur. Ainsi, si l'on a deux vecteurs X et Y, de même dimension 1xn on aura :

$$e = \sum_{i=1}^{n} (X[n] - Y[n])^2 = (X - Y).(X - Y)^T$$

Une simple opération de tri peut nous permettre de sélectionner la valeur de e la plus petite et ainsi de nous donner la chanson la plus proche dans une base de données

1.2 Analyse rythmique

Un de nos axes de recherche était la reconnaissance d'instruments en contexte polyphonique, et plus précisément la reconnaissance de signaux de batterie en présence d'instruments harmoniques. La plupart des instruments de musique produisent des sons harmoniques, dits non-percussifs, à l'inverse des instruments de percussions, qui produisent des sons à fréquence élevée sur un intervalle de temps très court. Les instruments harmoniques ont de plus des harmoniques qui n'excédent pas 6kHz. Il s'agit donc de détecter les impulsions provenant d'une batterie, en conservant dans le signal la plage de fréquences occupée par les sons de grosse caisse, caisse claire, tom, etc. La méthode utilisée pour extraire la source percussive du signal polyphonique est tout d'abord de réaliser un filtrage passe-haut à 7000Hz avec une fenêtre de Hanning.

A l'écoute du signal ainsi filtré on entend clairement tous les sons de la batterie, de ceux provenant de la caisse claire à ceux de la grosse caisse, d'intensité plus forte. Quelques sons harmoniques subsistent, il s'agit des amorces de note du signal de la voix humaine. L'intensité des signaux est également diminuée. Pour remédier à ce problème, on divise les valeurs du signal filtré par leur variance, afin d'en extraire les pics les plus courts sur l'échelle temporelle. En passant au logarithme les valeurs ainsi obtenues, on accentue encore le phénomène.

Les sons d'intensité courte et de fréquence élevée sont identifiés par les valeurs maximale sur ce nouveau signal. On peut alors réaliser le vecteur des instants pour lesquels on a une impulsion en provenance de l'instrument percussif.

Algorithm 3 Code matlab pour le filtrage

```
%ESSAI GIMMICK A 22050HZ
%lecture du signal d'origine
clear all;
[y,Fe,Nbits]=wavread('gimmick-22050Hz.wav');
figure(1),plot(y), axis tight;
title('signal d''origine');

%lecture et affichage du signal filtré
load export;
figure(1),hold on,plot(gimmickF4.data,'g');
title('signal filtré(vert) comparé au signal d''origine');
%Ecriture du fichier .wav permettant d'écouter le signal filtré
wavwrite(gimmickF4.data,Fe,Nbits,'Filtrage4-gimmick.wav');
```

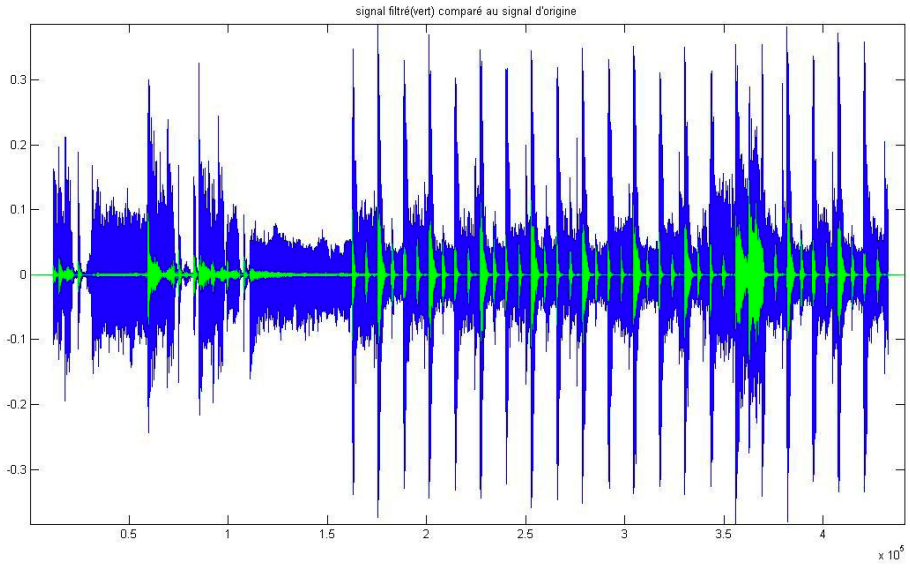

FIG. 1.2 – Extraction de la rythmique d'un morceau

1.2.1 Inconvénient de la méthode

Cette méthode ne tient pas compte du fait que la voix humaine peut également produire des harmoniques à fréquences élevées, tout particulièrement dans le cas où le chanteur est une femme. Elle fonctionnerait très bien pour une musique essentiellement instrumentrumentale. La méthode est uniquement applicable dans les cas où un instrument percussif est présent dans la chanson. Le signal doit être finement échantillonné, de façon à prendre en compte les impulsions de batterie sur des intervalles de temps assez faibles, ce qui mène rapidement à des traitements lourds.

