AN EXTENDED AUDIO-FINGERPRINT METHOD WITH CAPABILITIES FOR SIMILAR MUSIC DETECTION

Sébastien Fenet, Yves Grenier, Gaël Richard
Institut Mines-Télécom ; Télécom ParisTech ; CNRS LTCI
37 rue Dareau, 75014 Paris, France
Firstname.Lastname@telecom-paristech.fr

ABSTRACT

Content-based Audio Identification consists of retrieving the meta-data (i.e. title, artist, album) associated with an unknown audio excerpt. Audio fingerprint techniques are amongst the most efficient for this goal: following the extraction of a fingerprint from the unknown signal, the closest fingerprint in a reference database is sought in order to perform the identification. While being able to manage large scale databases, the recent developments in fingerprint methods have mostly focused on the improvement of robustness to post-processing distortions (equalization, amplitude compression, pitch-shifting,...). In this work, we describe a novel fingerprint model that is robust not only to the classical set of distortions handled by most methods but also to the variations that occur when a title is re-recorded (live vs studio version in particular). As a result our fingerprint method is able to identify any signal that is an excerpt of one of the references from the database or that is similar to one of the references. The issue that we cover thus lies at the intersection of audio fingerprint and cover song detection, meaning that the functional perimeter of our method is substantially larger than the classical audio fingerprint approaches.

1. INTRODUCTION

Audio identification consists of retrieving some meta-data associated with an unknown audio excerpt. The typical use-case is the music identification service which is nowadays available on numerous mobile phones. The user captures an audio excerpt with his mobile phone microphone and the service returns meta-data such as the title of the song, the artist, the album... Other applications include automatic copyright control, automatic segmentation and annotation of multimedia streams, jingle detection, ... (see [1] for more details).

Audio fingerprint is the most common strategy for performing audio identification when no meta-data has been embedded in the unknown audio excerpt. It consists of extracting from each audio reference of a given database a compact representation (the fingerprint) which is then stored in a database. When identifying an unknown excerpt, its fingerprint is first calculated and then compared with all fingerprints stored in the database. This general scheme (see Figure 1) has been explored by numerous authors (for example, see [2–5]). The state of the art thus comprises a wide variety of models for the "Fingerprint" block associated with their search strategies ("Matching" block). The authors of recognized works have put their efforts on two major points: first, the captured signal should be recognized even if it has undergone a series of distortions; second, the algorithm has to manage a database containing huge amounts of audio references. This robustness to distortions usually refers to the robustness to different post-processings of the signal. Indeed, the successful fingerprint technologies can achieve identification of a signal that has been cropped, re-equalized, amplitude-compressed, pitch-shifted and/or time-stretched and possibly recorded in a noisy environment with a bad microphone. Clearly, traditional fingerprint methods do not handle the connection that exists between two pieces of music that are similar. This means that if the system has learned one song and is required to perform an identification of a cover version, it will not be able to indicate the correspondence. This includes the case of an artist performing 'live' one of his titles that is stored in its studio version in the database. At a more dramatic level, the same song, with the same orchestration, that is re-recorded in the same studio conditions will not be considered as a match of the first recording. This comes from the fact that traditional fingerprint approaches use low level features to characterize the

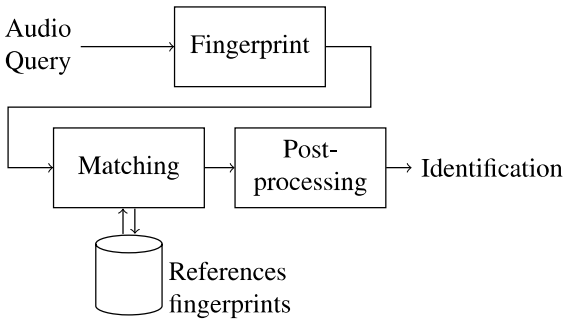

Figure 1. Architecture of a fingerprint system

THIS WORK WAS ACHIEVED AS PART OF THE QUAERO PROGRAMME, FUNDED BY OSEO, FRENCH STATE AGENCY FOR INNOVATION.

signal. These features, which bear little musical meaning, are not preserved from one version of the song to the other.

