A Framework for Robust Audio Fingerprinting

Carlo Bellettini, Gianluca Mazzini
Department of Engineering, University of Ferrara, via Saragat, 1 44100 Ferrara, Italy
Email: {carlo.bellettini, gianluca.mazzini}@unife.it

***Abstract*—We present a framework for audio fingerprinting, rather general in its essence, but especially tuned for being used in the context of broadcast monitoring. We efficiently implemented a robust fingerprinting algorithm and a suitable retrieval method. Ample sections are devoted to strategies for improving both the reliability and the speed of the overall system. The outcomes of plentiful experiments on a database of 100 000 songs are analyzed, and two common kinds of distortion (pitching and thermal noise) are investigated. To better drive design decisions, we also provide in-depth discussion on the scalability of the indexing algorithm.**

***Index Terms*—Broadcast monitoring, audio fingerprinting, music indexing and retrieval.**

I. Introduction

Besides the unquestionable scientific interest, the availability of vast, digital archives of music and the related commercial interests are pushing many efforts into the field of automatic audio recognition. The opportunity of exploiting more computing power, and at steadily decreasing costs, certainly plays a key role as well. Indeed, chances are that what a decade ago would have been practical only by means of specialized hardware could now be accomplished in software (sometimes even on portable devices).

Though related to speech recognition, this truly multiform and demanding topic is actually addressed in distinct ways. Audio recognition may aim to discern whether or not two pieces are in fact the same, regardless of their outer appearance (i.e. coding, distortions). More generally, it comes into play when we are interested in reliably identifying an unknown excerpt of e.g. music, given a large set of references.

The task of automated audio recognition used to be accomplished by using invasive watermarking techniques. However, this requires either permanent human intervention, or that a single watermarked source accounts for every possible instance of a given pattern. Neither method is feasible: the former needs an active listener who correctly marks pieces, whereas the latter is clearly not realistic. A different approach is therefore needed.

A working, and actually very good, solution is the so-called audio fingerprinting [1], [2], known also as robust audio hashing. Its purpose is to allow electronic devices to identify perceptual similarities between audio contents. The term "fingerprint" recalls the fact that every piece of audio bears unique features, as detectable by listeners. We reserve the next section for more insights.

The usual system in which a fingerprinting algorithm operates, sees the building of a large fingerprint database, used as a reference source for identifying unknown fingerprints. Note that the search task is heavily demanding, since we are supposed to find the most similar (not simply exact) match in a huge amount of data (e.g. some 100 000 song fingerprints). Such tight search time requirements (a gain factor of roughly 100, at least, should generally be attained over real time) and its high complexity also led to putting considerable effort into the matching problem. In [3], for example, gene-sequence matching algorithms are borrowed from biology, allowing for a lightning-fast identification of an event-based fingerprint, computable over the output of any feature extraction algorithm.

On our side, we focused our attention on a simple yet very robust algorithm, whose basic operation was first described by Haitsma, Kalker and Oostveen in [4]. The distinctive feature it extracts is intimately related to the audio signal energy. More specifically, it takes into account how the energy difference among a set of sub-bands varies in time. In other words, it evaluates the second-order derivative of the energy spectrogram. The reasons that drove our choice and the details of the algorithm are provided in III-A.

In this paper, we analyze many essential aspects of algorithm [5]. We chose it as the core of a more general framework for audio fingerprinting, which we describe in detail. Focusing especially on the application of broadcast monitoring, we also provide effective strategies for improving the overall system and discuss the results of plentiful experiments, based on a database of approx. 100 000 songs. By broadcast monitoring we mean the continuous tracking of an audio source such as a radio channel. Musical TV and satellite channels may also be processed in a similar fashion, even if a combined approach including their video component would probably be a better choice. We believe our contribution can be exploited to both deepen the understanding of the topic and to better drive design and layout decisions in the field of automatic audio recognition.

The paper is organized as follows. In the next section, we discuss the main points behind audio fingerprinting and a broad selection of existing works. The system is detailed in section III, while in IV we introduce strategies for improving its performance, both in speed and reliability (the latter with particular regards to the broadcast monitoring application). In sections V to VII, we present

Manuscript received September 1, 2009; revised January 5, 2010; accepted January 28, 2010.

doi:10.4304/jcm.5.5.409-424

our scalable testbed, a viable solution to the pitch-shift distortion and the robustness of the algorithm to thermal noise. Some further scalability issues are addressed in VIII and finally IX concludes the paper.

II. Audio fingerprinting and related works

As we previously pointed out, the interest in the field has been steadily growing in these years, thus producing quite a large amount of good contributions. Significant reviews can be found in the cited [1], [2] and also in [6], but they date back to 2005 and cannot account for the latest developments.

First of all, we stress that applications are really manifold. For example, copyright issues could be easily addressed in p2p file-sharing networks or video sharing services, while meta-data retrieval and broadcast monitoring [3], [7] can be achieved. A music consumer could fast check whether his or her large collection already contains a given song, too [8]. Another nice application is the so-called "query by humming" [9], an early work in multimedia searching. A similar algorithm is now freely available on-line [10].

According to the researchers background and goals, the topic can be addressed in many different ways. The first step invariably involves isolating a sequence of these "features" in the piece of audio, and the longer the piece of audio, the more the features. This set of features is said to be the "fingerprint" of the piece.

Most of all, the fingerprint must be sufficiently distinguishable, so that two fingerprints can be reliably told apart or regarded as similar [11]. Along with the retrieving technique, it must prove also robust, i.e. exhibit high reliability even when various kinds of distortions occur, such as equalizing, white noise, pitching and so on. Finally, it must be fast to compute from e.g. a PCM (Pulse Code Modulation) signal, and also achieve a significant reduction factor in size, with respect to the original PCM samples.

On the basis of these considerations, we can also speak of "robust hashing" since, similarly to standard hashing techniques, audio fingerprinting aims to produce a compact and fast-to-check-and-retrieve representation of a complex audio signal. Usually, as in this paper, there is the need to work only on the waveform representation of the audio signal, with no further semantic aid (as a score would be). Needless to say, regular hash functions cannot be employed, since they rely on the precise bit sequence of the particular digitalization of the audio signal.

A large selection of different audio features can be extracted, usually chosen on the basis of heuristic considerations, with a few exceptions [12]. It is also possible to combine more of them into one single identification model [13], aiming to mutually compensate eventual weaknesses. However, this can be computationally expensive, if not prohibitive beyond small databases. Other less common approaches include Principal Component Analysis (PCA), as in [14] and [15], or statistical modeling combined with Hidden Markov Models (HMMs), for example [16]. Although effective, it is often difficult to render them efficient enough for real-time employment.

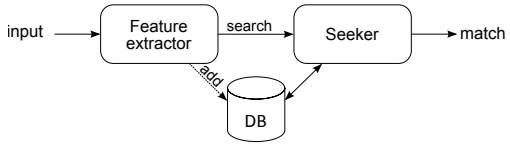


Fig. 1. Overview of the fingerprinting system.

The usual feature extracting technique involves a combined time-frequency analysis, mainly performed through Fourier transforms. A few works propose instead wavelet transforms, which may be a more rewarding choice [17]–[19]. A combination of both a FFT and a DCT is then employed in [20]. This paper is particularly interesting since it builds on [4], as our approach does, but introduces a further processing stage (which involves the DCT), in order to improve the indexed search reliability. The technique is especially effective when the audio query is rather distorted. However, it is not clear to what extent this extra stage elongates processing time. Moreover, since we focus on broadcast monitoring, the added complexity would not pay off, given the relatively mild distortions broadcast audio usually faces, for which the solutions we suggest in IV prove very effective.

On the retrieval problems, we point out the contributions [21]–[23], which mathematically argument effective strategies. In particular, the first two borrow respectively from classical full-text indexing and coding theory. As far as the latter is concerned, we stress that its applicability in some contexts is limited by the use of short clip fingerprints, instead of whole songs, and in this respect it is not trivial to foresee its possible efficiency, when extended. A second drawback is that, at its present state, it is not able to track an audio stream (e.g. to evaluate the duration of a broadcast song), since the scheme is inherently incapable of handling cropping. For this reason, there are even cases in which two different songs could be regarded as identical, for example when their fingerprinted excerpts are a same musical section, while the sung parts differ (i.e. different performers).