1.2.2 Intérêt de la méthode

Possibilité de produire une empreinte sur toute la durée de la chanson (vecteur 1 * n)

1.3 Estimation de la fréquence fondamentale

Une grande partie de la bilbliographie s'est faite à la biliothèque de l'IRCAM[1]. Le but était de trouver les articles les plus pertinents au niveau de l'estimation de fréquence fondamentale.

D'après les chercheurs de l'Ircam, la démarche la plus pertinente dans notre cas aurait été d'utiliser un modèle quasiharmonique vu dans [MIREX7] utilisé dans de la forme suivante :

$$y[n] = \{ \sum_{m=1}^{M} \sum_{h_m=1}^{H_m} a_{m,h_m}[n] \cos((1 + \delta_{m,h_m}) h_m \omega_m n + \phi_m[n]) \} + v[n]$$

où n est l'index temporel, M le nombre de sources, H_m le nombre d'harmoniques pour la mième source, ω_m représente la fondamentale de la source m et $\phi_m[n]$ dénote la phase. $v_n$représente le bruit. Ce dernier est considéré assez petit pour permettre de détecter la sinusoïde.

1.3.1 Avantages/Inconvénients

Après réflexion, cette méthode a l'avantage de bien coller à la réalité du modèle physique d'un fichier audio, mais présente le gros désavantage de devoir procéder à l'estimation de beaucoup de paramètres comme le nombre de sources, le nombre d'harmoniques etc.

De plus ce modèle nécessiterai un très long travail de précision pour être générique. En plus, ce modèle est sensible au bruit, ainsi qu'à certains formats de compression comme le MP3, qui détruisent certaines fréquences inaudibles, mais essentielles pour l'estimation de certaines harmoniques élevées.

1.4 Estimation spectrale

On peut faire ici une application directe du cours de SY16. Le spectrogramme est en effet la plus usuelle des représentations temps fréquence. On a en effet le temps en abcisse et la fréquence en ordonnée, ce qui permet littérallement de "voir les fréquences" grâce à un astucieux système de couleurs, on peut savoir directement ou sont représentées les fréquences les plus intenses.

Le paramètre le plus sensible dans l'estimation spectrale est la taille de la fenêtre sur laquelle la transformée de Fourier sera effectuée. En effet ce paramètre devra être choisi de sorte à ce que l'information privilégiée (temps ou fréquence) soit la plus pertinente.

1.4.1 Avantages - Inconvénients

L'avantage de cette technique est incontestablement qu'elle est proche de ce qu'entend l'oreille humaine. Elle va donc contenir les informations pertinentes d'un point de vue psycho-acoustique. De plus si l'on s'y prend bien, il est possible de s'affranchir des problèmes liés à l'encodage (surtout si l'on regarde les basses fréquences, étant donné que les formats comme le MP3 ou le ogg-vorbis compressent en supprimant les composantes fréquentielles inaudibles).

Le gros inconvénient lié à cette technique provient surtout de la complexité algorithmique liée au spectrogramme. Il est cependant possible de diminuer cette complexité en faisant un réechantillonage du signal en amont. Ce que nous verrons dans la partie 2.