Conversely, approaches that can match two very different versions of one song have been developed [6–8] in the field of cover recognition. The counterpart of these approaches is that they usually cannot handle a large reference database. The proposed methods are indeed CPU-demanding and since they do not include any kind of fast search mechanism, using them in an audio-identification use-case would require the exhaustive comparison of the unknown signal with every reference. Given the computation time of the method and the number of references in traditional use-cases, this would not be feasible.

We propose in this work an extended fingerprint algorithm, that is able to do the matching between two identical records with different post-processings (called near-exact matches) but that can also recognize a query that is only "similar" to a song of the database, while keeping the ability to manage large scale databases. The paper is organized as follows: Section 2 is dedicated to a detailed description of our method. Following the presentation of the general workflow, the signal model is introduced and it is shown how this model can integrate in an index scheme that allows the management of large-scale databases. We finally explain a scoring procedure that can be applied to a reduced set of candidates that are output after the index phase. In Section 3, we propose some experiments to evaluate the performance of the algorithm on an audio-fingerprint use-case that includes both challenges of identifying broadcast sections that are near-exact matches of references and others that are only similar to references. The paper ends with a synthesis of the work and a critical analysis of the results.

2. DESCRIPTION OF THE SYSTEM

2.1 General Workflow

In our system, any piece of audio signal is modeled by a sequence of states computed in the fashion of Section 2.2. Similarly to the vast majority of fingerprinting approaches (*e.g.* [3] [4]) the efficiency that is necessary to handle large scale databases is achieved thanks to the use of an index-based search. Our method thus includes a learning stage, during which features extracted from the audio references are used to build the index. However, the task of identifying matches that are "near-exact" as well as "similar" is too demanding to be performed by the index search on its own. We thus propose a "two-levels" architecture, such as shown in Figure 2. When analyzing an unknown query (modeled by its sequence of states), the system performs a first search in the index. The output of this step is a reduced set of M best candidates (typically $M = 10$). Thanks to the index, the candidates are efficiently selected among the numerous references. The counterpart is that this matching stays vague. The post-processing step is a finer comparison between the candidates and the query. In our implementation the comparison is achieved thanks to dynamic programming, which has a heavy CPU cost but it is here limited to a small number of candidates. This gives a matching score for each of the candidates with the query. A threshold strategy can finally be set up in order to decide if there is a match or not. The same kind of funnel-shaped architecture is used in [4].

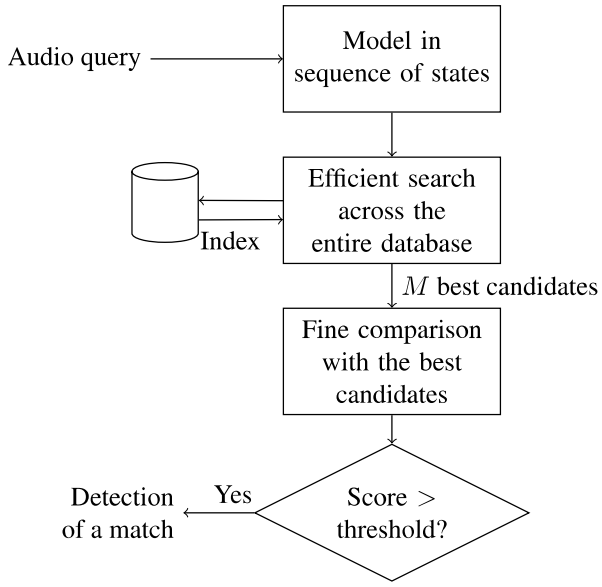

Figure 2. Workflow when identifying an audio query

2.2 Signal Model

The signal model that we suggest can be seen as an extension of the model proposed in [9] in a different context. The main idea to obtain the sequence of states is to sample the signal at instants that are "musically meaningful". These occur at dates $\{t_1, t_2, .., t_n\}$. To each of this date t_k we associate an information i_k which locally characterizes the signal. Finally, we define the *state* as the association of the date with the local piece of information:

$$s_k = (t_k, i_k)$$

In our implementation, the dates $\{t_1, ..., t_n\}$ correspond to the peaks of an onset detection function based on a sub-band decomposition (as suggested in [10]). In brief, the method starts from the magnitude of the spectrogram of the signal. The latter is temporally filtered in order to mask the rapid variations while keeping the sharp attacks. The filtered spectrogram is then log-compressed, which has the effect of remapping the amplitudes of the spectrogram on a scale that is closer to our perception. The Spectral Energy Flux is subsequently computed by estimating the temporal differential of the pre-processed spectrogram and then half-wave rectified in order to keep only positive values. Finally the information is integrated by summing the values of the Spectral Energy Flux across the frequency bands. This gives a global onset detection function that shows high values at instants with a meaningful amount of change in the spectrogram in one or several frequency bands. Onsets are then localized thanks to a peak-peaking strategy.

As for the local information i_k, it is composed of two features:

$$i_k = (c_{k,l}, c_{k,r})$$

where $c_{k,l}$ is the mean chroma vector at the left of t_k and $c_{k,r}$ is the mean chroma vector at the right of t_k. The use of chroma vectors is motivated by the fact that these features show little variations from one version of a song to the other [7].

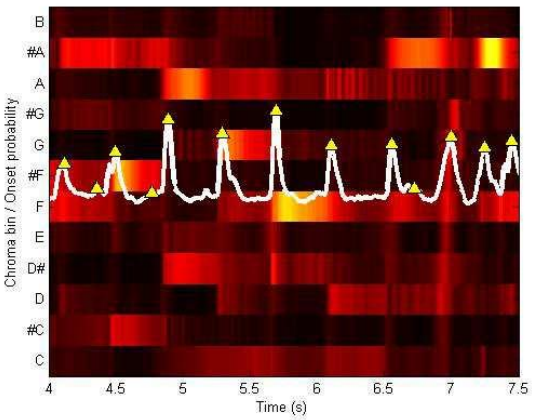

Figure 3. Illustration of the decomposition of the signal

Figure 3 shows the superposition of the graphical representations of an onset detection function (in white) and a chroma representation of the same signal. In our model, the dates $\{t_1, ..., t_n\}$ would be given by the dates of the yellow triangles on the onset function (obtained by peak-picking). For each date, the left mean chroma vector (mean chroma vector between the previous triangle and the current) and the right mean chroma vector (mean chroma vector between the current triangle and the next) would be computed, associated to the date and stored. The advantage of this model is that it is rather compact (which is an interesting feature for the search in large-scale databases) but still contains both rhythmical and harmonic information.

2.3 Indexing Scheme

An index is a function that takes a key in input and outputs a set of values.

$$h : k \longmapsto \{v_i\}$$

The interest of the operation is that it can be done in time $O(1)$. In our case, the values will be pointers to the references of the database and the keys will be audio features that characterize these references. The crucial point is to set up a distortion-invariant key, while still sufficiently discriminating. The invariance to the distortions should handle the classical post-processing distortions (equalization, pitch-shifting, ...) but also the recording of the same song in different conditions (matching of similar items).

For a given audio signal, we propose the following key-generation mechanism. For each state s_k one key is generated. The key is a binary version of the mean chroma vector at the right of t_k. Practically, for any bin b of the chroma vector, the binary chroma vector $\widetilde{c}_{k,r}$ is given by:

$$\widetilde{c}_{k,r}(b) = \begin{cases} 1 & \text{if } c_{k,r}(b) > \overline{c_{k,r}} \\ 0 & \text{otherwise} \end{cases}$$

with $\overline{c_{k,r}}$ the mean value of vector $c_{k,r}$:

$$\overline{c_{k,r}} = \frac{1}{12} \sum_{b=1}^{12} c_{k,r}(b)$$

We can note that at this stage of the process, the mean chroma vector at the left is unused. It will however be involved in the fine comparison step.

The learning phase is similar to [3]. The keys of all the references are extracted. They are then stored in the index. When storing one key, we define its associated value as the identifier of the reference containing this key together with the time of occurrence t_k of the key in the reference. Let us note that if one key k occurs several times, $h(k)$ will consist of all the different values stored with this key.

