INF230 - AudioFingerprinting Project

François Blas[1]*, Anthony Reinette[2], Florian Riche[3]

Abstract
In the scope of the INF 230 [Mining of Massive Datasets] module in Telecom ParisTech, students had the opportunity to realise an optionnal project related to a subject of their choice. This paper is the report of our project whose aim was to reproduce an audiofingerprinting algorithm. First we will present the theory and the technical knowledge required to understand our project. Then we will explain how we organised ourselves to realise it. We will then explain how we implented our project before discussing the results.
The different papers we read related to our subject were very dense and didn't explain the theory in depth. During the writing of this paper, we tried to be more exhaustive in our explanations. As a consequence, even though the professors might find this paper too verbose, we believe it is a good introduction to the subject for someone who has no previous related technical knowledge.

Keywords
WavePrint — Audio — Fingerprinting — MinHash — MapReduce — Haar wavelets

[1] *Mastère spécialisé Big Data, Telecom ParisTech, Paris, France*
[2] *Mastère spécialisé Big Data, Telecom ParisTech, Paris, France*
[3] *Mastère spécialisé Big Data, Telecom ParisTech, Paris, France*
***Contact:** francois.blas@telecom-paristech.fr, anthony.reinette@telecom-paristech.fr, florian.riche@telecom-paristech.fr
***Github repository** `https://github.com/florianriche/AudioFingerPrinting`

Contents

Introduction

Audio fingerprinting is a method used to recognize snippets of sound and link them to their corresponding data. For instance, this method allows one to record a song playing from speakers to its smartphone and retrieve data such as the artist name and the title of the song.

The Audio Fingerprinting method gives the ability to build a database of original audio content along with its data. This method is also used to query the said database, and match a snippet of sound to the original audio content.

Audio fingerprinting has numerous applications : One of the most famous example is given by the smartphone application called Shazam. Shazam recognizes music and media playing around the user. It allows users to record an 30 seconds long extract of the sound the user wants to identify and provides the user with information about the original content.

Another example is Echo Nest which is a cloud based music intelligence platform. Thanks to its API, the service is able to recognize songs and give comprehensive insights

about the song such as the artist, the tempo of the song, the tonality etc ... Besides from recognizing snippets of recorded songs, audio fingerprinting can also be used to monitor media for copyright infringement, as illegal use of music (especially in Youtube videos) is getting more and more important.

Audio fingerprinting is a complicated task, and have to take into account many factors to become an efficient method. The method should be :

- Able to identify a song from an extract, with the length of the extract being the shortest possible.
- Able to identify a song from an extract, with a varying quality of the recording.
- Able to identify a song from an extract, with the extract starting anytime in the song.
- Independent to the ambient noise.

Having all these constraints in mind, this paper is going to present how we implemented the audio fingerprinting method, and built a database of songs.

1. Team and Project Organisation

1.1 Initials objectives

The first idea of this project came when we read an article explaining in details how the shazam program works [15]. The author of this article implemented his own version of the shazam software.

After the MapReduce lesson we had in Telecom ParisTech, we thought it would be an interesting exercice to implement our own version of an audiofingrprinting algorithm using mapReduce in order to identify a whole library all at once.

After a few research, we found another similar work that implemented a slightly different audiofingerprinting algorithm : The Google Waveprint algorithm [1][2]. That's why we decided to implement this one as the arguments for this choice were clearly explained.

The initial plan of our project was to implement only the first part of the process, the fingerprint creation, and not to implement the retrieval process.

1.2 Project Organisation

Our project was split in the four following steps :

1. Documentation and clear understanding of the algorithm
2. Allocation of an algorithm part to each team member
3. Bonding of the algorithm pieces
4. Redaction of the present report

During the documentation process, we found many papers related to the subject and decided to implement the waveprint algorithm. The different theoritical bases were the subjects of further documentation research. Besides, in order to know which database system would be the best for our implementation, we asked for the help of a Telecom ParisTech Teaching assistant.

The development phase was done using an Agile method with regular meeting where team members could share their progress and their obstacles.