[1] IRCAM : Institut de Recherche et Coordination Acoustique/Musique

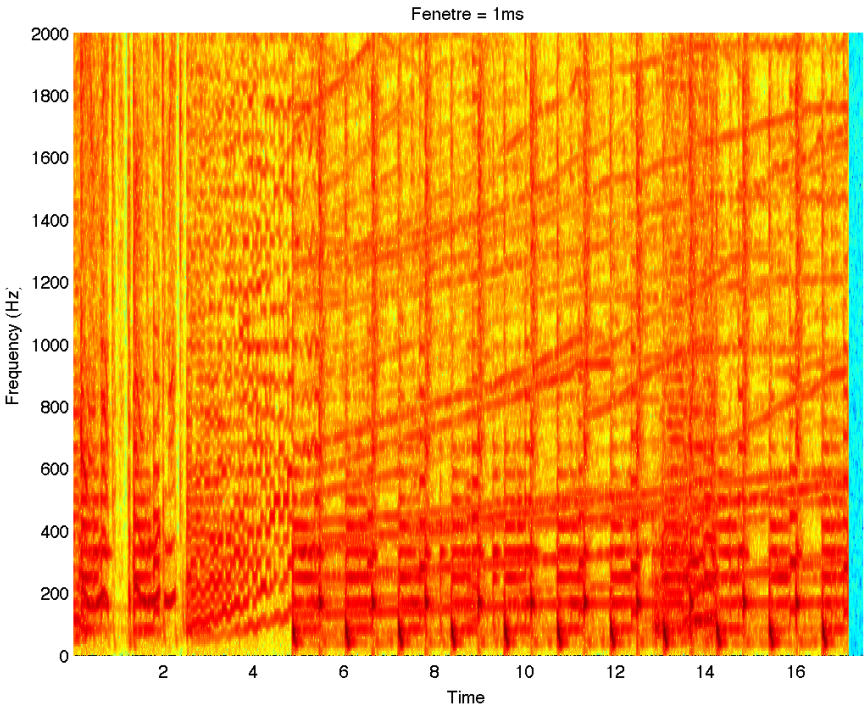

FIG. 1.3 – Exemple de spectrogramme

Chapitre 2

Mise en place de l'analyse automatique

Nous verrons dans ce chapitre comment précisément nous avons procédé pour l'analyse automatique des signaux

2.1 Extraction d'une empreinte caractéristique

Nous allons voir ici les techniques permettant l'extraction d'une empreinte unique par morceau de musique.

2.1.1 Rééchantillonage

Après avoir fait quelques expériences, ils s'avère qu'un rééchantillonage à basse fréquence (1KHz) donne de très bons résultats pour plusieurs raisons :

- Réduction de la complexité algorithmique : en effet ce rééchantillonage fait en sorte que l'algorithme de FFT utilisé pour calculer le spectrogramme doit travailler sur beaucoup moins de valeurs.
- Plus grande stabilité de l'algorithme d'exctraction de fréquence la plus intense : l'empreinte étant réalisé par extraction de la fréquence la plus intense sur le spectrogramme, le fait d'appliquer un passe bas à 500 Hz vas nous permettre d'extraire avec plus de précision une fréquence plus proche de la fondamentale. On aura ainsi beaucoup moins de chances de sauter d'une harmonique à l'autre.
- Suppression des bruits hautes fréquences et indépendance du format de compression choisi. En effet les formats de compression usuels ont pour habitude de couper les fréquences inaudibles. Ainsi le fait d'effectuer un passe bas en amont de l'analyse rendra celle ci plus robuste au bruit.

Algorithm 4 Passe bas et rééchantillonage sous matlab

```
1 x=resample(y,Fe_new,Fe);
```

2.1.2 Spectrogramme

Une fois le passe bas effectué, on va créer un spectrogramme d'un extrait du morceau. On prendra pour cela un échantillon d'une dizaine de secondes au milieu de la musique dont on veut réaliser l'empreinte sur lequel on effectuera le spectrogramme.

Algorithm 5 Spectrogramme sous matlab

```
1 [s,f,t]=spectrogram(x(10000:50000),Nw,fix(Nw*.98),512,Fe_new);
```

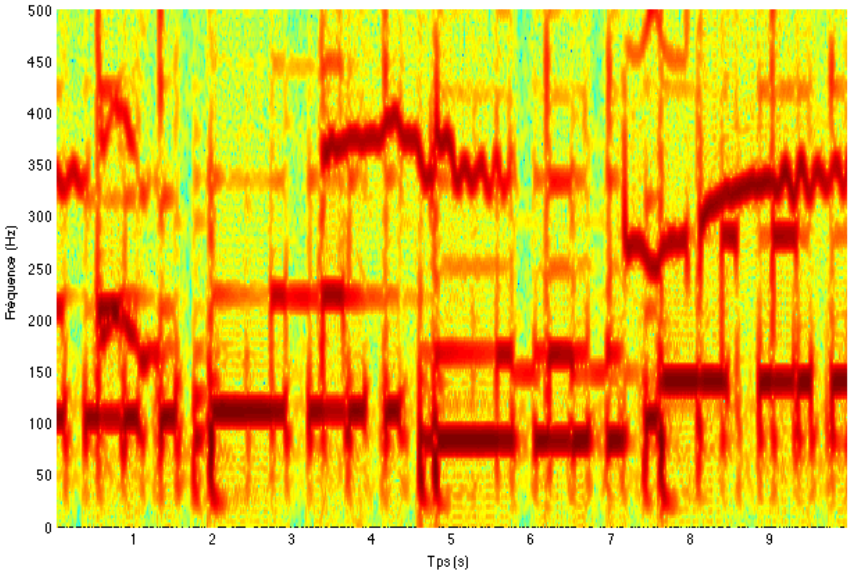

FIG. 2.1 – Spectrogramme d'une chanson

On voit clairement sur la figure du spectrogramme 2.1.2 qu'on peut facilement détecter la fondamentale ou au moins la fréquence la plus importante entre 0 et 500Hz.