In the analysis phase, the keys of the query are extracted. For one key k occurring at time $t_q(k)$ in the query we can efficiently retrieve, thanks to the index function, all its occurrences in the references. If k occurs at times $t_r^1(k)$, ..., $t_r^i(k)$ in the reference r, we store the couples $(t_r^1(k), t_q(k))$, ..., $(t_r^i(k), t_q(k))$ in a scatter plot (one scatter plot per reference).

Let us now consider that the query is an excerpt of the reference r starting at time t_0. Let us also consider that the query is time-stretched by a factor κ (either because of some specific post-processing or because the query is another record of the same song with a slightly different tempo). Then, the key k extracted from the query at time $t_q(k)$ should be retrieved in the reference r at time $\kappa(t_q(k) - t_0)$. This means that, in the scatter plot of reference r, all these corresponding keys produce dots that are located on the straight line:

$$Y = \frac{1}{\kappa} X + t_0$$

Though, the scatter plot also contains a meaningful number of dots that are outside this line. These correspond to keys that are found in the query and that occur several times in the reference. We must indeed keep in mind that the keys we defined are not very discriminative. Figure 4 shows an example of such a scatter plot.

The last step consists of finding, for each scatter plot, the 'best' straight line. Intuitively enough, the 'best' line is the one comprising the highest number of dots. This is done thanks to the Hough transform [11]. In brief, it is an efficient counting technique that relies on the set up of an accumulator, which references all possible lines. Each dot of the scatter plot is iteratively tested in the following way: all lines that go through that dot have their values increased in the accumulator. At the end of the process, the line with the highest value in the accumulator is the best one.

In the end, we have, for each reference, the parameters of the best line (κ and t_0) and the number of dots (*i.e.* the number of keys) that match this line. The M references with the highest number of matching keys are considered

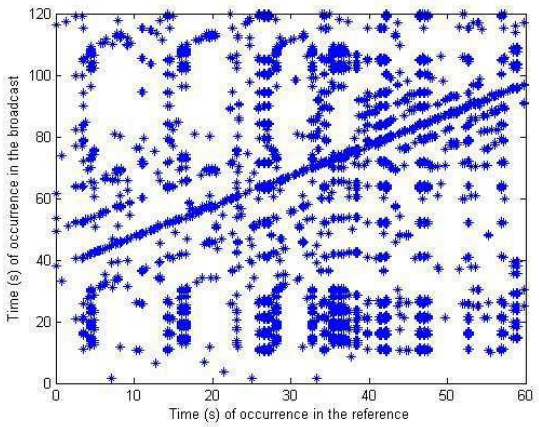

Figure 4. Scatter plot of reference r

as the M best matches to the query. They are output as candidates.

2.4 Final Scoring of the Candidates

At this stage of the process, the system has to compare the reduced set of M best candidates with the query. We can consequently afford a CPU-consuming comparison. The state of the art in similarity offers a wide variety of algorithms. In this work, we suggest a dynamic programming approach that relies on the modeling of the audio signal that we introduced in section 2.2. This can be seen as an extension of the work in [9].

First, the query is cropped and rescaled according to the parameters κ and t_0 that have been output in the search-by-index phase. The corrected query is represented by the sequence of states $\{s_1...s_m\}$. As for the candidate reference, we use the notation $\{s'_1...s'_n\}$.

When dynamically aligning two sequences of states, one has to define three types of penalties.

- $f^d(s_i)$: penalty assigned to the deletion of state s_i
- $f^i(s_i)$: penalty assigned to the insertion of state s_i
- $f^s(s_i, s'_j)$: penalty assigned to the substitution of state s_i by state s'_j.

In our implementation f^d and f^i are both taken constant and equal to 0.3. This value has been experimentally determined. Knowing that $s_i = (t_i, (c_{i,l}, c_{i,r}))$ and $s'_j = (t'_j, (c'_{j,l}, c'_{j,r}))$, we define f^s by:

$$f^s(s_i, s'_j) = \cos(c_{i,l}, c'_{j,l}).\cos(c_{i,r}, c'_{j,r}).\mathrm{e}^{\frac{|t_i - t'_j|}{c}}$$

The first cosine term penalizes the resemblance of the mean chroma vector at the left of s_i with the one at the left of s'_j. The second cosine term does the same for the mean chroma vector at the right. The exponential term penalizes the timing error between the occurrence of state s_i and the occurrence of state s'_j.