III. System description

As previously hinted, our system follows the usual architecture of three blocks (see Fig. 1): the feature extractor, the fingerprint database and the seeker. The extractor must be fed with some audio input, while the seeker produces a match. Such match can be found by comparing against the reference database.

Though sufficient for evaluation purpose, these blocks fall short in real-world applications. For example, we may be interested in processing the output of a microphone or of a radio receiver. Therefore, a transcoder must be used to properly format the input. Our implementation takes audio files as input, handling a variety of compressed and non-compressed formats, and then works on raw waveforms. In particular, MP3 files (or parts thereof) were used as input.

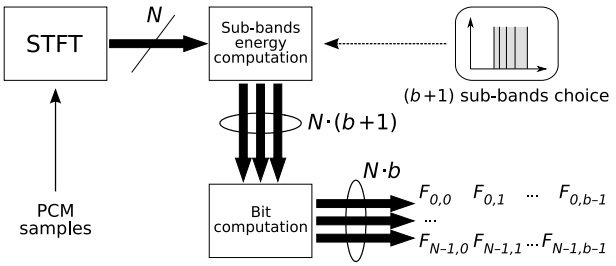


Fig. 2. The Streaming Audio Fingerprinting (SAF) algorithm.

Post-processing is also needed, first of all to correctly interpret the outcome of the seeker, say match a given id to the correct metadata. With the aid of Fig. 1, let's now examine in detail how things work.

A. Feature extractor

The feature extractor is the most relevant block, since it greatly accounts for the overall effectiveness of the system. On the other hand, a highly efficient seeker must be implemented (see III-D).

We recall that the feature extractor is an algorithm that must produce the fingerprint of a piece of audio, by selecting some of its distinguishing characteristics. Among the many methods investigated in the last years, we selected that proposed by Philips in 2001, in its revised and extended version [5]. Beyond its experimented effectiveness, we mainly based our choice on its great flexibility. According to its original proposers [24], we will denote the feature extractor algorithm as Streaming Audio Fingerprinting (SAF), but it is also known in the literature as Philips Robust Hash (PRH).

The Philips approach provides a convenient and easy-to-handle output (a bit matrix), and ample opportunity of customization. Moreover, its simple steps allow some kind of statistical modeling, though only approximated [25], [26].

Let's follow Fig. 2. The input is constituted by raw PCM samples, whose frequency must be $44\,100$ Hz, with a quantization level of 16 bit/sample[1]. A stereophonic input is immediately converted to monophonic by averaging the two channels.

Since the subsequent processing discards much of the original bandwidth, the input is downsampled to 5 kHz by a low-pass filter and a decimator. Its integer period varies so to have, on average, the correct downsampling ratio. In the present case of PCM samples extracted from MP3 files, we verified that a rough equiripple 16-tap FIR filter (designed with Parks-McClellan algorithm) gives the same results as a more refined 41-tap filter, which should be the minimum order for the required specifications. Therefore, we used the faster 16-tap filter.

As is often the case in the audio analysis field [27], the algorithm works on the input spectrogram. It can be obtained by taking first a Short-Time Fourier Transform (STFT) of the audio samples, and then the squared modulus of the transform (energy). This gives N frames in the frequency domain, where N is linearly proportional to the length of the input.

The three steps for the STFT are summarized in Fig. 3:

1) input segmentation into N *overlapping* frames;
2) windowing of each frame according to a convenient function;
3) discrete Fourier transform by means of a FFT (Fast Fourier Transform) algorithm.

Here, the length of each frame is $16\,384$ samples (or approximately 0.37 s), while the overlap leaves out only 512 samples, or $\frac{1}{32}$ of the frame length. These parameters lead to a good trade-off between frequency and time resolution. Such a large overlap is crucial in the further step of recognizing unknown excerpts. Not only does it allow a comparison between misaligned framing structures, but it also makes the system more robust to distortions: a good algorithm should always produce fingerprints that look very much alike, regardless of the particular framing offset.

The window function considerably smooths the frame to be transformed and alleviates the problem of the spectral bias in the spectrogram. As in [5], the choice fell on the Hann window, which has been proved to be nearly optimal as to its bias characteristics in this context.

As a further step, each transformed frame is sliced into $b+1$ bands. Differently from the original paper, we used as reference the usual 12-TET tuning system, keeping frequencies up to about 1976 Hz and setting the lower bound l as a function of b, given the constraints imposed by the musical scale. In particular, its value in Hz is given by

$$l = 440 \cdot 2^{\frac{26-b}{12}} \quad .$$

This division is beneficial since it allows to successfully combat the pitch distortion, an intrinsic weakness of the original system. The bandwidth kept depends on the particular value of b. The details are presented in section VI.

The distinguishing feature considered by the algorithm is the energy difference of sub-bands, both among them and in time. By summing every relevant sample, the energy of each sub-band is computed as a contribution $E_{n,m}$ where $0 \leq m \leq b$ is the sub-band index and $0 \leq n \leq N-1$ is the frame index. A value $\mathrm{FP}(n,m)$ is finally obtained according to the rule:

$$\mathrm{FP}_{n,m} = \begin{cases} 1 & \text{if } \alpha_{n,m} - \alpha_{n-1,m} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathrm{FP}_{n,m}$ (for which $0 \leq m \leq b-1$) is the m-th bit relative to frame n and α is the energy difference among

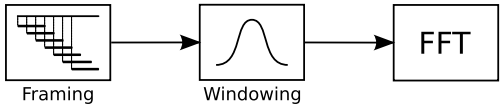


Fig. 3. Rationale of Short-Time Fourier Transform.

[1] We observe that these limitations can be easily circumvented by the pre-processing stage, before feeding the feature extractor.

contiguous sub-bands, namely

$$\alpha_{n,m} = E_{n,m} - E_{n,m+1} \quad .$$

We highlight that there has been at least one proposal to improve the binarization [28], but at some more computational expense. Note that, for a given n, $\mathrm{FP}(n,\cdot)$ is a b-bit vector spanning the considered bandwidth and representative of a very tiny fraction of the whole song. Conversely, when m is fixed, $\mathrm{FP}(\cdot,m)$ is a N-bit vector which roughly gives the energy variation trend in a sub-band. The sequence of the N b-bit vectors, obtained from the N temporal frames, is the fingerprint of the input piece of audio. An example is depicted in Fig. 7a, where the patterns are stacked vertically and the frequency dimension is horizontal.

It is worth noting that, given the length L in seconds of an excerpt (sampled at a sample frequency f_s), the number N of patterns (lines) extracted as fingerprint may be easily computed as

$$N = \lfloor \frac{L \cdot f_s - \mathrm{frame\_length}}{\mathrm{skip}} \rfloor + 1 \quad .$$

B. Fingerprint database

The database is basically an archive of song fingerprints. As we have seen in a previous section (II), many different approaches can be exploited to manage and search the database, also depending on the particular representation of the extracted features. Here, the database is a collection of binary files, possibly organized in distinct folders, each representing a single and whole song. It currently holds $100\,000$ entries, highly differentiated in genre and mostly of western origin. Its actual organization is detailed in V-A.

Normally, no extra-processing is performed over the extracted fingerprints, though a further layer could be easily inserted here to provide additional functionalities. For example, it would make the database either more compact, to save storage space, or more convenient, to save search time and to be more reliable or robust in seeking. An example is introduced in IV-D.

C. Distance metric

Regardless of the particular method employed for searching, a distance metric must be defined between the fingerprint X of an unknown excerpt and a fingerprint R referenced in the database. In practice, being R typically much longer than X, we define

$$d(X,R) = \underset{\tilde{R}}{\operatorname{argmin}}\, d(X,\tilde{R}) \quad ,$$

where $\tilde{R}$ is a fraction of R as long as X. How do we define $d(X,\tilde{R})$? The binary representation induces to use as metric the normalized Hamming distance, i.e. the number of different bits between X and $\tilde{R}$, divided by the total number of bits of X (or of $\tilde{R}$, since they are the same). That is:

$$d(X,\tilde{R}) = \frac{1}{N \cdot b} \sum_{n=0}^{N-1} \sum_{m=0}^{b-1} X_{n,m} \oplus \tilde{R}_{n,m} \quad ,$$

where $\oplus$ denotes the bitwise exclusive OR. One or more thresholds on the value of $d(X,\tilde{R})$ can be chosen, so to provide a hard indication of the similarity between the two excerpts. On the basis of our experiments, good-quality excerpts give always a correct match when $d(\cdot)$ is below 0.30.