2.1.3 Détection de la fréquence dominante

Après avoir extrait le spectrogramme d'un échantillon d'un morceau de musique, il faut maintenant extraire une empreinte correspondant aux fréquences les plus intenses en fonction du temps. Pour ca, une simple détection de maximum nous permet d'obtenir un vecteur contenant cette information

Algorithm 6 Extraction du maximum fréquenciel

```
1 [a,c]=max(abs(s));
```

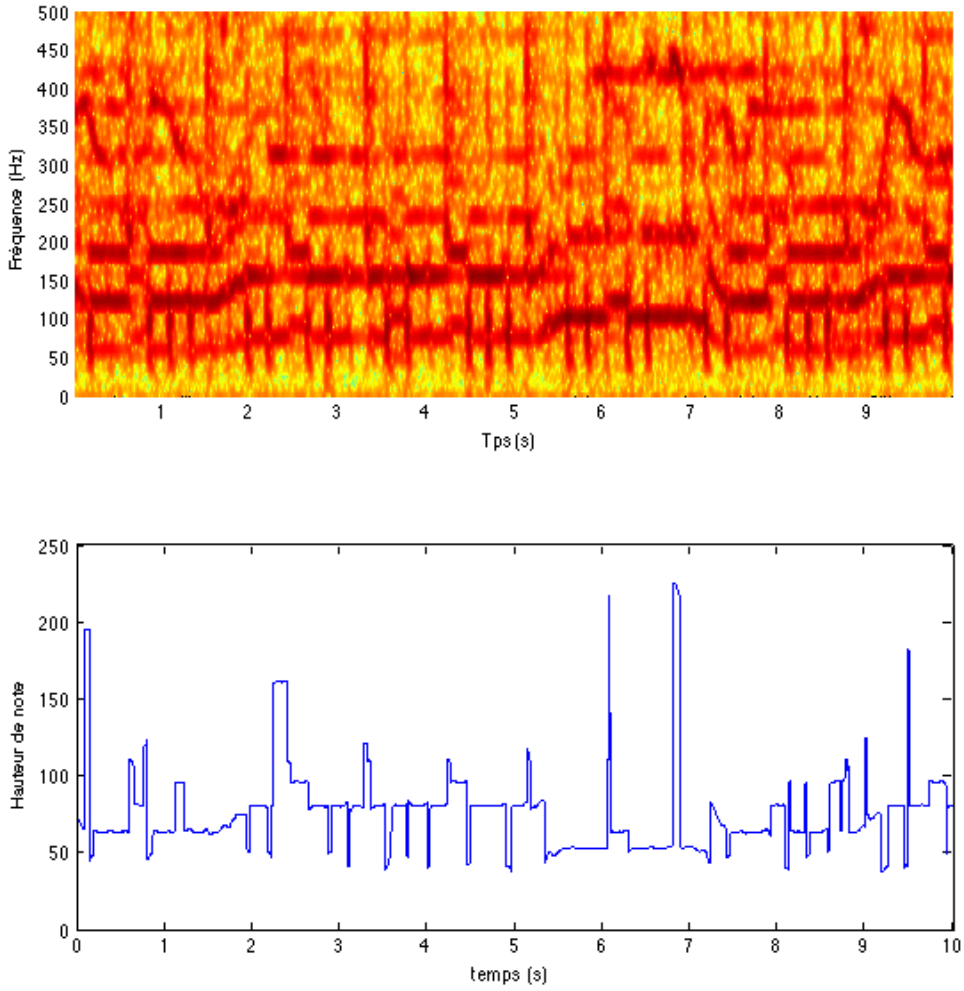

FIG. 2.2 – Spectrogramme et hauteur de note

2.2 Construction d'une base de données de musiques

Maintenant que nous avons réussi à extraire une empreinte digne de ce nom pour une musique donnée, il s'agit de créer une mini base de données des oeuvres pour pouvoir faire la reconnaissance d'une parmi les autres. Pour cela, on va simplement créer une matrice contenant les vecteurs empreintes.

