A FRAMEWORK FOR FINGERPRINT-BASED DETECTION OF REPEATING OBJECTS IN MULTIMEDIA STREAMS

Sébastien Fenet[1], *Manuel Moussallam*[1], *Yves Grenier*[1], *Gaël Richard*[1] *and Laurent Daudet*[2]

[1]Institut Mines-Telecom - Telecom ParisTech
CNRS/LTCI - UMR 5141

[2]Institut Langevin - ESPCI ParisTech
Paris Diderot Univ. - UMR 7587

ABSTRACT

We present an original framework for the detection of repeating objects in multimedia streams. This framework is designed so that it can work with any fingerprint model. A fingerprint is extracted for each incoming frame of the multimedia stream. The framework then manages this fingerprint so that if one similar frame comes later in the stream, it will be identified as a repetition. The framework has been tested with two distinct fingerprint models on simulated and 'real-world' data. The results show that the framework performs well with both presented models and that it is suitable for industrial use-cases.

Index Terms— Fingerprint, repeating objects, indexing, framework

1. INTRODUCTION

Multimedia streams often contain repetitive data (see [1]). Depending on the considered medium, repeated objects can be entire programs, songs, advertisements or jingles. Let us note that the repeated objects might however be distorted from one version to the other (different volume, different equalization, addition of noise, ...). For numerous reasons, it is interesting to automatically detect these repetitions. Applications include compression, automatic annotation of repeating objects, segmentation and data mining.

As some authors have pointed out [2], it is relevant to use the notion of fingerprint for fast detection of a repetition. A fingerprint in signal processing can be viewed as a compact representation of a signal excerpt. This representation is designed so that two similar signals should have the same fingerprint. When it is queried with an unknown signal, the challenge for a fingerprinting system is to perform an efficient search. Indeed, it has to compare the unknown fingerprint with a reference database that usually contains thousands of them. Hence, the powerful summary and indexing capabilities that are required for a fingerprint system suit well the repeating objects detection problem.

Although methods have been proposed [2, 1, 3], each comes as a specific combination of a detection method and a fingerprint. In this work, we present a general framework that allows the use of any fingerprint extraction system for the detection of repeating objects. The only requirement is that the fingerprint extraction system outputs a set of time-localized keys for any frame of signal. This is the case for most fingerprint systems. They usually use this set of keys in an indexing scheme for fast detection. For instance, Wang's fingerprint system [4] (also known as *Shazam's system*) extracts time-localized pairs of peaks in the spectrogram and uses them as database keys. Haitsma's [5] (known as *Philip's system*) extracts one binary feature every 10ms that summarizes the spectral content. The authors call these features 'hashes' and use them as keys in a 'look-up table'. In spite of the general applicability of the framework, we have restricted our experiments to audio use-cases. However, one should note that indexing tasks on multimedia signals (e.g. video) can often be achieved by working on the sole audio component [6].

In the first section, we describe the framework and all its components. In the second section, we provide two examples of fingerprints that have been integrated in the framework. Finally, we test these configurations in two distinct experiments. The first experiment is meant to accurately evaluate the performance of the systems. The second one is based on a 'real-world' use-case.

2. DESCRIPTION OF THE FRAMEWORK

2.1. General architecture

We describe here the general flow of the framework, given in Figure 1. The stream is framed and then linearly processed. Each frame undergoes a fingerprint extraction. From here, the system forks. One branch is dedicated to analyzing the fingerprint (in practical terms, looking for matches in the past), the other is dedicated to storing the fingerprint in the database containing the past fingerprints. In the analysis branch, the fingerprint is matched against the database containing the previous frames' fingerprints. Based on this matching result combined with the matching results of previ-

This work was achieved as part of the Quaero Programme, funded by OSEO, French State agency for innovation.