However, distortions (such as recordings in harsh environments) tend to significantly increase the acceptable threshold value. Prior knowledge on the quality of the input may be of help in setting the threshold. Clear examples of this phenomenon are shown later in VI-C and VII. Our proposal there to tune the threshold according to the devised task, tries actually to heuristically address the "metric learning" problem, very recently investigated in [29].

In general, d will always be greater than zero, even if X and $\tilde{R}$ come from the same non-distorted song, due to the non-synchronized framing. If we were to exhaustively search the database, we would adopt a minimum distance approach, taking as a match for X a region $\hat{R}$ such that

$$\hat{R} = \underset{\tilde{R}}{\operatorname{argmin}}\, d(X,\tilde{R}) \quad ,$$

where the search is extended to the whole database. If this value is above the reference threshold, we can safely conclude that no match was found.

D. Seeker

Given the fingerprint of an unknown excerpt, the seeker has the role of either finding the best match in the database, or declaring that the excerpt has no match. As a corollary application for the seeker, we observe that it can be a precious help also in updating the database. Consider the following scenario: a fairly large database as ours, of roughly $100\,000$ entries, is currently available. Every week or so, there's the need to enlarge it, in order to comprise recent acquisitions: the availability of a reliable seeker can successfully avoid adding duplicates [8], which could lead to troubles.

The naïve searching method requires an exhaustive comparison (i.e. linear scan) against the whole database, but this becomes rapidly unfeasible as the database grows, even with a careful implementation. On the basis of our experience and for good-quality songs, some computation may be saved reducing by an integer factor $2 \div 10$ the alignments tried, relying on the large overlap in the STFT block. However, brute-force is highly inefficient.

A possible work-around, which scans anyway every database entry, is described in [30]. There, we compute the distance only on promising regions, whose locations are determined by the peaks of the cross-correlation between X and each possible R. The cross-correlation is performed along the time dimension and averaging along the frequency axis. An example is reported in Fig. 4. Nevertheless, also this approach requires too much time for being used in practice, even if it can be taken into consideration when careful investigations must be carried out. Indexing (IV-A) is probably the best choice.

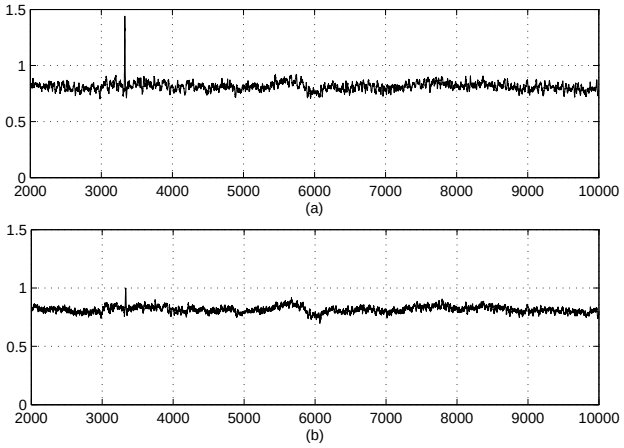


Fig. 4. Cross-correlations between an unknown excerpt and the matching reference fingerprint. In case (a) the excerpt is undistorted and the synchronization point is easily detectable in correspondence of the high peak. On the other hand, in (b) the test excerpt is corrupted by white noise (SNR of 5 dB), which impairs the peak detection.

IV. Performance improving

This section proposes a number of strategies for improving the reliability or the speed performance of the system, with special emphasis on issues connected to the broadcast monitoring application. Some of them are later discussed and evaluated in more detail.

A. Indexing

Following the rationale suggested in [5], we implemented our seeker on the basis of a simple, yet effective, method. A lookup-table (in fact, a *chain* hash table) acts as an index to quickly retrieve all the locations of a given pattern in the database. By "pattern" we denote any of the possible b-bit binary vectors, as those computed by the feature extractor, and by "location" we mean the song and the relevant offset, or an equivalent information.

When a match does exist, if we suppose that the unknown fingerprint and its matching reference both exhibit at least one feature vector exactly alike, then the index may be indeed effective. Querying it for the patterns computed from the unknown excerpt will give a number of promising alignments, thus dramatically reducing the number of comparisons needed, in the same fashion of the cross-correlation method. Moreover, look-ups are thousands of times faster (the exact ratio depending on the parameters used, compare section VIII) than linear scan.

Ideally, the index would list all possible patterns, but this is not always feasible or advisable. Memory constraints may render impossible handling a 2^b-entry look-up table (and permanent storage would be too slow), and excessively distributing the patterns in too many bins may lead to missing the correct synchronization points, as we verified. We thus propose to:

1) hash each b-bit pattern w in the database according to some convenient function $h(\cdot)$, giving a k-bit word $\tilde{w} = h(w)$, where $k \leq b$;
2) update entry $\tilde{w}$-th of the look-up table with the location (song and offset) of w, possibly chaining it to the existing ones (*separate chaining* collision resolution method, by means of a linked list).

The hash function carries out two roles. Firstly, it is needed to reduce the pattern space from 2^b to 2^k, so that standard amounts of memory (e.g. 4 GiB) become acceptable. Secondly, it greatly helps in equalizing performance. In this respect, it should be noted that the pattern distribution in the database is highly uneven, and in real-world application it is important to precisely foresee, and take into account, the time required for a search task.

Conversely, we point out here, and numerically justify in VIII, the need of a large enough k: too small a value, actually, would lead to a small number of large buckets in which patterns are collected, so that the search time could still be unacceptably long. Let's now dive into the details of our implementation.

B. Pattern hashing

We understand that the unstated hypothesis of having at least one exact match in the look-ups is quite strong. However, we propose the following pattern hashing method, which greatly relaxes this constraint and achieves very good results, as will be immediately cleared.

The overall hashing process is summarized in Fig. 5, where the dashed frame encloses the rationale just devised. In implementing the look-up table, the choice of a good hash function $h(\cdot)$ must not be underestimated. Apart from distributing the patterns as evenly as possible, it must also be very fast to compute. Two good algorithms with these properties may be found at [31] and [32]. Both were carefully evaluated and compared, giving similar results, and we chose the former.

Since it realizes a 32-to-32-bit transformation, we kept a number k of the least significant bits of the hashed pattern as the search key. In other words, the key used for building and then querying the index is actually computed as $h(w) \otimes (2^k - 1)$, where h is the hash function, x the input pattern and $\otimes$ denotes the bitwise AND operator. Values from $k = 16$ through $k = 26$ at steps of 2 have been tested.

When the indexing technique is used, we also observed a significant improvement in the recognition error rate if some bits of the patterns are cleared prior to the hashing. This means replacing $w \rightarrow w \otimes \mathrm{mask}$, where mask is a convenient b-bit value. The dimensionality reduction of the pattern space spreads the patterns in a lower number of bins in the hash function target space, so the increased search time must be properly taken into account.

Fig. 6 clearly highlights this dependence, for some special cases and $b = 32$. In particular, we represent $\mathrm{mask} = \mathrm{0x0FFFFFFF}$ as an example of clearing 4 bits and $\mathrm{mask} = \mathrm{0x00FFFFFF}$ for 8 bits, when the duration of the excerpt is 5 seconds.

When $b = 32$, we experimentally found the best results by setting $\mathrm{mask} = \mathrm{0x00FFFFFF}$, which leads to acceptable search times. We believe that this choice is rewarding

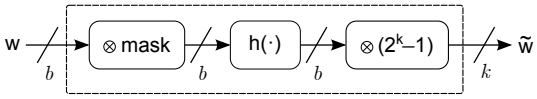


Fig. 5. Scheme of the algorithm that transforms each single feature vectors from b into $k \leq b$ bits. The transformed vectors then populate the hash table used for efficient pattern look-ups.

because of the high variability and inherent weakness of the particular bits involved. In our implementation, they are representative of the lower frequencies. Since it is not possible to exhaustively try the whole range of possibilities, further investigation may be required at this stage. We also point out that this approach is related to the bit-toggling solution illustrated in [5].