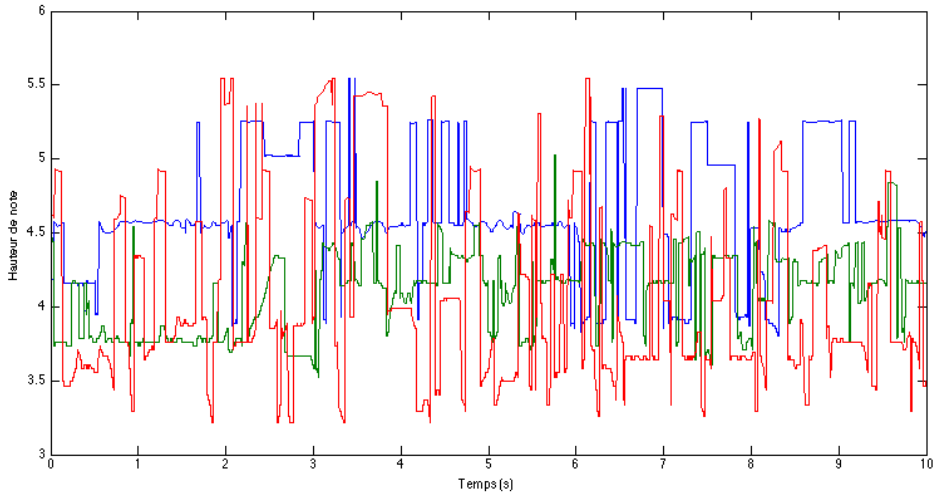

FIG. 2.3 – Plusieurs extraits musicaux représentés

On voit bien sur le graphique représentant les hauteurs de notes qu'elles sont toutes bien différentes et qu'on peut difficilement les confondre

2.3 Reconnaissance d'une chanson dans la base de données

2.3.1 Mise en place d'une relation d'ordre

Pour pouvoir comparer notre extrait avec les extraits présents dans la base de donnée, nous avon besoin d'une relation d'odre total dans notre ensemble d'extraits. Nous avons aussi besoin d'une méthode qui soit robuste aux décallages temporels.

2.3.1.1 Intercorrélation

Une méthode qui résiste bien aux décallages temporels sera d'effectuer l'intercorrélation entre l'extrait à tester et tous les élements de notre base de données. Une fois normalisées (on divisera ici par l'écart type du signal pour cela) on devrait obtenir un graphe qui aura l'allure d'une auto-correlation, le théorême de cauchy schwartz nous assure que le résultat de notre intercorrélation présentera un maximum en zéro (avec un potentiel décallage au cas ou les enregistrements soient décallés).

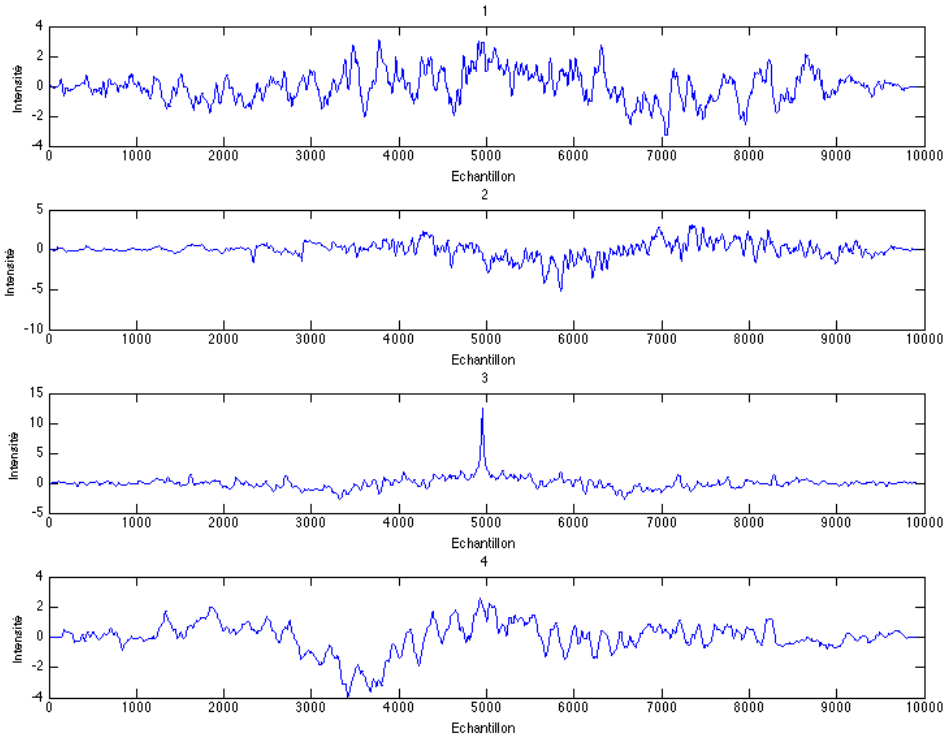

FIG. 2.4 – Intercorrélations entre signaux de la bdd

2.3.1.2 Ordre

Une fois ce maximum d'intercorrélation effectué pour chaque élément de la base, on obtient un critère qui nous permet de créer notre relation d'ordre. On a bien ici une relation d'ordre car on peut aisément démontrer que la relation est transitive, antisymétrique et réflexive.