2. The Waveprint algorithm

2.1 The STFT and the Spectogram creation

The first step of the audio fingerprinting process consist to generate a spectrogram of the song. A spectrogram [7] is a visual representation of the spectrum of frequencies in a sound as they vary with time. This is one of the best representation, in signal's field, to analyze and identify a song by frequencies. A sound spectrogram looks like that :

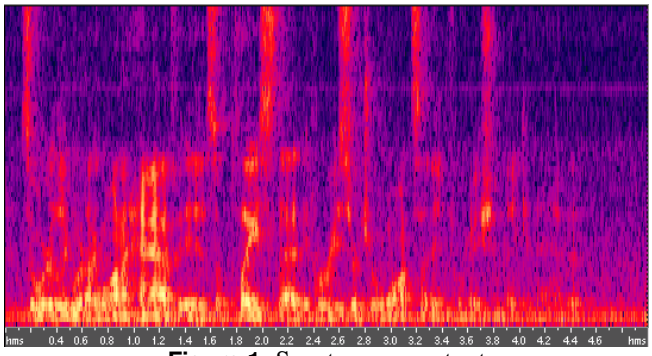


Figure 1. Spectrogram output

This 2D representation is very common : the horizontal axis represents time , the vertical axis is frequency and a third dimension indicating the magnitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image. A spectrogram can be estimated by computing a STFT.

2.1.1 FTFT

Before explaining what's a STFT, it's convenient to have a look on the FFT [4]. The FFT, or Fast Fourier Transform, is an algorithm to compute the Discrete Fourier Transform (DFT [3]). Fourier analysis converts time to frequency and vice-versa. The FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. The most popular algorithm to compute the FFT is the one named Cooley–Tukey[5], and its simpliest form is the Radix-2. The principle of the algorithm is to divides a DFT of size N into two interleaved DFTs of size N/2 with each recursive stage. If we define the DFT by this formula (with $0 < k < N\text{-}1$) :

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

Radix-2 first computes the DFTs of the even-indexed inputs and of the odd-indexed inputs , and then combines those two results to produce the DFT of the whole sequence. This

idea can then be performed recursively to reduce the overall runtime to O(N log N). This simplified form assumes that N is a power of two. With this idea, the formula of the DFT is rearranged as follow :

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

2.1.2 STFT

The STFT or Short Time Fourier Transform [6] is a process of the Fourier Transform family. It's used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. Basically it's multiple FFT performed over time on the sound signal. To compute the STFT on the sound, we choose a time frame and shift this frame into time. For each shift operation we compute the FFT on the signal existing into the frame.

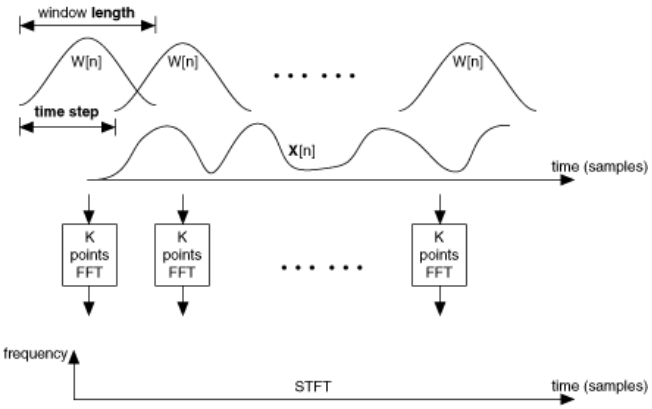


Figure 2. STFT principle

One important parameter is the accuracy (or resolution) of the process. The width of the windowing function relates to how the signal is represented; it determines whether there is good frequency resolution or good time resolution. A wide window gives better frequency resolution but poor time resolution. A narrower window gives good time resolution but poor frequency resolution.

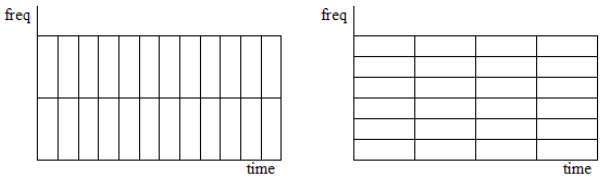


Figure 3. STFT time/frequency resolution tradeoff

Despite a good choice for the window's width, the process is limited by the Gabor limit. The result implies that the process cannot have both high temporal resolution and frequency resolution.

2.2 Haar Wavelet Decomposition

Wavelets are a mathematical tool for decomposing functions. They allow a function to be described by its overall shape, plus successively increasing details.