C. Caching

The computer volatile memory is typically orders of magnitude faster than ordinary storage (i.e. the hard disk). To better drive implementation decisions, a variable-size cache was thus implemented in RAM, with respect to the fingerprint archive. Although there may exist more refined replacement policies, we used a FIFO (First In, First Out) ring buffer. Whenever part of an entry (reference fingerprint) is needed, the database entry is loaded in its entirety in the first available bin, for future reference. In many cases, as when tracking an audio broadcast, it is reasonable to assume that many excerpts from the same song will be present. When there are no free bins, then the oldest one is replaced. The size of the cache has been varied from 0% to 100% of the database size, at steps of 10%.

D. Runs filtering

In Fig. 7a we report a sample, short fingerprint. Because of the high temporal correlation between the overlapping segments of the input, the bit matrix is mainly constituted by runs of 0s and 1s. We note anyway some "undecided" regions, where 0s and 1s tend to alternate. The negative

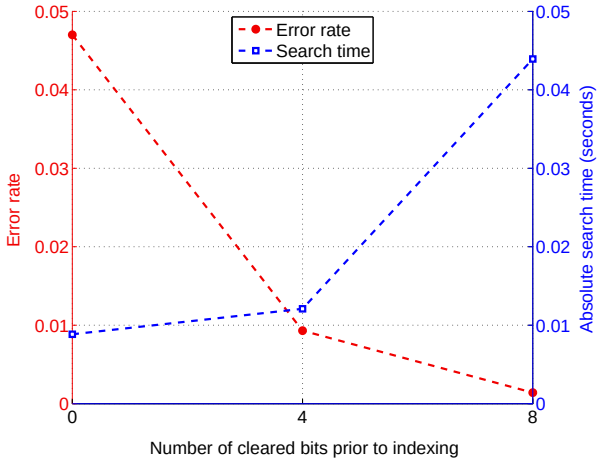
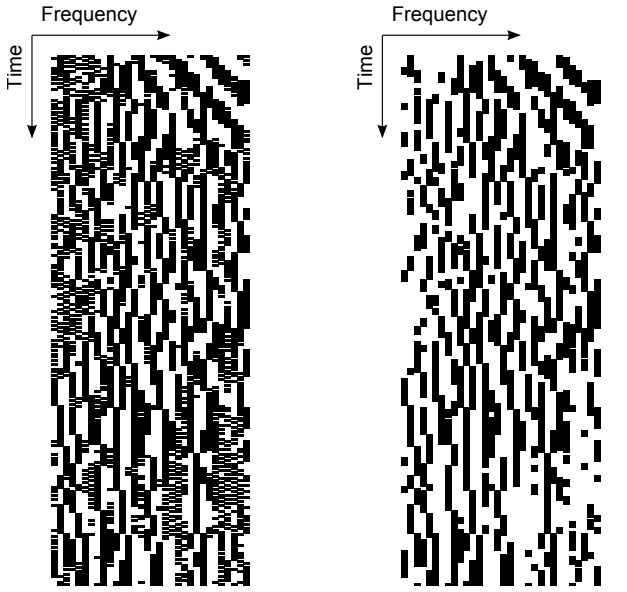


Fig. 6. Impact of clearing bits in the computed patterns.



(a) (b)

Fig. 7. Sample fingerprints (a) before and (b) after filtering with $\delta = 3$. 256 32-bit patterns, each representative of a time frame, are vertically stacked, thus spanning the frequency dimension horizontally. The duration of the chunk is about 3 s. δ denotes the magnitude of the runs filtering described in IV-D.

impact on the overall system, in the long run, is twofold: not only does it lead to an increased distance between excerpts and possibly matching references, but also limits the robustness in case, for example, of pitch-shifted audio (VI).

These thoughts suggested us to add little processing to both the unknown and the reference fingerprints, in order to keep only runs of a minimum configurable length δ, as depicted in Fig. 7b for $\delta = 3$. After this extra stage, the operation follows as usual with the eventual look-up of promising locations and then distance metric computation. We will show that this precaution (i.e. the runs filtering) allows, to a certain degree, to obtain an average smaller distance in case of match. At the same time, it does not reduce too much the distance with respect to non-matching entries. Finally, we point out that it does not cause the database to be rebuilt and can be easily done on the fly at the time of loading.

E. Database partitioning

It is intuitively clear that the search time is inherently related to the size of the database, as we will see in detail in section VIII-C. Therefore, finding a way of partitioning the database could be a practical way to reduce search times and complexity, beyond increasing scalability.

A first approach could be that of determining the genre of a song, e.g. following [33], which is also a good review of some relevant works, or [34]. Unfortunately, there is no expressed agreement on the definition of "genre" and if it were, it would change over time, along with the common perception of music. Moreover, its evaluation is usually computationally expensive. We thus propose to

classify songs on the basis of their *tempo* (or beats per minute, bpm), which in many cases can be objectively measured with a good accuracy. An additional bin may be considered for all those songs whose tempo is unclear, irregular, or anyway needs human supervision for being estimated.

Tempo can be described as the rhythm pattern of a piece of music and characterize especially western music. If the beats per second are constant along a song, then we can compute them from each song and partition the database into sections, each of them including a specific beat range, possibly non-overlapping.

On the other hand, song beat may differ in different parts of a song, e.g. between verse and chorus. In that case, the database sections may overlap and the same song may be assigned to more than one group. This leads to a loss of efficiency, but does not impair the correctness, as long as the beat estimation is correct.

Many approaches for beat estimation have been proposed, and even a contest was held for finding good algorithms [35]. We also propose the following, which is straightforward and gave good results on many songs taken from a commercial radio broadcast. Assuming that the song exhibits energy peaks on beats, we compute the temporal cross-correlation between its PCM samples and a number of impulse trains of convenient period. We then take as beat reference the train with the highest correlation with the song. In order to reduce the complexity, the song may be also downsampled, and possibly segmented in order to take variations into account.

The periods of the impulse trains may be chosen among a wide range or it may be limited by prior information. The impulse is bell-shaped, with a duration of 100 ms. For efficiency, the impulse trains should be generated just once and then retrieved when needed.

When the song presents a distinct rhythm pattern given by a drum or a similar instrument – as is the case of many rock and pop songs –, our approach yields typically little or no errors, at most within a few bpm of what a human experimenter would feel right. However, it is unable to produce meaningful results when no rhythm patterns is clearly perceptible (e.g. melodic and classical music).

Given the size of the database, the number of segments and the computational burden required by the beat estimation, it will be possible to evaluate the trade-off between the effort needed to estimate the tempo and the advantage derived from the database partitioning. Although the database can be partitioned off-line, we stress the need for a fast beat estimation algorithm, in order to actually have an overall processing gain, since the beats per minute must be estimated on the input chunks as well. This constraint may be lifted only if we can rely on prior tempo information. We currently do not have an implementation efficient enough to prove useful, but according to [35], the beat estimation could take as low as 0.02 times the excerpt length.

It is worth noting that the beat estimation may impose some constraints on the minimum excerpt length. However, encouraged by our tests, we believe that a value of 5 seconds, adopted and justified in the following sections, should be enough for most cases.

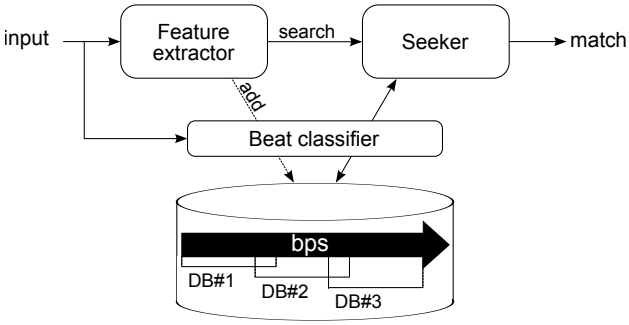


Fig. 8. Database partitioning through beat classification, with possibly overlapping sections.

F. Speech and silence detection

In the view of broadcast monitoring and tracking, a mechanism of speech detection may prove really profitable. On the other hand, we are aware there are notable applications which would not benefit from it, like the addressing of copyright infringement issues in file-sharing networks. Nevertheless, we believe it is worth discussing.

First of all, speech detection can be implemented by pre-processing the audio prior to the fingerprinting, thus allowing to discard non-musical sections and achieve a better efficiency. Secondly, it could also occur at the output of system, trying to analyze what has not been recognized, along with other blocks, such as a recognizer that operates on the whole database instead of on just the group determined by the estimated beat.

As in other relevant works [36], our approach to speech/music separation exploits a well-known feature, the zero-crossing rate (ZCR). It can be computed directly on temporal frames and gives a rough indication of the frequency range in which most of the energy lies. Since the human speech is bounded up to a few hundreds hertz and music usually involves also much higher frequencies, the ZCR allows to discriminate the two cases.