Dynamic programming consists of iteratively filling a scoring matrix D. For any $(i, j) \in \{1..m\} \times \{1...n\}$, $D(i, j)$ contains the score of the alignment of the subsequence $\{s_1...s_i\}$ with the subsequence $\{s'_1...s'_j\}$. D is computed in the following manner:

$$D(i,j) = \max \left\{ \begin{array}{l} D(i, j-1).f^d(s'_j) \\ D(i-1, j-1).f^s(s_i, s'_j) \\ D(i-1, j).f^i(s_i) \end{array} \right\}$$

The score of the alignment of $\{s_1...s_m\}$ with $\{s'_1...s'_n\}$ is finally given by $D(n, m)$ (Figure 5 shows an example of matrix D). The candidate reference is considered as an actual match to the query if the score is greater than a predetermined threshold.

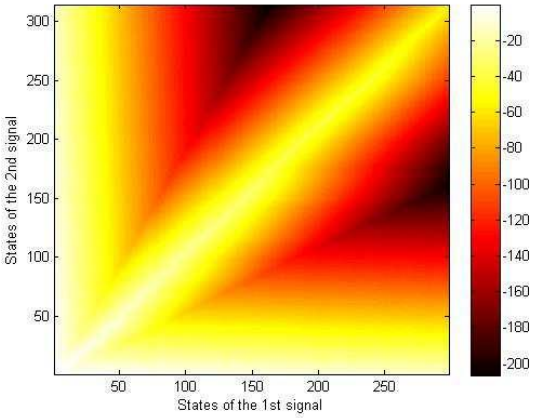

Figure 5. Scoring matrix of a dynamic alignment

3. EXPERIMENTS

3.1 Experimental Framework

Our experiments follow the evaluation framework described in [12]. This framework was designed for the evaluation of audio-fingerprint systems on the basis of real broadcast streams. This constitutes a quite challenging evaluation, even if one restricts the task to the detection of near-exact matches, because the broadcast stations apply a wide variety of different sound-processing to the music, thus resulting in a heavy level of post-processing distortion to be handled by the algorithm. The evaluated system learns a database of reference music titles before the actual evaluation. Each music title broadcast in the stream that corresponds to one of the references has been manually annotated in a global annotation file. The annotation contains a unique identifier of the broadcast title, its start and end times in the broadcast. When the algorithm is run on the stream, it has to detect all the occurrences of the broadcast references. One detection output by the algorithm contains the identifier of the detected reference and its detection time in the stream. If one output detection contains the identifier r_i and has a detection time that lies between the annotated start time and end time of one occurrence of the same title r_i, we make this broadcast a *detected title* (DT). Let us note that multiple detections of the same occurrence

of one given title are counted only once. Conversely, if the algorithm detects a reference during an empty slot (denoted by n) or during a slot that contains another music title, we count one false alarm (FA). There is no kind of integration performed on the false alarm counting: each output corresponding to a false alarm is added up. Figure 6 illustrates the task of the algorithm as well as the scoring procedure. The detections output by the algorithm are figured by the arrows. The way we integrate these detections when scoring is illustrated by the overbraces.

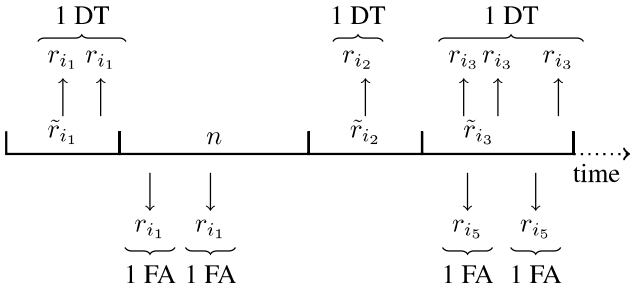

Figure 6. Illustration of the evaluation framework

It is noticeable that the main objective of this framework is to provide an evaluation methodology that fits a realistic use-case with real-world data. The counterpart of this strategy is that we have a limited control over the corpus, which prevents us from displaying metrics such as the level of distortion in the broadcast titles or the detection ability of the algorithm at a scale that is shorter than the duration of a song.