2.3.1.3 Extraction du maximum

Une simple extraction de maximum nous permet de détecter quelle chanson sera la plus pertinente dans notre cas.

Algorithm 7 Extraction du maximum

```
for i=1:Nmusic
    % Exctraction de la corrélation
    y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
    % Normalisation du vecteur
    sd = sqrt(var(y));
    y=y./sd;
    % Extraction du maximum yv = valeur et ypos la pos
    [yv,ypos]=max(y);
    if(yv > vmax)
          vmax = yv;
          num = i;
          % Détection de l'offset de notre chanson
          % Par rapport à celle dans la BDD
          offset = ypos-length(y)/2
          offset = offset*2/Fs;
    end
end
```

Cerise sur le gateau, notre algorithme est en mesure de dire le décallage de la musique analysée par rapport à la musique dans la BDD. Ceci grace à la position du pic d'intercorrélation.

2.4 Tests de robustesse

2.4.1 Ajout d'un bruit

Le premier test de robustesse de robustesse a été l'ajout d'un bruit blanc en vue de mesurer l'impact de celui ci sur la mesure

Algorithm 8 Ajout de bruit

```
y = wgn(length(sig),1,0.001).*PNoise;
sig_bruit = sig + y;
```

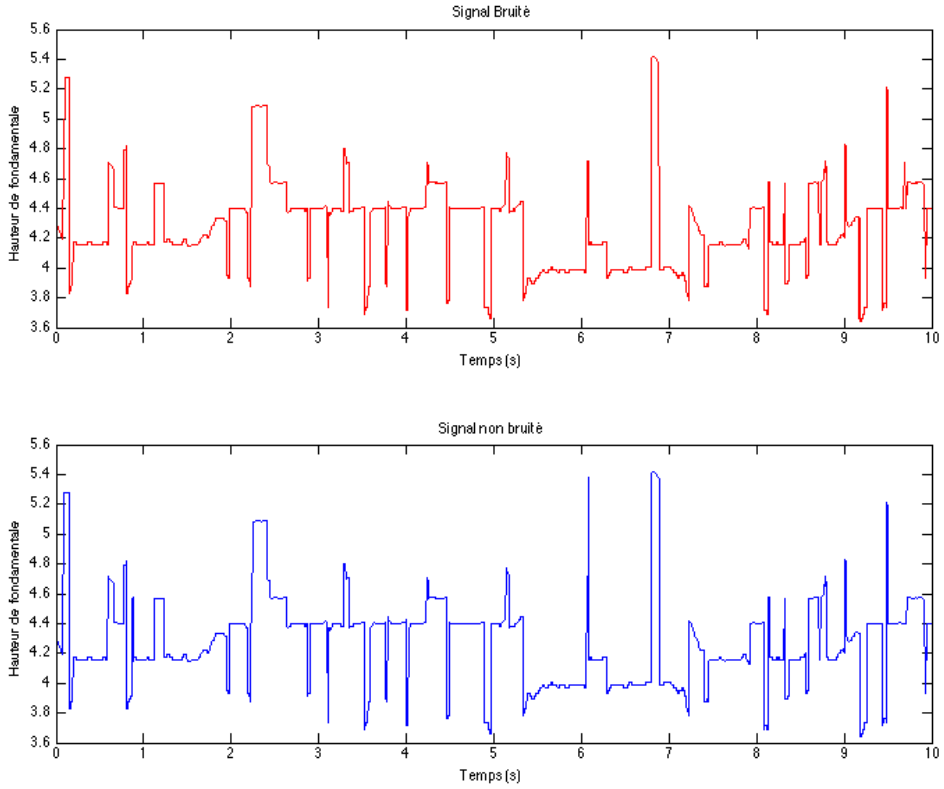

FIG. 2.5 – Comparaison signal bruité/ Non bruité

On voit clairement sur la figure 2.5 que même après ajout d'un bruit d'une puissance non négligeable (audible), on reste toujours dans la plupart des cas très proche de l'empreinte dans la base de données.

2.4.2 Test avec deux musiques de deux sources différentes

Le test ultime de notre système consiste à comparer une musique trouvée sur internet sur un site présentant des vidéos de qualtité médiocre avec une bibliothèque, sachant que notre chanson à tester est dans la base de données.