Of course, since this method has no real understanding of the content of the eventual speech detected, it is not able to tell apart an actual speech from a rap excerpt or a sung part with little or no music. It can also be tricked by loud background music.

By considering the frequencies of human speech, we enhanced this method by computing the fractional residual energy outside this band. Combining the ZCR and the residual energy indicator, we have very good estimation results, tested on many real radio recordings. The exact parameters can be tuned according to the particular context, so to allow for a flexible tolerance.

Similarly to the beat recognition case, the speech detection algorithm is implemented off-line and is not part of the audio recognizing system. Moreover, it constrains the excerpt length, though a few seconds are enough for a reliable speech detection.

We conclude this section by noting that it is also important to discriminate silent inputs, as they easily match randomly against the database and lead to false positives. We do take into account this case by making sure that the total energy of the processed excerpt is above a threshold, which was heuristically tuned. In particular, we evaluate whether more than 40% of the waveform samples (normalized in the range $[-1, 1]$) have an absolute value greater than 0.02.

G. Post-processing of results

When analysis is made to carefully track the content of an audio stream, as in the broadcast monitoring application, the reliability of the audio recognizer can be greatly increased by taking into account the analysis history. That is, those excerpts difficult to recognize can be handled by looking at the matches previously found. Even better, in most situations it is possible to take advantage of future matches as well.

We point out that the duration of the excerpts in the stream to track must be carefully chosen. Since each excerpt will provide one and only one match, it cannot be too long, otherwise we will easily mix different songs. On the other hand, too short an excerpt won't provide enough information for a reliable matching. On the basis of our experiments and previous works, we suggest excerpts of 3.3 to 5 seconds each (compare also Fig. 9).

Once the stream, or a sufficiently large part of it, has been processed, our algorithm verifies whether some excerpts could not be recognized. Unknown excerpts surrounded by long enough segments of a same song, are assigned to it as well. The particular values to use highly depend on the context and cannot be determined once for all. By this strategy, however, we were able to correctly and precisely track ambient recordings made in a very noisy discotheque, under the hypotheses of a minimum duration for each song.

V. Implementation details and

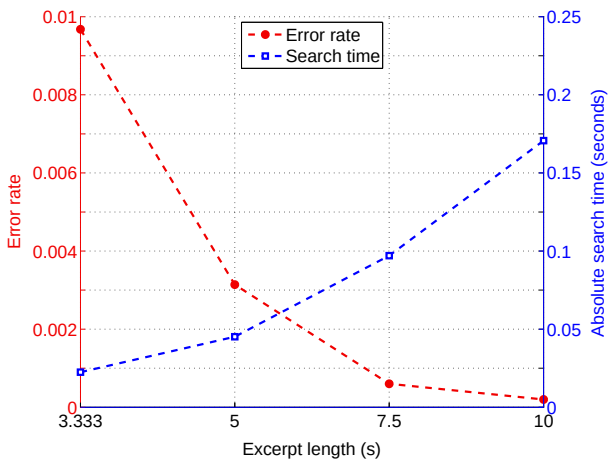


Fig. 9. Overview of the impact of the excerpt length on the error rate and on the search time ($k = 24$).

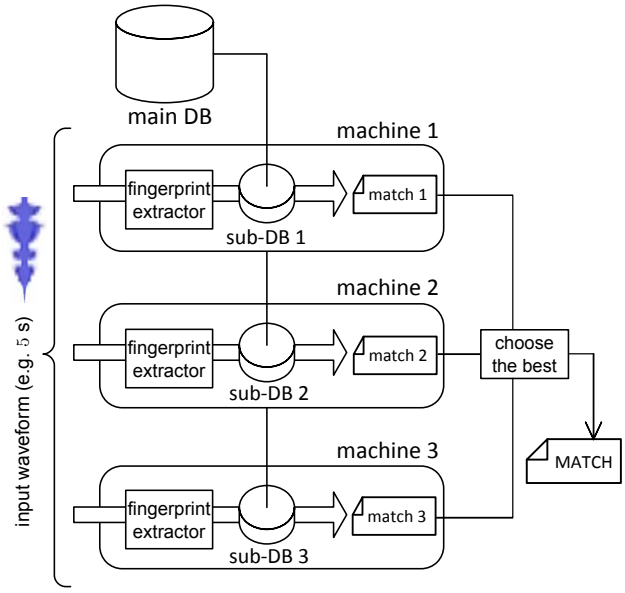


Fig. 10. Overview of the off-line recognition process. For clarity, only 3 sub-databases are depicted here.

PRELIMINARY INVESTIGATIONS

We first present our testbed and then, through the rest of the paper, we discuss the outcomes of ample experiments on SAF, the fingerprint algorithm detailed in III-A. For reasons of efficiency, our implementation is written entirely in (portable) C++. Tests were run on 32-bit middle-end desktop PCs, with 3.00 GHz CPUs, and 3.5 GiB of available RAM. If not stated otherwise, tests were carried out on excerpts whose duration is 5 s, corresponding to $N = 399$ b-bit vectors with the parameters given in III-A, where $b = 32$ and the band division is 12-TET.

A. Testbed

With the proposed parameters, we extract about 5168 4-byte feature vectors per playing minute. Since the average song length in our $100\,000$-song database in MM:SS is 4:18, we have a total storage of 8460 MiB, or 87 KiB per track, with $2.22 \cdot 10^9$ feature vectors stored.

We highlighted in IV the strong need for both indexing and operating in the RAM memory. Even leaving out the size of the index, however, the size of the database alone rendered this task unfeasible to the machines available to us. We thus devised and operated the following solution, as represented by Fig. 10:

1) we evenly split the database into 10 sub-databases, each assigned to an independent instance of the recognizer. To be precise, we used 10 sub-databases of $10\,000$ entries each, but on average, this is the same as evenly dividing the total database size, which linearly depends on the duration of the songs;
2) each of the 10 independent recognizers is then fed with the same audio excerpt, randomly extracted

from those existing in the database[2];

3) we finally merge the results, taking into account the smallest distance among the 10 distances independently computed across the 10 sub-databases. Decisions on the acceptability of such a distance follows as previously described.

On the positive side, we highlight that the instances of the recognizer may safely be separate processes, since each will deal with just a tenth (or a fraction, in any case) of the whole database, independently. The same rationale can also be easily customized and tailored to one's needs and resources (e.g. some machines are faster than others).

If we stick to an on-line recognition task, the main drawback is the need for either 10 rather modest machines (which is our case and it is cheap), or for one or more powerful ones (e.g., a 16 GB RAM machine would be able to handle approx. 70 000 fingerprints, if we include the indexing structure). Post-processing is not an issue, since it can be done on the fly (given suitable inter-process mechanisms) and requires practically no time.

Nevertheless, for the sake of easiness of implementation, we performed off-line the merging step, which is indeed realistic for broadcast monitoring. We also stress that in this and other contexts which do not require an immediate answer to an audio query, part of the audio stream can be processed multiple times, sequentially, by a same machine, which iterates through the sub-databases. Loading a sub-database from disk induces some temporal overhead, but this is acceptable if the audio stream is long enough (see also table II). The good gain of processing time over real time must be also considered (compare section VIII).

We observe that the blind database partitioning just discussed is instrumental to allow a certain degree of scalability, and is logically orthogonal to the tempo-based partition suggested in IV-E. The two approaches can thus be profitably combined, or even introduced at different times into an existing architecture.

B. Effectiveness investigations

We speak of "success" when the system gives as match the correct reference. The "error rate" is thus computed as the one's complement of the success rate, obtained by averaging over a very large number of independent trials, in a Monte Carlo fashion.

While the error rate figure is given by comparing against the whole 100 000-entry database, we point out that all the timing results presented in later sections are related to the single sub-databases of size 10 000: the sub-databases are in fact approximately homogeneous and browsed concurrently, and we can safely neglect the post-processing time.

First of all, we verified through brute-force search that the fingerprint algorithm under analysis is thoroughly effective on undistorted excerpts. In this case, the error rate is exactly 0, even for very short excerpts of just 3.3 seconds.