3.2 Corpus

It is quite a major issue to bring together a reference database of real music titles with the corresponding annotations (with a reasonably low number of wrong annotations). In our context, we could assemble an annotated database of 2400 pop-rock music titles. Let us note that such a database still constitutes a realistic reference set for many applications. It is for instance noticeable that the assembled database covers all the music broadcasts of two different radio channels during two weeks. Besides, such a quantity of references is large enough to get a fair idea of the system's capabilities.

As far as the analyzed stream is concerned, we worked with 24h of the French radio RTL. These contain one program that essentially features live performances of contemporary titles. The latter are not explicitly present in the database but the corresponding studio versions are. In total, the stream contains 107 annotated music titles from the database, 99 of which are near-exact matches whereas the 8 remaining are live versions of the corresponding references.

Given the framing that we apply to the stream, whose hop is set to 15 seconds, the algorithm is queried $\frac{24.3600}{15} = 5760$ times. Since the queries are extracted from a real broadcast, they consist of distorted versions of the references, live versions of the references or empty slots (speech, ads or musics that are not in the database). In terms of false alarms, this means that the algorithm has the possibility to output around $5760 \times 2400 = 13,824,000$ of them. These figures confirm the relevance of the experiment.

3.3 Results

Figure 7 shows, in red, the results of the algorithm on a classical ROC diagram. As most detection systems, our algorithm's output relies on a set of candidate detections with given scores. The candidate references are originally selected in the search-by-index step and they are finally scored thanks to a dynamic alignment. In order to complete the process, one can simply set a threshold on the score: only the candidates with a score above the threshold are output by the algorithm. In such a configuration, one can evaluate the algorithm's output with different threshold values. Each point of the ROC curve thus corresponds to the results obtained by the algorithm with one specific threshold value. The X-coordinate corresponds to the number of false alarms and the Y-coordinate to the proportion of *detected titles*. Such a curve allows to observe the overall response of the algorithm and to compare different methods independently of the final post-processing. For comparison, the ROC curve of a 'traditional' audio-fingerprint method [13] is plotted in black. This method has proved to be at the level of the state of the art, notably when working with very large databases ($> 30,000$ references).

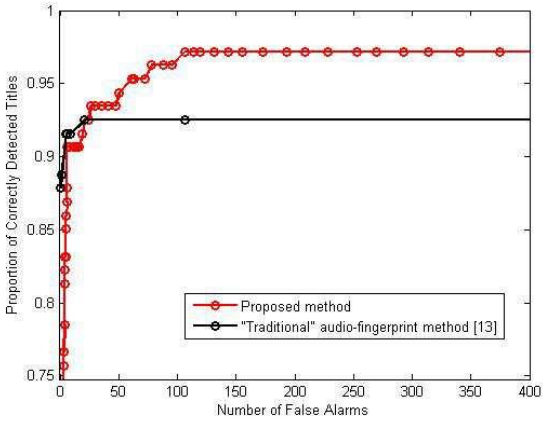

Figure 7. Results of the algorithm (ROC curve)

The results show that the method from [13] is indeed very efficient at detecting near-exact matches: the ROC curve is almost ideal if we do not consider the similar items. The approach can indeed reach the highest number of correct detections (*i.e.* the 92.5% of the corpus that are near-exact matches) while virtually outputting no false alarm. As far as our novel method is concerned, the outputs of the algorithm show that it is also able to detect near-exact matches with a low number of false alarms, albeit a little higher than the traditional system. But most importantly, it is able to overstep the borders of exact matching. We

can indeed see that some of the points of the ROC curve have a higher Y-coordinate than the proportion of near-exact matches in the corpus. This means that the algorithm is able to successfully detect some of the broadcasts that are live versions of music titles that are stored in their studio version in the reference database! Interestingly, the annotated broadcast titles that are never reached by the ROC curve correspond to live versions that have been transposed compared to the studio version in the reference database. Of course the more we go toward similarity, the higher number of false alarms we generate. This sounds logical since by loosening the matching requirements in order to detect the live versions of some references, we also favor the detections of distinct items that are musically similar to some of the references.