Ce test réalisé a donné de très bons résultats avec notre méthode, en effet, le bon fichier a été trouvé dans la base de données, de plus le bon décallage a été trouvé.

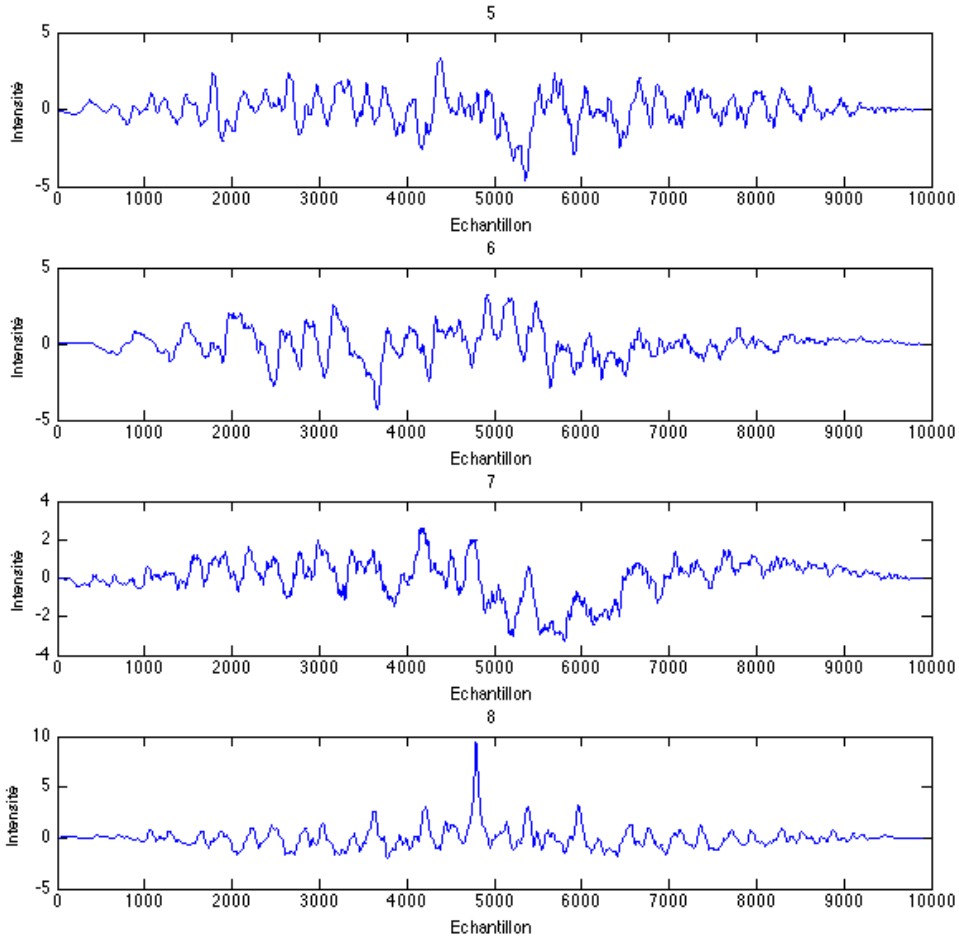

FIG. 2.6 – Diverse intercorrelations dans la bdd

Comme on le voit dans la figure 2.6 un pic se dégage nettement pour le signal 8, ce qui correspond exactement au signal en question dans la base de données.

2.5 Ouvertures quant à la complexité et l'algorithmique

2.5.1 Complexité

Une première analyse quand à la compexité de nos algorithmes peut se présenter sous la forme suivante :

1. La complexité mémoire est assez intéressante. En effet on se retrouve à stocker une empreinte d'environs 5ko par morceau de musique, et on obtient quelque chose qui n'explose pas en fonction du nombre de musiques dans la base de données, on reste donc en $O(n)$ au niveau mémoire.

2. La complexité en temps est un peu plus complexe à analyser. En effet, pour chaque morceau de musique, on doit effectuer une intercorrélation avec le morceau à analyser, ce qui

fait que dans tous les cas nous sommes amenés à comparer tous nos morceaux. D'une part l'intercorrélation présente une complexité $O(n^2)$ par rapport à la taille de l'échantillon ($5.10^3.5.10^3 = 25.10^6$ opérations). De plus nous sommes en $O(n)$ par rapport au nombre de musiques dans la biliothèque. On se retrouver donc à terme avec une complexité $O(nm^2)$ avec n le nombre de morceaux dans la base de données et m la taille de l'échantillon. Cette complexité est à la fois la complexité dans le cas moyen et la complexité dans le pire des cas, les deux étant confondues dans ce cas précis étant donné qu'il faut parcourir toute la base.