Wether the function is an image, a curve or a surface, wavelets are a time frequency transform that allows to represent a function with different levels of details. Often used in computer graphics problems such as image compression, or image querying, Haar Wavelets transformation play an important role in audio processing.

The Haar Wavelet [14] transformation consists in decomposing a function in the Haar wavelets basis. We will first have a look at the one-dimensional Haar Wavelet transform, then move on to the two-dimensional Wavelet transform and how we apply it to an image.

2.3 One-dimensional Haar Wavelets

2.3.1 Basis functions

Let's consider a one-dimensional image : It can be seen as a vector of pixels. Each pixel has a value between 0 and 255. Alternatively, this image can be represented as a piecewise-constant functions on the interval [0,1[.

A one pixel image is a function that is constant over the entire interval [0,1[. Let V^0 be the vector space of all these functions.

A two pixel image has two constant pieces over the interval [0,1/2[, and [1/2,1[. We will call this vector space V^1. Recursively, we create the space V^j, for images with 2^j pixels. We want to build a basis for the vector space V^j. To do so, we choose the following functions :

$$\sigma_i^j(x) = \sigma(2^j x - i), \text{ for } i = 0, ..., n$$

Where :

$$\sigma(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

For instance, we can see what a function of the V^3 vector space looks like :

We then choose an inner product, and for this, we choose the standard inner product :

$$\langle f | g \rangle = \int_0^1 f(x)g(x)dx$$

It is important to notice that $\forall j$, $V^j \subset V^{j+1}$. Given the internal product, we can define the W^j vector space, which is the orthogonal complement of V^j in V^{j+1}. A collection of linearly independent functions γ_i^j spanning W^j are called wavelets.

Where :

$$\gamma_i^j(x) = \gamma(2^j x - i)$$

And :

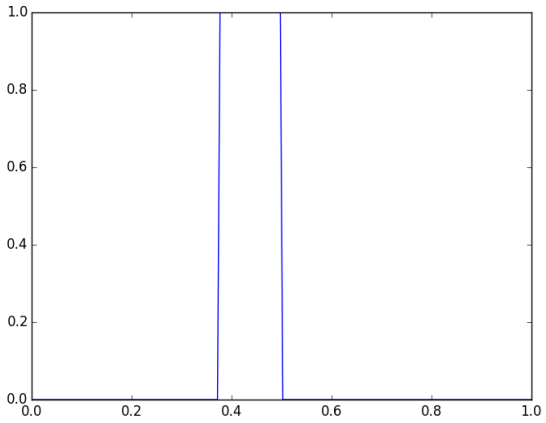


Figure 4. Exemple of a V^3 vector space function

$$\gamma(x) = \begin{cases} 1 & \text{for } 0 \leq x < 1/2 \\ -1 & \text{for } 1/2 \leq x < 1 \\ 0 \text{ otherwise} & \end{cases}$$

Below is a representation of γ_2^3

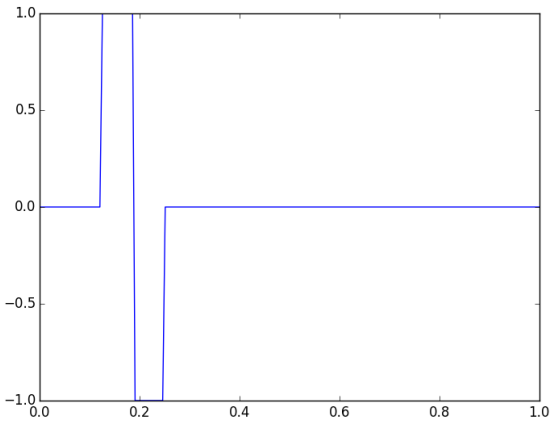


Figure 5. Exemple of a W^3 vector space function

We now have described two basis for expressing our images. Let's take an example : Let's consider an image decomposed in the V^2 basis :

$$\mathscr{I}(x) = c_0^2\sigma_0^2(x) + c_1^2\sigma_1^2(x) + c_2^2\sigma_2^2(x) + c_3^2\sigma_3^2(x)$$

We can rewrite $\mathscr{I}(x)$ as a sum of basis functions in V^0,W^0, and W^1 :

$$\mathscr{I}(x) = c_0^0\sigma_0^0(x) + d_0^0\sigma_0^0(x) + d_0^1\sigma_0^1(x) + d_1^1\sigma_1^1(x)$$

2.4 Two-dimensional Haar Wavelets

We can apply the one-dimensional wavelet transform to each row of pixel values. This operation gives us an average value along with detail coefficients for each row. Next, we treat these transformed rows as if they were themselves an image and apply the one-dimensional transform to each column.