LD is on a joint position between Univ. Paris Diderot and Institut Universitaire de France

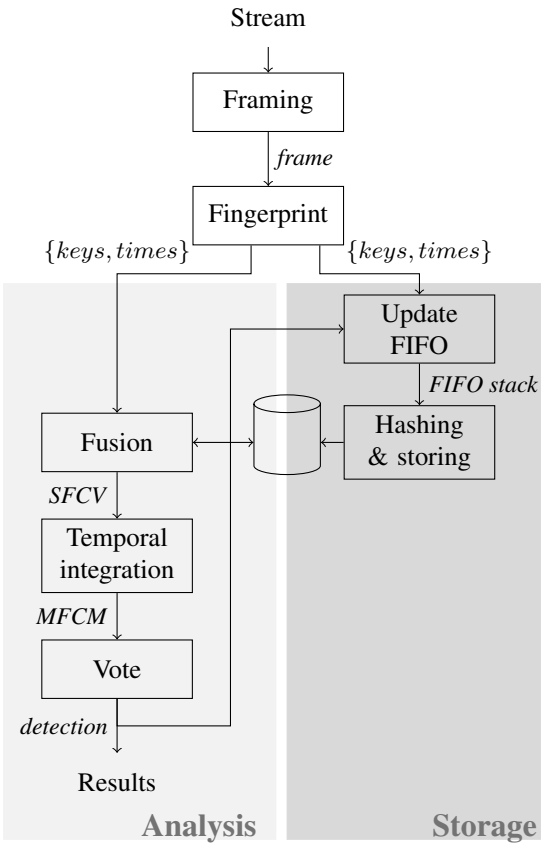

Fig. 1. Framework architecture

ous frames, a repetition detection decision is taken. In case of a detected repetition, the 'storage branch' is updated so that it will not store repeated frames in the database. In the storage branch, the frame's fingerprint is pushed into a FIFO ('First In First Out') buffer. This buffer delays subsequent storage processing for this frame. As the current frame fingerprint enters the buffer, the last fingerprint of the FIFO buffer is pushed into the database. Though, this latter will be written only if it has not been detected as part of a repeating segment.

The different building blocks of this general framework are further described below.

2.2. Framing and fingerprinting

The input stream is sliced in non-overlapping *analysis frames* f_n ($n \in \mathbf{N}$) of length L_a. A typical value for L_a is 5s. For a given analysis frame, the fingerprint module outputs a set of features along with their dates of occurrence. We call these extracted features *keys*. Formally, we define $\mathcal{K}$ the set of keys extracted in f_n. Let $O_k(f_n)$ be the number of occurrences of the key k in f_n. We then define $t_k(f_n) = \{t_k^i(f_n)\}_{i=1..O_k(f_n)}$ the set of times of occurrence of the key k in the frame f_n. The output of the fingerprint module is $\{(k, t_k(f_n))\}_{\forall k \in \mathcal{K}}$.

The database contains all the keys that have been extracted in the past stream with their times of occurrence in the stream. As it is meant to represent the past of the stream, we use the notation f_{-1} to refer to the database. Consequently, a key k appears $O_k(f_{-1})$ times in the database at times of occurrence $\{t_k^i(f_{-1})\}_{i=1..O_k(f_{-1})}$. When querying the database with key k, we get in output $\{t_k^i(f_{-1})\}_{i=1..O_k(f_{-1})}$. Our implementation uses the database engine "Berkeley DB" set to its "Hash Table" mode.

2.3. Analysis

The analysis starts with the fusion step. It aims at finding the closest match to the current analysis frame in the stream. The main idea is that if the current frame is the repetition of a previous section of the stream, its keys should all be stored in the database. Furthermore, all the keys extracted from the analysis frame should be retrieved in the past with the same delay. We then adopt the following methodology to find the best candidate in the past.