According to IV-A and IV-B, indexing and pattern masking are then introduced. We report in Fig. 9 the trade-off between accuracy of the match and search time required per each excerpt, with respect to its duration. While we observe a significant drop-off in the error rate switching from 3.3- to 5-second excerpts, the improvement is less noticeable if we further increase its duration. Thus, we chose 5 seconds as the reference duration for excerpts.

VI. Addressing pitch distortions

The SAF algorithm proves highly reliable with respect to a large number of signal degradations [5], as we also experimented. However, due to its inherent characteristics, this holds as long as the frequency and time texture of the song are not significantly altered. As reported in [24] and [37], even a moderate time- or frequency-scaling voids the effectiveness of the system.

By "pitch distortion" we denote a lowering or a raising in pitch of an audio signal, without affecting the tempo (time scale), i.e. the time length of the excerpt is unchanged. Note that this is first of all a frequency shifting, but it also involves a dilation in the frequency domain, in order to preserve the musical relationship of the harmonics [38].

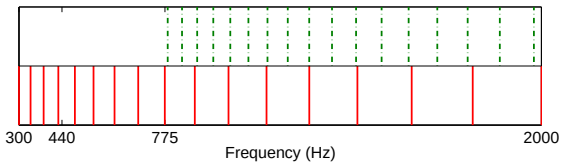
As with any other distortion, a trivial approach is trying to restore (e.g. "de-pitch") an excerpt before taking its fingerprint. If correctly carried on, this is always successful, but most probably unfeasible, as seen later on.

As hinted in III-B, we could also combat these weaknesses by re-encoding each fingerprint, e.g. as a sequence of symbols, untying this new fingerprint from the original song structure and allowing for a more tolerant search. An example was cited in the introduction [3]. Both time- and frequency-scale distortions could be successfully addressed, with the great benefit of building over the existing database, and with no need to switch to a different fingerprinting algorithm.

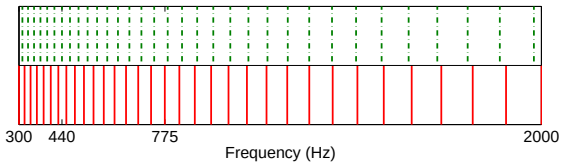
However, this approach can be overkill for most of the applications. Then, it would be interesting to exploit the existing data without considerable modifications. We thus propose a simpler method, which can successfully address very high pitch distortions, giving at the same time very good results.

The band division introduced in III-A exploits quite an often used tuning system, at least as far as the western world is concerned: equal temperament, or 12-TET. Leaving out the historical reasons which drove his adoption, we only point out that it is built from a basic tone in the neighborhood of 440 Hz (called A4 or La4). In our case, this exact value was used. Each of the 12 available notes is then obtained by multiplying (or dividing) the frequency of the adjacent tone by a factor of $2^{\frac{1}{12}}$ (as seen in III-A). This interval is called a "semitone".

[2]The issue of false positives has been addressed in [5], and in our experience we honestly never found none.



(a)



(b)

Fig. 11. Comparison between 12-TET (dashed) and EQL sub-bands, in linear scale. (a) $b = 16$ (b) $b = 32$

The iterated procedure leads to the musical scale (C, C♯, D, ...B) or (Do, Do♯, Re, ...Ti), probably known to the reader, where C♯ and D♭ (and so on for analogue couples) have actually the same frequency. The interval between a tone and its doubled-frequency counterpart is called "octave", and is such that we perceive two sounds an octave apart as having the same pitch. In conclusion, 12-TET is a tuning system where we can perceive as much as 12 unique pitches. Note that 12-TET is not related to the Bark scale.

For the sake of comparison, we built an additional database, so to evaluate the response of the system with respect to two different band quantization choices. The most straightforward approach is to log-equally divide a meaningful bandwidth, justified by the fact that the human ear has an approximate logarithmic response. The base used for the logarithm is 10 and the performed subdivision will be denoted by EQL. We report in Fig. 11 a comparison between the two schemes, 12-TET and EQL, for the cases $b = 16$ and $b = 32$. The former case leads to a lower bound of $l = 784$ Hz, whereas the second to $l = 311$ Hz. This last is very similar to the $300 \div 2000$ Hz bandwidth suggested in [5].

A. An alternative search strategy

Thanks to the 12-TET band division, it is possible to conceive an alternative search strategy. Taking up the picture of the unknown fingerprint slid along the time dimension of the database, we propose to extend the search to the frequency dimension, thus shifting by one or more bits the fingerprint of the excerpt prior to the comparison. Depending on the expected magnitude and direction of the distortion, the search should be appropriately limited, in order to save time. Since a frequency shifting is involved in pitching, we will show that this method allows to greatly improve the robustness of the recognizer, at least for not excessively strong distortions.

The major drawback is the impossibility to use the indexing technique described in IV-A with no modifications. Therefore, the time required by the tests presented here is of the same order of magnitude of a brute-force approach (see also VIII-C), with the further multiplying factor of a frequency-wise shifting. At the moment, with realistic database sizes, this can be acceptable only when a deep automatic analysis is required or there are no strict time constraints. This can well be for certain off-line monitoring tasks, as the processing of a relatively-short daily stream, or the analysis of a weekly event.

Although the proposed 12-TET band division is based on a western convention, we point out that the described search strategy holds regardless of the specific kind of music being processed, save the limitations discussed in the following section (i.e. the tailoring of the band division for combating a conceivable distortion granularity).

Recalling the "de-pitch" approach, it would naturally fall in the pre-processing stage, or – more efficiently – just after the STFT computation, whose large time-overlap parameter will greatly benefit the accuracy of the pitching operation [38]. On the positive side, this approach always compares full b-bit words, but in turn it requires a complex, non-negligible computational effort, and almost always some trial-and-error (since it looks improbable to know in advance the intensity and direction of the distortion, even if just approximate). Conversely, performing a bit-shift operation is simple and almost instantaneous, although the number of actually compared bits is linearly reduced by the magnitude of the bit shift.

Another more subtle issue is due to the choice of the frequency domain quantization in the fingerprint algorithm. At the fingerprint level, it actually maps a bit shift to a fixed frequency shift, so it accounts for at least two major drawbacks in the fingerprint-shift approach. Firstly, it allows to correctly handle pitch distortions only if their magnitude falls about evenly on quantization boundaries and secondly, it poses a limit on the maximum tolerable distortion. In other words, the bit shift cannot be too large, in order to retain a significant fraction of the available information.

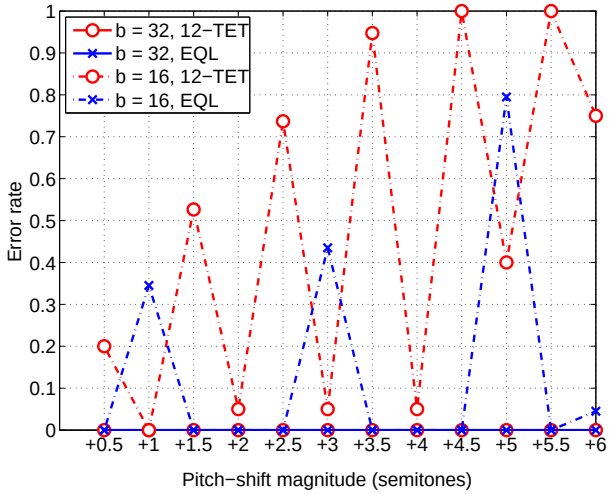


Fig. 12. Error rate as a function of the pitch-shift and for different parameters.

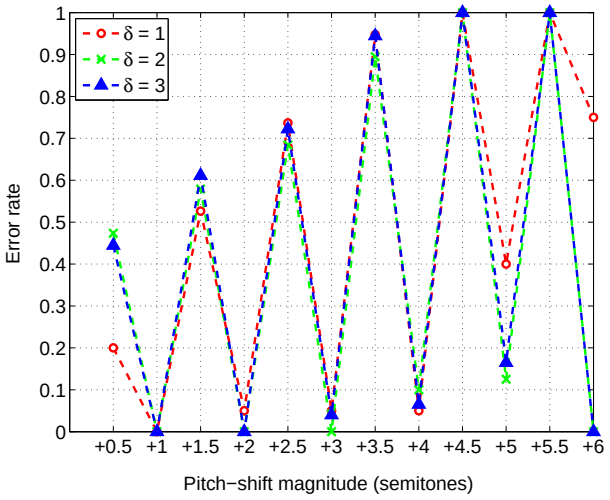


Fig. 13. Error rate as a function of the pitch-shift and δ ($b = 16$).