INSERT EXAMPLE

2.5 Application to our problem

The Haar Wavelets gives us a decomposition of images in a function basis. They allow a description of an image by its overall shape. In our problem, we are going to **apply a Haar Wavelet transformation for every spectrogram produced**. This will give us the shape of the spectrogram.

To enhance the robustness of our algorithm, we want to make the spectrogram resistant to **noise and audio degradations**. To do so, we only **keep the strongest wavelets among all the wavelets (magnitude wise)** : We only keep the ones that most characterize the image. These wavelets are called the top-t wavelets.

Below is the comparison of the Haar Wavelet transformation of the spectrogram of an original song, with the Haar Wavelet transformation of the spectrogram of a recorded song both with the top-t wavelets kept. We can easily see identifiable patterns both in the wavelet space.

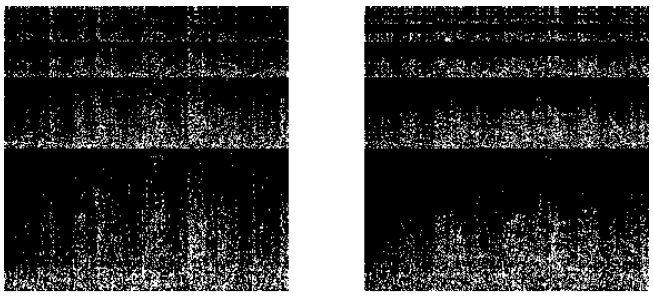


Figure 6. Comparison of two haar wavelets transformation of spectrograms

2.6 Signature creation

2.6.1 Vectorization

We now have a sparse matrix and need to store a unique identifier of this matrix under a compressed form. That is where the fingerprint operation appears. First, as the magnitudes can differ between two different recording of a songs, we will only keep the sign of the remaining magnitude. We will code the sign with two bits and store the resulting matrix into a single vector.

- 01 for negative magnitudes
- 00 for zero magnitudes
- 10 for positive magnitudes

And the transformation is as follow :

$$\begin{pmatrix} \underline{5} & 0 & \cdots & -10 \\ 0 & -7 & \cdots & 0 \\ 0 & \vdots & \cdots & 20 \\ 3 & \vdots & \cdots & 0 \end{pmatrix} \Rightarrow (\underline{1}00000100001\cdots01001000)$$

2.6.2 Similarity measure

Now we need a way to compare easily two bits vectors. The measure of similarity of two bits vector is the Jaccard measure. Namely, for two bits vector C_i and C_j the similarty is :

$$sim_j(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

When we compare the two vectors bit by bit, we can find 4 cases :

Type	C_i	C_j
A	1	1
B	1	0
C	0	1
D	0	0

From now on we will note A for card(A).
The Jaccard similarity can then also be expressed as :

$$sim_j(C_i, C_j) = \frac{A}{A+B+C}$$

2.6.3 MinHashing

When we want to compare two massive vectors, the comparison bit by bit can be extremely long. That's why we have to find a trick to approximate the similarity :the Min-Hash.

The basic idea is to randomely permute the rows of C_i and C_j and to define a function h(C_i) that gives the index of the first 1 in C_i and get the surprising property : inger

$$P[h(C_i) = h(C_j)] = sim_j(C_i, C_j)$$

h look down C_i and C_j until the first non 0, so both do not count D. In addition, h(C_i) = h(C_j) means that we are in the type A. And the probability that it appears can be expressed as $\frac{A}{A+B+C}$, which is the same as the similarity measure.

So, we are going to try to get this probability rather that directly searching for the similarity.

2.6.4 MinHashing signatures

At first we define P, the number of random permutations[1] vectors we are going to create.

We then define the MinHash Signature $sig(C)$ as the list of P indexes of first rows with 1 in column C Namely, we apply the h function for every different permutation.