2.5.2 Optimisations potentielles

En vue d'optimiser la recherche dans notre base de données, il faudrait pouvoir passer à une complexité logarithmique. La seule facon pour cela est de trouver un critère de tri sur notre base de données de sorte à ce que la base de données soit conservée triée et que les accès à une chanson précise soit faite en log(n).

Chapitre 3

Annexes

3.1 Code matlab du programme principal

```
%% Fichier de test pour projet SY16
f1 = 'samples/president.wav';
f2 = 'samples/bach_toccata.wav';
f3 = 'samples/nyd.wav';
f4 = 'samples/radiohead_calm.wav';
f5 = 'samples/guitar.wav';
f6 = 'samples/music.wav';
f7 = 'samples/radiohead.wav';
f8 = 'samples/cake_orig.wav';
fb = 'samples/cake.wav';

T=1000;
Ws = 0.1;
tstart = 55;
tend = 65;
Fs = 1000;

%
% Reading songs :
% [sig,fe] = wavread(f1);
% sig = sig(:,1);
% c(1,:)= log(get_freq(sig,fe,Ws,tstart,tend,Fs));
% 1
%
% [sig,fe] = wavread(f2);
% sig = sig(:,1); % c(2, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
% 2
%
% [sig,fe] = wavread(f3);
% sig = sig(:,1);
% c(3, :)  = log(get_freq(sig,fe,Ws,tstart,tend,Fs));

% 3 % % [sig,fe] = wavread(f4);
% sig = sig(:,1);
% c(4, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));

% 4 % % [sig,fe] = wavread(f5);
% sig = sig(:,1);
% c(5, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
```

```
40
41 % 5 % % [sig,fe] = wavread(f6);
42 % sig = sig(:,1);
43 % c(6, :)   = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
44
45 % 6
46 %
47 % [sig,fe] = wavread(f7);
48 % sig = sig(:,1);
49 % c(7, :)   = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
50 % 7
51
52 %
53 % [sig,fe] = wavread(f8);
54 % sig = sig(:,1);
55 % c(8, :)   = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
56 % 8
57 %
58
59
60
61 % Bruitage
62 [sig,fe] = wavread(fb);
63 sig = sig(:,1);
64 PNoise = 0.01;
65 %y = wgn(length(sig),1,0.001).*PNoise;
66 sig_bruit = sig; % + y; cb = log(get_freq(sig_bruit,fe,Ws,tstart,tend,Fs));
67 %
68 % Affichage de toutes les courbes figure(2) for i=1:8 subplot(3,3,i);
69 y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
70 sd = sqrt(var(y)); y=y./sd; plot(y); title(i) xlabel('Echantillon');
71 ylabel('Intensité'); end
72 vmax = 0; sc = size(c);
73 Nmusic = sc(1);
74
75 for i=1:Nmusic
76  y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
77  vy = var(y);
78  y=y./sqrt(vy);
79  [yv,ypos]=max(y);
80
81   if(yv > vmax)
82    vmax = yv;
83    num = i;
84    length(y)
85    offset = ypos-length(y)/2
86    offset = offset*2/Fs;
87   end
88
89 end
90 disp('La chanson la plus proche est la numéro');
91 disp(num);
92 disp('Offset temporel :');
93 disp(offset);
```

3.2 Code matlab de la fonction get_freq

```
1 function [c] = get_freq(y,Fe,Ws,tbegin,tend,fe_n)
```

```
% fonction permettant de convertir une chanson au format wav en frquences
% y: signal
% Fe : fréquence d'échantillonage de y
% Ws : taille de la fenetre du temps fréquence
% tbegin : temps de départ pour l'analyse
% tend : temps de fin pour l'analyse
% fe_n : nouvelle fréquence d'échantillonage
% les plus dominantes
% Conversion signal .wav sous Matlab
Te=1/Fe;
N=length(y);
t1=0:Te:(N-1)*Te;
x=resample(y,fe_n,Fe);

%Analyse temps frequence du signal audio
% Nb d'échantillons dans la fenetre
Nw=fix(Ws*fe_n);
% Calcul des valeurs du spectrogramme
[s,f,t]=spectrogram(x(tbegin*fe_n:tend*fe_n),Nw,fix(Nw*.98),512,fe_n);
subplot(211);
surf(t,f,log(abs(s)),'EdgeColor','none');
%axis xy;
%axis tight;
%colormap(jet);
%view(0,90);
%ylabel('Fréquence (Hz)');
%xlabel('Tps (s)');
%subplot(212);
%plot(c);
%ylabel('Hauteur de note');
%xlabel('echantillon (ech)')
[a,c]=max(abs(s));
```