We conclude this section by noting that, if both the time and frequency scales are linearly stretched up or down by a few percents, an effective workaround has been presented in [24]. Its main advantage – in contrast to our approach – is a very good handling of these distortions with a fine granularity, at least within its range of use.

Unfortunately, it has also many drawbacks. In the first place, it requires substantial modification of the algorithm, rendering any existing databases completely useless, which is unacceptable in many cases, e.g. because of the unavailability of the source songs. Secondly, the running time of the new algorithm is appreciably longer, since it requires – in addition – as much as two Fourier transforms (implementing autocorrelation), a low-pass filter and a downsampling (this could not be an issue if the constraints on the search time are not particularly tight).

The authors locate the need of their method especially in the recognizing of radio broadcasts, where time constraints may indeed lead to speeding up songs. In these cases, however, the speed up is limited and involves only the time scale, a situation which is well handled by the large-overlap framing in the existing algorithm. We conducted many experiments in radio broadcasts with results entirely comparable to the non-distorted case (i.e. error-free). Nevertheless, the approach could prove useful in certain contexts.

B. Error rate

A number of random songs was selected among those existing in the database and manually pitched using the open source tool *Audacity* [39]. The magnitude of the distortion varies from $+0.5$ to $+6$ semitones, with a step of 0.5 semitones, involving an approximate percent frequency displacement summarized by table I. On the basis of our previous discussion, a 12-semitone pitch shift, i.e. an octave, would correspond to a 100% frequency shift. For reasons of symmetry emerged in a previous work [37], we do not consider here negative pitch shifts.

TABLE I
PITCHING/FREQUENCY-DISPLACEMENT APPROXIMATE MAPPING.

$+0.5$	$+1$	$+1.5$	$+2$
$+2.93\%$	$+5.95\%$	$+9.05\%$	$+12.24\%$
$+2.5$	$+3$	$+3.5$	$+4$
$+15.54\%$	$+18.92\%$	$+22.41\%$	$+25.99\%$
$+4.5$	$+5$	$+4.5$	$+6$
$+29.68\%$	$+33.48\%$	$+37.40\%$	$+41.42\%$

Let's first focus on the fingerprint-shift search strategy. As we can see in Fig. 12, a careful choice of the sub-bands may be essential, also depending on the particular value of b used. For $b = 16$ and EQL, the behavior alternates between an excellent success rate (around distortions of an even number of semitones) and a very poor one. Similarly, its 12-TET counterpart is heavily troubled by half-semitone distortions. When $b = 32$, the proposed search strategy is instead thoroughly effective, regardless of the particular band division. The same can be observed in the following Fig. 15

On the contrary, if we had a viable "de-pitch" approach, it would lead to perfect results in terms of success rate. This would be comparable to the undistorted case, with the only drawback of a slightly increased distance. According to our tests, this increase would be at most in the neighborhood of an additional $5 - 6\%$.

If the exact pitch shift is not known or guessed, indeed, we are exactly in the same case of having a pitch-shifted excerpt and employing the basic search strategy. That is, without a very good estimation of the pitch shift involved, or without enough (and computationally expensive) trials and errors, the "de-pitch" solution is quite useless. A concerted strategy could be thought of, but at a great expense in terms of search time. From now on, only the bit-shift approach will be brought into discussion.

When $b = 16$, it is possible to further reduce the error rate by employing the strategy described in IV-D and filtering out the shortest runs in the fingerprints, prior to the distance computation. For clarity, we report here the results up to a value of $\delta = 3$. As we can see from Fig. 13, $\delta = 2$ appears a well-balanced choice, with respect to the error rate metric.

C. Computed distance

Since we use a threshold mechanism for telling correct matches apart, a most important metric to investigate is the average distance computed in case of success or error. Indeed, both can help in fixing (a) reliability threshold(s) and are a soft indication of the overall robustness of the recognizer. In Figs. 14 for $b = 16$ and 15 for $b = 32$, we observe that the distance in case of success tends to increase with the magnitude of the distortion, with more or less a constant penalty when half-semitone distortions occur. The lower values for the high pitch shift of $+6$ semitones may be due to the particular distribution of most of the erroneous bits when pitch shifting is involved.

Introducing $\delta > 1$ does help in lowering the evaluated distance with respect to the correct match. This is already clear in the presented figures, so we did not report

the dependence of the metric from δ itself, which is constituted by slowly-decreasing, quasi-parallel straight lines, as δ grows.

We point out that $\delta = 2$ can be regarded as very appropriate, since it reduces both the distance with respect to the match and the overall error rate. The only, minor drawback is a slightly reduced distance in case of error (should they occur): this could potentially lead the computed distance close to the threshold used for discriminating whether or not we have a match.

VII. Performance under AWGN

Additive white gaussian noise (AWGN), or thermal noise, is very common and worth investigating as far as the performance of the system is concerned. In a previous work [30], we found that $b = 16$ could in this case lead to better results. However, the database used was orders of magnitude smaller than the actual one, and the cross-correlation search method described in III-D was employed.

Carrying on similar experiments on a much wider database and cautiously relying on the brute-force search, we obtained the curves in Fig. 16, where we represent the average error rate with respect to the Signal-to-Noise Ratio (SNR) of the excerpt. The excerpts were randomly chosen from the whole database and then conveniently corrupted on the fly.

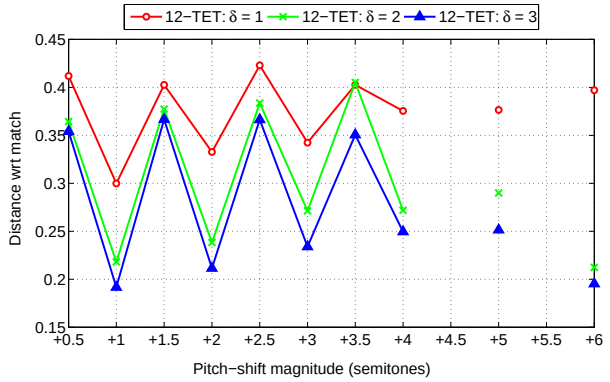


Fig. 14. Given $b = 16$, normalized Hamming distance against the correct match as a function of both the pitch-shift magnitude and δ. The distances for $+4.5$ and $+5.5$ are missing because no success was recorded.

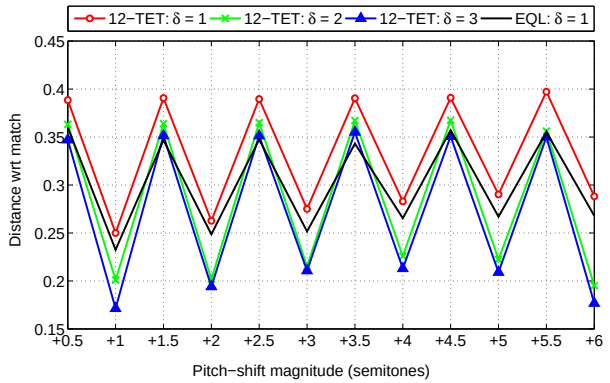


Fig. 15. Given $b = 32$, normalized Hamming distance against the correct match as a function of both the pitch-shift magnitude and δ.

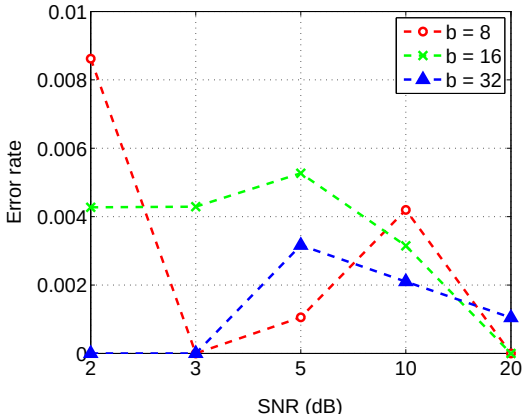


Fig. 16. Error rate for three different choices of b.

The choice $b = 16$ may actually pay off in certain circumstances, but it emerges that $b = 32$ is actually better in a wider range of cases. Moreover, as we will soon see in the next section, $b = 32$ is almost a mandatory choice if we aim at reasonable search times.