[1] The permutations are the same for every C_i , C_j

Then we define the similarity of signatures.
$sim_H(sig(C_i), sig(C_j))$ is the fraction of permutations where MinHash values agree and it comes that :

$$E(sim_H(sig(C_i), sig(C_j))) = sim_j(C_i, C_j)$$

For a P big enough:

$$sim_H(sig(C_i), sig(C_j)) \approx E(sim_H(sig(C_i), sig(C_j)))$$

And it comes :

$$sim_H(sig(C_i), sig(C_j)) \approx sim_j(C_i, C_j)$$

That is this similarity we are going to look after to approach the similarity of our initial columns C_i and C_j.

As a consequence, we are not going to store the vector into the database but only its signature of length p. We call it " the fingerprint".

2.7 Database storage

The next step is to store our fingerprint vector into a database. We must store it with keeping in mind that we are going to use it to compare to other fingerprint.

Comparing directly a new fingerprint to every fingerprint in the database would be extremely heavy. As a consequence, we are going to split our fingerprint into l subfingerprints and hash those subfingerprints into l hash tables.

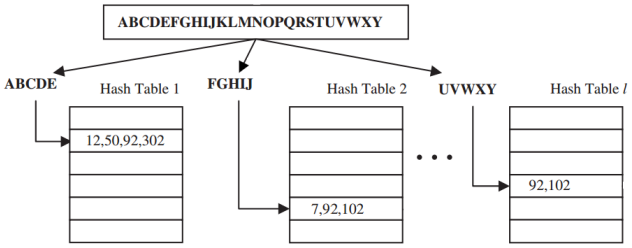


Figure 7. Illustration of the subfingerprints hashing

This storage method enables us to store a unique fingerprint for every spectrogram of a song with which we can easily compare.

2.8 Song retrieval

The last step of our algorithm to explain is the retrieval process.

We already explained that we compare the fingerprints of a song's spectograms. As a consequence, the beginning of the retrieval process is the same as the storage process. We create a spectrogram, apply a Haar transformation and a MinHash signature and end up with a fingerprint.

Now, as previously stated, comparing the fingerprint to every stored fingerprint to find the corresponding song would be extremely heavy.

What we do first is split our fingerprint into l subfingerprints. Then we are going to find a matching subfingerprint into every column and increment a counter for every song id.

At the end of this process we have a counter associated with every song. The songs whose counter is greater that a vote threshold v will be kept for a full comparison.

The candidate fingerprint is going to be compared using a Hamming Distance to the full signature of those kept fingerprints.

3. Implementation

3.1 Use of MapReduce for Spectrogram Generation

The spectrogram generation can be a tough and long process, that's why we have decided to compute it with a Map Reduce [13]. This choice was mainly done for an educationnal purpose.

3.1.1 Map

The mapping is related to the shift of the frame : each shift is a mapper. The mapping part express the idea that the frame is moving into time.

3.1.2 Reduce

The reducing part compute the FFT for each key and write the results into the final file.

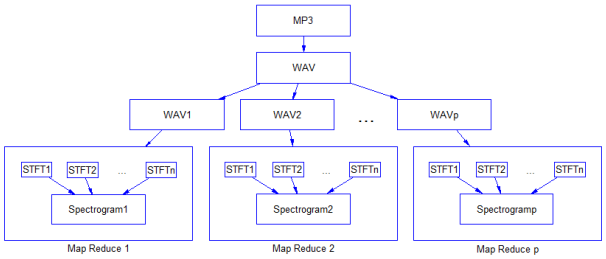


Figure 8. Spectrogram creation using MapReduce

The splitting part of the WAV file is done with thread and each Map Reduce is also encapsulated into a thread.

3.2 MapReduce for the Haar

The Haar Wavelet algorithm implies a lot of iteration on the row of a matrix and so can be parallelized with a Map Reduce process. The Haar Wavelet, compute the spectrogram and filter the matrix to select the best results for the signature.

3.2.1 Map

The mapping is related to the iteration of the process on a row (or a column) : each iteration is a mapper.

3.2.2 Reduce

The reducing part compute the row decomposition for each key and write the results into the final file. At the end, it gives a the filtered matrix. The Haar Wavelet process not only implies some row computations but also some column computations, that's why two Map Reduce are done in this part.