In terms of run time, it takes our current Matlab implementation of the method 6 seconds to perform the index search (across the entire database) for a 120s-long query. We are thus confident that the algorithm can scale up to much larger reference databases while keeping reasonable computation times. For the sake of precision, the code was run on an Intel Core 2 Duo @ 3,16 GHz with 6MB of Cache and 8GB of RAM. We do not give details about the running time of the fine comparison step. The latter is indeed meant to process a fixed number (M) of best candidates, whatever the size of the database. Its running time thus has no impact on the scalibility of the algorithm.

4. CONCLUSION

In this work we have presented a novel fingerprint model associated with a tailored innovative search strategy. As a member of the "audio-fingerprint algorithms" family, the resulting algorithm has the ability of managing large reference databases. Besides it has the very interesting capability of indifferently managing queries that are near-exact matches of some reference of the database or queries that are only similar to one reference. This result is achieved thanks to a careful modeling that preserves the musical significance of the signal at every calculation step (computation of the model in state, extraction of binary keys, aggregation of the index outputs).

The algorithm has been tested on a well-tried evaluation framework for the detection of near-exact matches. We have selected a specific corpus that includes both near-exact and similar matches. The conclusion of this experiment is that the algorithm could detect with a performance close to the state of the art the items that were near-exact matches and it could also detect some of the similar items. These results are thus very encouraging.

Our perspectives include a more extensive testing of the algorithm. The conducted experiment indeed involves a middle-size database. We need to go towards bigger corpora to check the scalability of the approach. We also target the further extension of the robustness of the model. The described experiment has indeed emphasized the lack of robustness of the current model to transposition. Besides the search for straight lines in the scatter plots makes it clear that the model is not robust to a change of structure. Trying to handle these while keeping the indexing capability of the system are challenges of high interest.

5. REFERENCES

[1] P. Cano, E. Batlle, E. Gomez, L. de C.T. Gomes, and M. Bonnet, "Audio Fingerprinting: Concepts and Applications," in *International Conference on Fuzzy Systems and Knowledge Discovery*, (Singapore), Nov. 2002.

[2] M. Ramona and G. Peeters, "Audio identification based on spectral modeling of bark-bands energy and synchronization through onset detection.," in *ICASSP*, (Prague, Czech Republic), May 2011.

[3] A. Wang, "An Industrial-strength Audio Search Algorithm," in *ISMIR*, (Baltimore, Maryland, USA), Oct. 2003.

[4] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *CBMI*, (Brescia, Italy), Sept. 2001.

[5] C. Cotton and D. Ellis, "Audio Fingerprinting to Identify Multiple Videos of an Event," in *ICASSP*, (Dallas, Texas, USA), Mar. 2010.

[6] D. P. W. Ellis and G. E. Poliner, "Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking," in *ICASSP*, (Honolulu, Hawaii, USA), Apr. 2007.

[7] J. Serrà, E. Gómez, P. Herrera, and X. Serra, "Chroma binary similarity and local alignment applied to cover song identification," *IEEE Transactions on Audio, Speech and Language Processing*, Aug. 2008.

[8] M. Müller, F. Kurth, and M. Clausen, "Audio matching via chroma-based statistical features," in *ISMIR*, (London, UK), Sept. 2005.

[9] O. Gillet and G. Richard, "Drum loops retrieval from spoken queries," *Journal of Intelligent Information Systems*, vol. 24, pp. 159–177, May 2005.

[10] M. Alonso, G. Richard, and B. David, "Extracting note onsets from musical recordings," in *ICME*, (Amsterdam, NL), July 2005.

[11] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, pp. 11–15, Jan. 1972.

[12] M. Ramona, S. Fenet, R. Blouet, H. Bredin, T. Fillon, and G. Peeters, "Audio Fingerprinting: a Public Evaluation Framework Based on a Broadcast Scenario," *Applied Artificial Intelligence: An International Journal*, Feb. 2012.

[13] S. Fenet, G. Richard, and Y. Grenier, "A Scalable Audio Fingerprint Method with Robustness to Pitch-Shifting," in *ISMIR*, (Miami, Florida, USA), Oct. 2011.