We compute the set of differences

$$\left\{\{t_k^i(f_n) - t_k^j(f_{-1})\}_{\forall (i,j) \in [\![1;O_k(f_n)]\!] \times [\![1;O_k(f_{-1})]\!]}\right\}_{\forall k \in \mathcal{K}}$$

We store these time differences in a histogram. The highest peak in the histogram gives the best candidate delay for a repetition.

Let us note that this methodology ensures the retrieval of the best candidate. Though, it does not require that all keys are preserved from one version to the other. It only requires that the majority of the keys are preserved. This makes the method robust to distortions that would corrupt part of the keys between the two versions. In order to bring even more robustness to the system, we output the M best candidate delays.

The work presented in [7] underlines the fact that the performance of a fingerprint system is much higher when issuing detections based on a vote mechanism involving several successive matching results rather than making a frame-by-frame decision. This idea is also exploited in the work from [8]. Our framework thus stores the matching results of several successive analysis frames before making a detection decision. The fusion step outputs a vector of M best candidates (that we call *Single Frame Candidates Vector* - SFCV) that is integrated in a $H \times M$ matrix that contains the H (for horizon) last SFCVs. We call this matrix the *Multiple Frames Candidates Matrix* - MFCM.

The last step of the analysis is a majority vote on the MFCM. Previous works suggest the following vote mechanism. For a given detection threshold θ, let M_n be the MFCM after integrating the matching results of frame f_n. Let $\mathcal{C}$ be the set of candidate delays that appear in M_n and h be a function that counts the number of occurrences of a candidate in the MFCM. Let δ be a function that is defined by:

$$\delta(x,y) = \begin{cases} x & \text{if } y > \theta \\ \emptyset & \text{otherwise} \end{cases}$$

The vote function is then defined by:

$$v(M_n) = \delta(\underset{c\in\mathcal{C}}{\text{argmax}}\{h(c)\}, \max_{c\in\mathcal{C}}\{h(c)\})$$

However, this vote model may generate some instability when dealing with objects that contain an inherent repetitive structure. For example, let us imagine that the database contains one song with two similar choruses. As a consequence, when processing an analysis frame belonging to a chorus of the same song, the two choruses in the database will be candidates in the MFCM. Besides, their number of occurrences in the MFCM will be very close. The result is that, when processing the successive analysis frames of a chorus, the detections issued by the vote algorithm may look like that: `chorus1-chorus2-chorus1-chorus1-chorus2...`

This, of course, is not desirable, since we would like our algorithm to consider that the successive analysis frames all belong to the same chorus. Ideally, we would like the algorithm to detect `chorus1` when processing the first analysis frames containing a chorus and `chorus2` later on in the stream.

In order to achieve this goal, we modify the preceding vote model so that it becomes auto-regressive. The autoregressive aspect is obtained by favoring the delay that best corresponds to the preceding vote result. This ensures a certain continuity in the algorithm detections. So, if the start of the song has been detected, and when reaching the chorus, the algorithm will naturally tend to select the first chorus of the song in the database. Formally it consists of replacing function h in the vote by $\widetilde{h}$, which is defined by:

$$\widetilde{h}(c) = \begin{cases} h(c) + \beta & \text{if } v(M_{n-1}) = c \\ h(c) & \text{otherwise} \end{cases}$$

In our implementation, $\beta = 1$.

2.4. Storage

One of the principles of the framework is to store the fingerprints of the analysis frames that have been processed in a database. However, we do not wish to store in this database the fingerprints of the frames that are detected as repetitions. There are two reasons for that. First, it would be a waste of space. Second, when matching a third fingerprint that would be alike, we would obtain two good candidates instead of one. That would uselessly jam the matching process.

As we have seen, the algorithm requires the matching results of H analysis frames before being able to make a decision. This is why we store in a temporary FIFO buffer the fingerprints of the processed analysis frames. This buffer contains $B > H$ processed fingerprints. If further processing outputs a repetition detection for frame f_n, its fingerprint in the FIFO buffer is labeled so that it will not be written in the database.