Each process (both Map Reduce + both transpose + filter) are encapsulated into a thread.

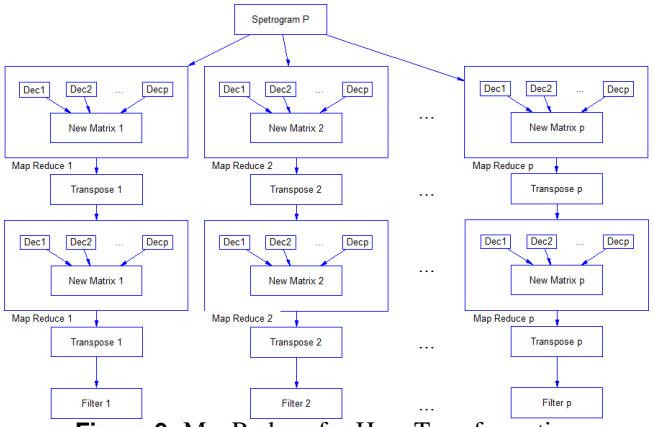


Figure 9. MapReduce for Haar Transformation

3.3 Cassandra database

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra is part of the NoSQL family, by comparison to relational or SQL database.

We apply Haar Wavelet transformation to **each spectrogram**, Once we get the Haar Wavelet transformation, we only keep the top-t wavelets.

3.3.1 Why Cassandra

First, Cassandra has a data model colum-oriented. We can add columns for a single or multiple rows on the fly. This is not the case in relational databases. This is a huge benefit in our case, if the fingerprint or its sub fingerprints overpass the model. Secondly, the query language, develop on top of Cassandra, is very similar to SQL. The CQL (Cassandra Query Language) has the same approach than Hive and drive easily data with simple requests. Cassandra is also highly scalable, and is very efficient inside a cluster. The project even has a implementation for Hadoop. It's a very popular database and is used in many large companies such as Twitter, Netflix, Facebook or Reddit.

3.3.2 Tables used

The project's database is simple. There are only 2 tables : one store musics characteristics (title, artist...) and the other store all the fingerprints for all the songs.

The first one is built as so :

IdMusic	Title	Artist	Length
Int	Text	Text	Int

Table 1. Musics table

The second one is built as so :

IdMusic	Fingerprint1	- 2	...	- p
Int	Blob	Blob	...	Blo

Table 2. fingerprints table

Because the fingerprint is stored as a byte array, we have to use a special type which is 'blob'. To communicate with the Cassandra database in Java [12][11], we are using some drivers from Datastax[10] company. It's convenient as they provide a dashboard for the database and some powerful Java class.

3.4 Implementation of the Haar decomposition

First, let's take a look at how to use the standard decomposition algorithm to compute the Haar Wavelet transformation. The input of the algorithm is a spectrogram, which is a two dimensional data structure. We process this data structure by applying the one dimensional Haar Wavelet transformation to each row. Once this is done, we apply the same transformation to the column of the data structure.

Below is an implementation of the algorithm in Java.

```
class HaarWavelet2D {

    double[][] spectrogram; // Input

    public void standardDecomposition(){ // Main process
        processRows(); // We apply the algorithm to the rows
        transpose(); // We transpose the spectrogram
        processRows(); // We apply the algorithm to the columns
        transpose(); // We transpose back the spectrogram
    }

    private void transpose(){
        ...
    }

    private void processRows(){
        for (int i = 0 ; i < spectrogram.length ; i++){
            spectrogram[i] = decompositionRow(spectrogram[i]);
        }
    }

    private double[] decompositionRow(double[] row){
        int steps = row.length;
        while (steps > 1){
            row = decompositionStep(steps, row);
            steps = steps / 2;
        }
        return row;
    }

    public double[] decompositionStep(int step, double[] row){
        double[] newRow = new double[row.length];
        System.arraycopy( row, 0, newRow, 0, row.length );

        for (int i=0 ; i < step/2 ; i++){
            newRow[i] = (row[2*i + 1] + row[2*i])/ 2;
            newRow[step/2 + i] = (row[(2*i)] - row[(2*i + 1)])/2;
        }

        return newRow;
    }
}
```

Second, we only keep the top-t wavelets : For this, we sort the wavelets by their highest number. We add a method to our previous class : `topTWavelets`.

To do so, we keep a sorted list of wavelet coefficients. This method allow to keep only a percentage of wavelets (2%, 5%, or even 10%). The determination of this threshold will be explained in the following section. This percentage determines a threshold under which most wavelets will be discarded.

```
    public void topTWavelets(double percentage){

        ArrayList<Double> ListValues = new ArrayList<Double>();

        for(int i = 0 ;i<matrix.length ; i++){
            for(int j = 0 ; j <matrix[0].length ; j++){
                ListValues.add(Math.abs(matrix[i][j]));
            }
        }

        Collections.sort(ListValues,Collections.reverseOrder());


        int ThresholdIndex = (int)(ListValues.size()*percentage);
        System.out.println("Threshold index :" + ThresholdIndex);
        double ThreSholdValue = ListValues.get(ThresholdIndex);

        for(int i = 0 ;i<matrix.length ; i++){
            for(int j = 0 ; j < matrix[0].length ; j++){
                matrix[i][j] = (Math.abs(matrix[i][j])<ThreSholdValue) ? 0 : matrix[i][j];
            }
        }
    }
```

This gives a us a very **sparse spectrogram** where only the top-x% of wavelets are kept. This is a representation of the overall shape of the spectogram.

3.4.1 Selection of the threshold

This is important to find an ideal threshold for the selection of the top-t wavelets coefficients. On the one hand, we don't want to keep too many wavelets as we want to provide a resistance to noise while comparing spectrograms. On the other hand, we want to keep enough wavelets so that different spectrograms have a very low similarity.

Below is an example of two spectograms. The left one is the original recording, while the right one is the song recorded from a smartphone microphone.

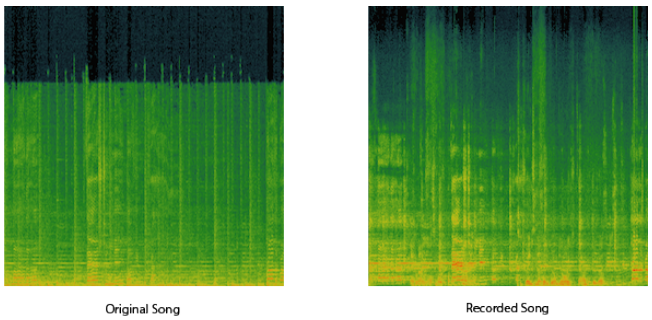
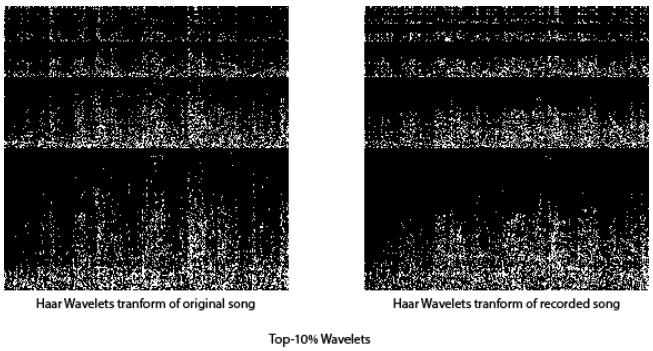


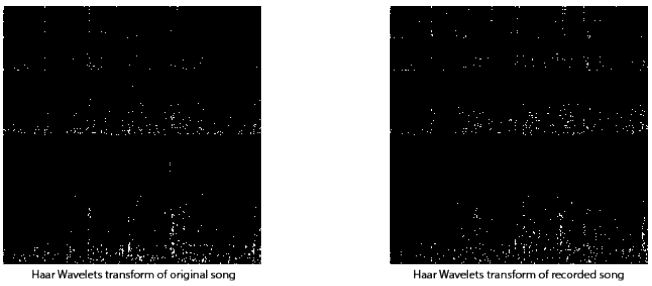
Figure 10. Spectrogram of the original song versus a recorded version

We apply the Haar Wavelet transform with a large threshold : keeping the top-10% wavelets.