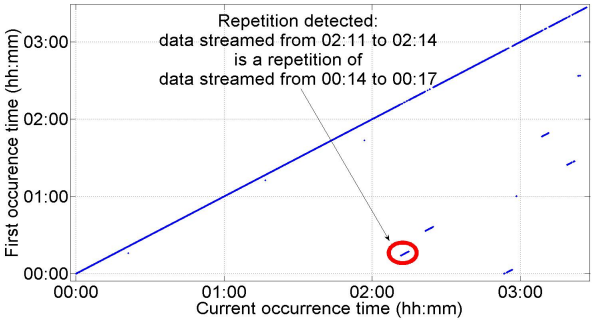

Fig. 2. Example of output visualization.

The FIFO buffer also has a screening function. Indeed, the repetitions that occur before B frames are not detected (since the corresponding fingerprints have not been added to the database). Depending on the use case, this can be useful to prevent over-segmentation. For instance, when segmenting a radio broadcast, one would usually want repeating segments that correspond to whole songs. Though, if there is no screening and if the songs contain repetitive choruses, the algorithm might annotate the songs in several repeating bits (the choruses) and unrepeated bits (the verses). By setting B to a larger value than the length of the song, we can prevent the system from detecting repetitions within the song.

2.5. Results

The framework outputs a decision for each analysis frame. It is either considered as a repetition of a previous frame, or as a first occurrence. A graphical illustration of the result is given in Figure 2. The origin point is the experiment starting date. Points on the diagonal indicate frames detected as first occurrences. Points outside of the diagonal indicate repeated frames. They are plotted with respect to their first occurrence dates.

3. DESCRIPTION OF THE FINGERPRINTS

3.1. A CQT-based fingerprint

The first fingerprint we use is fully described in [7]. In short, the methodology consists of using a 2-dimensional peak-peaking in the *Constant-Q-Transform* (CQT) spectrogram of each analysis frame. The extracted peaks are then grouped in pairs. Each pair is encoded in a form that makes it robust to common audio-distortions. These encoded pairs are used as *keys*. Their time localizations are given by the times of occurrence of the first peaks in the pairs.

3.2. A sparse decomposition-based fingerprint

The fingerprint presented above is based on a peak picking mechanism in the time frequency domain. Alternatively one can build a fingerprint based on a sparse decomposition of the signal in a redundant dictionary. Let x be a framed signal $\in \mathbb{C}^N$ and Φ be a dictionary of elementary waveforms $\phi_k \in \mathbb{C}^N$ called *atoms*. We denote $\hat{x}_m$ an m-term approximant of x in Φ, that is to say a linear combination of m waveforms: $\hat{x}^m = \sum_{i=0}^{m-1} \alpha_i \phi_i$. There are many different ways of building such approximant. A fast one is to iteratively select the ϕ_i according to an energetic criterion:

$$\phi_i = \underset{\phi \in \Phi}{\operatorname{argmax}} |\langle x - \hat{x}^{i-1}, \phi \rangle|$$

Algorithms based on this greedy paradigm are called Matching Pursuits (MP) following the work of Mallat et al [9].

Sparse decompositions have initially been proposed for compression purposes. Indeed, in a variety of multimedia contexts, wavelet dictionaries (e.g. for images) and Fourier-based transforms (e.g. MDCT for audio) have enabled a fair amount of dimensionality reduction. The idea of exploiting sparse decompositions for fingerprinting has already been proposed (e.g. in [10]).

An m-term approximant $\hat{x}^m$ can efficiently be used as a fingerprint if: 1) its dimension is much lower than that of x 2) two different signals would yield significantly different fingerprints and 3) the fingerprints exhibit some robustness to mild distortions. Most MP-like algorithms are only tailored for the first of these properties. However, in a fingerprint context, we are not interested in minimizing a reconstruction error, but in maximizing a discriminating power. Therefore, two options can be considered: either build a fingerprint from an existing m-term approximant or modify the decomposition algorithm so as to only select elements that will favor good fingerprint properties in $\hat{x}^m$.