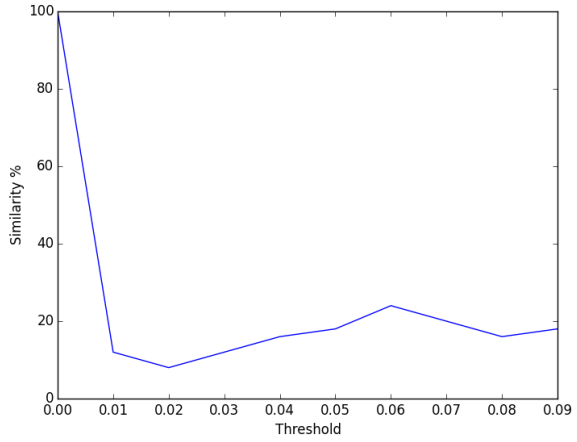
Applying a loose threshold makes the comparison of spectrogram more sensible to noise. Wavelets due to noise appear in the right spectrogram, curbing the similarity between the two spectograms.

This time, we are using a stricter threshold : we only keep the top-1% wavelets.



We also see that applying a too strict threshold will increase the similarity between differents spectrograms, as wavelets are getting rare.

This raises the issue of choosing the correct threshold for the wavelets. We have determined empirically the optimal threshold. We measured the similarity of different spectrograms with their corresponding spectrograms where noise was added. For each comparison, we tried different values of the threshold, choosing the one that maximized the similarity.



From our tests, the most significant value was to keep the **top-5% wavelets**.

4. Results and Discussion

4.1 Results

Ideas explained in the first section of this paper are implemented and show the relevance of the Waveprint algorithm to give an indexable representation of a song with fingerprints. Unfortunately, we encountered performance issues, especially with the Map Reduce parts because of the sheer amount of data IOs on the hard drive. We only tested our program in a local environment on a single computer. We did not have the opportunity to harness the power of a Hadoop cluster to run our algorithm. The algorithm and the whole process are performing well but w

Those results were expected as we knew beforehand that the hadoop system was not really fitted for this exercice. However, the fact that our implementation is not optimized yet add the insult to the injury. The prerequisites technical knowledge to implement the algorithm was also an obstacle to the project completion.

4.2 Future work

We now intend to go further in our work. First we are going to optimize the existing code so that it works faster. Then we plan on testing our solution on a real cluster. Finally, due to the streaming nature of the retrieval process, we would like to implement a Spark version of this process.

We were absolutely delighted to work on this project and will share our achievements to other students during a forthcoming meetup in Paris.

Acknowledgments

We would like to thank professor Mauro Sozio for letting us the opportunity to conduct this project.

We also thank Benoit PetitPas for his advices regarding the best choice of technology.

Finally, we also express our gratitude to the Telecom ParisTech Computer science department for the use of the network and hardware.

References

[1] Google waveprint algorithm : http://ecee.colorado.edu/~fmeyer/class/ecen5322/waveprint.pdf

[2] Cloud-based audiofingerprinting with mapreduce : http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6411864

[3] DFT : http://en.wikipedia.org/wiki/Discrete_Fourier_transform

[4] FFT : http://en.wikipedia.org/wiki/Fast_Fourier_transform

[5] Cooley-Tukey : http://en.wikipedia.org/wiki/CooleyE28093Tukey_FFT_algorithm

[6] STFT : http://en.wikipedia.org/wiki/Fast_Fourier_transform

[7] Spectrogram : http://en.wikipedia.org/wiki/Spectrogram

[8] FFT implementation : ttp://introcs.cs.princeton.edu/java/97data/FFT.java.html

[9] DFT implementation : http://jvalentino2.tripod.com/dft/

[10] FFT : http://en.wikipedia.org/wiki/Fast_Fourier_transform

[11] Cassandra Keyspace : http://planetcassandra.org/create-a-keyspace-and-table/

[12] Starting with Cassandra and Java : http://planetcassandra.org/getting-started-with-apache-cassandra-and-java/

[13] Hadoop, *The Definitive Guide 3rd Edition* : Tom White, O'Reilly edition.

[14] Haar wavelet explanation : http://www.whydomath.org/node/wavlets/hwt.html

[15] Audio recognition with python : http://willdrevo.com/fingerprinting-and-audio-recognition-with-python/

[16] Stanford minHash explanation : http://web.stanford.edu/class/archive/cs/cs276a/cs276a.1032/handouts/minhash-6in1.pdf