In this work we have implemented the second approach, and the following fingerprint construction is performed. We have used a multiscale MDCT dictionary and a plain MP algorithm with the additional property that atom selection in the time frequency neighborhood of previously selected atoms is discouraged. The selection criterion at iteration i becomes:

$$\phi_i = \underset{\phi \in \Phi}{\operatorname{argmax}} \lambda(\phi, \Phi_I).|\langle x - \sum_{j=0}^{i-1} \alpha_j \phi_j, \phi \rangle|$$

where Φ_I is the set of previously selected atoms and $\lambda(\phi, \Phi_I)$ is a binary penalty term set to zero if any previously selected atom is in the time-frequency neighborhood of ϕ.

For a given analysis frame f_n, an approximant $\hat{f}_n^m$ is computed and the set of keys used by the fingerprinting system is simply the set of indexes of the m atoms chosen in the dictionary Φ. By limiting the decomposition to a small number m of iterations, the dimensionality can be greatly reduced. However, the fingerprint discriminative power increases with the number of atoms selected in the decomposition.

	CQT-Peaks	MP-150
Precision (%)	95.1	94.5
Recall (%)	97.8	91.5
F-measure (%)	96.5	93.0
CPU Time - Fingerprint (s)	0.12	0.33
CPU Time - Total (s)	0.20	0.40
Memory (MBytes)	9.3	2.4

Table 1. Frame by frame detection performances of the proposed framework with two different fingerprint mechanisms on 2 hours of synthetic redundant audio stream.

4. EXPERIMENTS AND COMPARISONS

4.1. Frame by frame evaluation

This task consists of determining for each incoming analysis frame whether it is a repetition of a previous frame. If so, the exact first occurrence in the stream is retrieved. A synthetic stream is built as a concatenation of 140 audio excerpts randomly taken from a pop song database[1]. Each excerpt lasts 30 seconds, 100 of them occur twice in the stream and the 40 remaining are not repeated. The total duration of the stream is thus 2 hours. Analysis frames are 5 seconds long, the complete dataset therefore consists of 1440 frames, 600 of which are exact repetitions of previous frames. The framework is evaluated with both presented fingerprints: the CQT-based fingerprint (labeled CQT-Peaks) and the MP-based fingerprint with a dictionary of 3 MDCT scales and stopped after 150 iterations (labeled MP-150). The two systems are compared in terms of precision, recall and F-measure. Additionally, we compare for each analysis frame the average time needed to compute its fingerprint and the total processing time (measured on the same Dual-Core CPU at 3.16GHz). Finally, we compare the size of the databases.

Table 1 summarizes the obtained results. The framework reaches a good level of precision with both fingerprints. The recall with the CQT-based fingerprints is better than when using the MP atom indexes. However, it is also more memory consuming. The MP-based fingerprints are smaller, but less robust as the recall shows. The CQT-based fingerprints are faster to compute but the matching process roughly requires the same amount of time for both methods. The results confirm the relevance of the proposed architecture as a generic repetition detection framework.

4.2. Real-world evaluation

In [11], the authors underline the importance of evaluating fingerprinting systems on real-world (*i.e.* coming from real broadcasts) data. We here follow their framework applied to

[1] http://quaero.org

Fingerprint	Detected rep. / Total nb	False Alarms
CQT-Peaks	191 / 191 (=100%)	0
MP-150	178 / 191 (=93.2%)	1

Table 2. Repeating objects detection scores for a real-world radio broadcast

a 24-hours long radio broadcast for the evaluation of our system.

It is virtually impossible to get a frame-by-frame repetition annotation on a real broadcast. On the other hand, the use of the annotations provided within Quaero allows a detection-based evaluation. The annotations provide for each broadcast song in the 24-hours stream the identifier of the song, its broadcast time and duration. We extract from the annotations a *repeat list*. It contains, for each song that is broadcast for the second time or more, its time of broadcast, its duration and the song identifier.

We have set the vote parameters so that the system only outputs long scale repetitions ($H = 9, \theta = 6$). With these, no repetition shorter than 30s should be detected.

Our evaluation dataset contains numerous short repeating objects that are not annotated (advertisements, jingles, ...). A fair evaluation can only be performed on the music titles broadcasts. Therefore, we limit the evaluation to repeating segments that are longer than 90s. This matches the set of broadcast songs, for which we do have the annotations.

The evaluation is then defined as follows. For each repetition detected by the algorithm, we check that it does actually correspond to one entry of the *repeat list* (meaning the detection time is within the bounds of one repeated song and the algorithm points to a previous occurrence of the song). If one repetition is detected and does not correspond to any entry in the repeat list, we count one false alarm.

The results are given in table 2. Although both algorithms have performed well, the CQT fingerprints makes no error in this task. The MP-based method misses a few songs and has output one false alarm on the 24 hours broadcast. Let us note that avoiding false alarms in this task is far less challenging than in the previous experiment since we only consider long scale repetitions. This second evaluation shows that the framework is suitable for industrial applications and that it does actually fit real-world use-cases.

5. CONCLUSION

In this work we have presented a framework for the detection of repeating objects in multimedia streams. A remarkable feature of this architecture is that it can handle any time-based fingerprint. We have applied this framework to two distinct audio fingerprints and evaluated its performance. The evaluation has shown that the system performs well in both cases. More interestingly, this shows that the framework can be used as a test-bed for drawing comparisons between fingerprints in this specific use-case. Our evaluation includes a real-world experiment that shows that the framework is suitable for the detection of repeating objects in an industrial context.

In the future, it would be interesting to compare this approach with a "standard" fingerprint algorithm. Usual fingerprinting systems indeed rely on a prior static database that contains all the items that can be found in the analyzed multimedia streams. Our system builds its database adaptively as repeating objects occur in the stream. A final open question is whether this system can compete with traditional systems on a typical fingerprint use-case such as the broadcast monitoring described in [11].

6. REFERENCES

[1] C. Herley, "Argos: automatically extracting repeating objects from multimedia streams," *IEEE Transactions on Multimedia*, vol. 8, no. 1, pp. 115–129, 2006.

[2] C.J.C. Burges, D.Plastina, J.C. Platt, E. Renshaw, and H.S. Malvar, "Using audio fingerprinting for duplicate detection and thumbnail generation," in *ICASSP*, 2005.

[3] J. Ogle and D. Ellis, "Fingerprinting to identify repeated sound events in long-duration personal audio recordings," in *ICASSP*, 2007.

[4] A. Wang, "An Industrial-strength Audio Search Algorithm," in *ISMIR*, 2003.

[5] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *CBMI*, 2001.

[6] C. Cotton and D. Ellis, "Audio Fingerprinting to Identify Multiple Videos of an Event," in *ICASSP*, 2010.

[7] S. Fenet, G. Richard, and Y. Grenier, "A Scalable Audio Fingerprint Method with Robustness to Pitch-Shifting," in *ISMIR*, 2011.

[8] M. Ramona and G. Peeters, "Audio Identification based on Spectral Modeling of Bark-bands Energy and Synchronization through Onset Detection," in *ICASSP*, 2011.

[9] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[10] M. Covell and S. Baluja, "Known-audio detection using waveprint: Spectrogram fingerprinting by wavelet hashing," in *ICASSP*, 2007.

[11] M. Ramona, S. Fenet, R. Blouet, H. Bredin, T. Fillon, and G. Peeters, "A Public Audio Identification Evaluation Framework for Broadcast Monitoring," *Applied Artificial Intelligence: An International Journal*, vol. 1-2, no. 26, pp. 119–136, 2012.