Nevertheless, it is interesting to note that $b = 16$ would lead to a noticeably smaller distance between the unknown excerpt and its match, with respect to the case $b = 32$. In addition, should matching errors occur, they will show up with a higher distance than the case $b = 32$, as can be seen in Fig. 17. On the basis of these considerations, we believe the choice $b = 16$ could be acceptable and rewarding, if search times are not the primary concern.

This last figure also suggests the possibility of increasing the tolerable threshold for declaring a match, in order to adopt a more aggressive strategy and trying to avoid false negatives. We believe that a more rewarding choice is instead adopting a two-threshold mechanism, associating to them different reliability indicators. For example, a distance metric below a threshold $T_1 = 0.30$ can be taken as a safe match, while a value between T_1 and $T_2 = 0.35$ should be considered less reliable. From this point of view, the post-processing strategy described in IV-G can be really of help, as well as the tunable threshold hinted in III-C.

VIII. Timing performance

The results presented in the following sections, valid in accordance to the specifications in V, can be of great help when it comes to design the system. In particular, our considerations investigate its scalability and definitely show the need of operating as much as possible in RAM and having $b = 32$.

A. Indexing

First of all, let's focus on Fig. 18, where we report both the absolute search time and the number of comparisons effectively made (i.e. the number of alignment locations

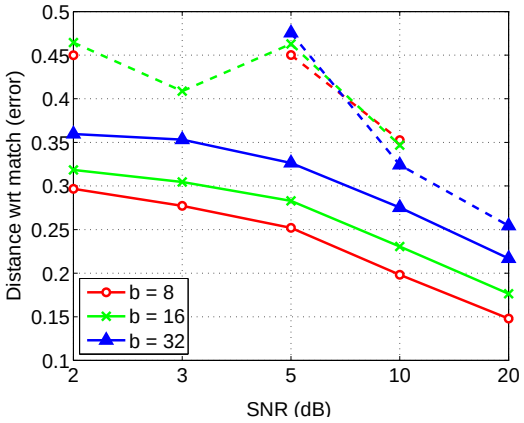


Fig. 17. In the continuous lines, we depict the distance with respect to match for three different choices of b. Their respective counterpart in case of error, when errors occur, is represented by the dashed lines.

tried in the database). Both non-cached and fully-cached indexing are reported, while we discuss intermediate values later in VIII-B. The most striking outcome is that a sub-database fully loaded on memory (i.e. a 100% cache) leads to a search time which is always well beyond an order of magnitude faster than the approach without cache, ranging from 370 s to 2.1 s for $k = 16$ (gain of about 176), and from 2.92 s to less than 0.04 s for $k = 26$ (gain of about 73).

As expected, a smaller number of comparisons (from 1 000 000 for $k = 16$ to 20 000 for $k = 26$) leads to a significantly smaller computation time (roughly linear with it), but it is interesting to note that only the full-cache approach can follow the trend of the made comparisons. In particular, the decreasing trend of the no-cache curve soon slows down, as enhanced by the logarithmic scale. This could be due to the constant and unavoidable hard disk seeking time, whose impact is the more noticeable, the fewer readings are performed. The average number of comparisons is about the same in both cases (i.e. with or without cache), with only tiny fluctuations.

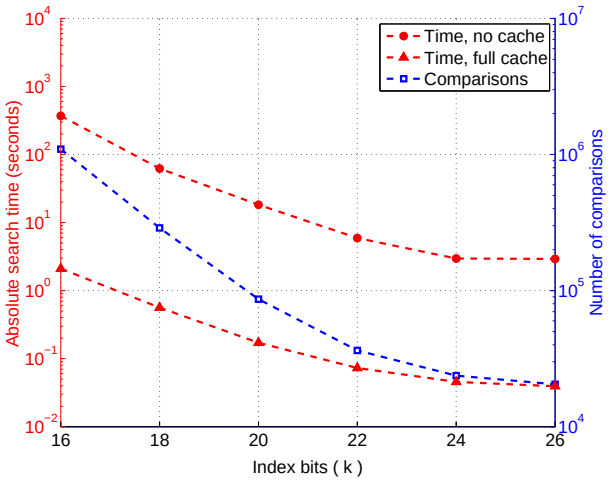


Fig. 18. Absolute search time as a function of the index parameter k and highlighting the number of actual comparisons made.

TABLE II
INDEX REQUIREMENTS IN TERMS OF STORAGE AND LOADING TIME, AS A FUNCTION OF k. NOTE THAT THE TYPICAL SOURCE SUB-DATABASE SIZE IS, FOR US, 846 MIB.

k	Storage (MiB)	Loading time (s)
16	802	14.84
18	810	15.33
20	840	17.14
22	960	22.87
24	1440	39.86
26	3360	86.64

From the presented results, it is clear that $k = 16$ does not lead to a very good performance, in comparison with higher values, since the associated index still exhibits too many candidate positions. Indeed, if we had a uniform pattern distribution in our 10 000-entry sub-databases, there would be about $2.22 \cdot 10^8/2^k$ candidate positions for each feature vector of the unknown fingerprint, thus suggesting to take k as large as possible. If we would have chosen $b = 16$, no value of k beyond 16 would have been acceptable. Taking two feature vectors at a time did not prove reliable at all, while an intermediate value of b between 16 and 32 would have lost the very practical 32-bit alignment. We pay the doubling of b in terms of doubling not only the database size, and this is generally not troublesome, but also the time required for a single comparison. Nevertheless, the dramatic reduction in the number of comparisons is enough to provide very good search times.

B. Caching

An important merit figure is the ratio between the actual playing time of the excerpt and the search time, an essential parameter for an effective, real-time operation. We report the results in Fig. 19, which is but 5 divided by the absolute times of Fig. 18. In the case $k = 24$, we have a value of about 110 in the full-cache case, compared to a bare 1.7 if we have to read every time from the storage disk. To be precise, the location list is ordered, then we could benefit from a one-entry cache, if a pattern shows more than once in a single entry, but the gain is really negligible.

For a more insightful discussion, table II shows the actual index sizes and their associated loading time, for the tested values of k. Both measures linearly scales with the sub-database size, but the index size may easily grow to 1.7 ($k = 24$) or even 4 ($k = 26$) times the associated sub-database. Given the actual cheap cost of data storage, this should not be an issue. The loading time, also, is not worrisome, since it must typically be performed only once, when the recognizing software is started. Note also that, speaking of single sub-databases and without index, the size gain factor of their 846 MiB over the corresponding PCM material (44 100 Hz, 16 bit, stereo) is 512, which is not very high. If we add the size of the index built for $k = 24$, the gain decreases to a mere 190 (we suppose that no compaction algorithm

is applied). When interested in mobile applications, we should be aware of this issue.

In Fig. 20 we report the impact of the cache size on the absolute and relative search time. On the basis of the previous discussion, k is set to 24, so to allow also for a fully-cached indexing. We see that the implemented cache must not be too small, in order not to worsen the performance. On the contrary, it is beneficial if it can hold at least half of the stored references. If the slow disk access is completely avoided, then – as seen above – we achieve a good gain of more than 100 over real time. It is probably possible to improve the implementation efficiency and/or the replacement policy, but we would expect similar trends.

C. Comparison with brute-force approach

We highlight here the benefits, in terms of search time, of both indexing and then caching over the trivial brute-force approach, as a function of the sub-database size, thus providing scalability hints. For the stated reasons, we again set $k = 24$. The exhaustive search is performed after the sub-database is loaded in memory in its entirety, and very long times (unreported here) are required if this condition is not met.

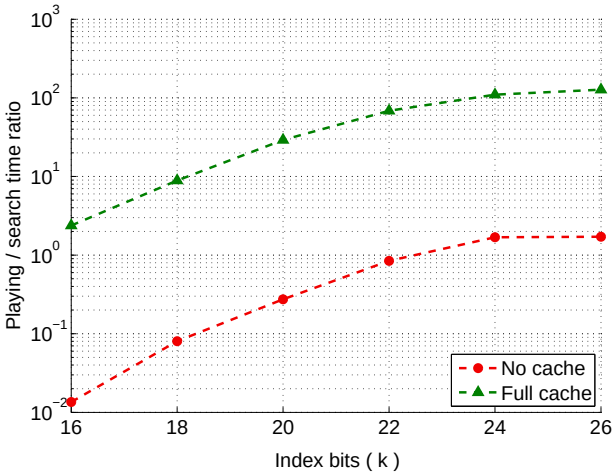


Fig. 19. Ratio between search time and playing time of the excerpt, as a function of the index parameter k.

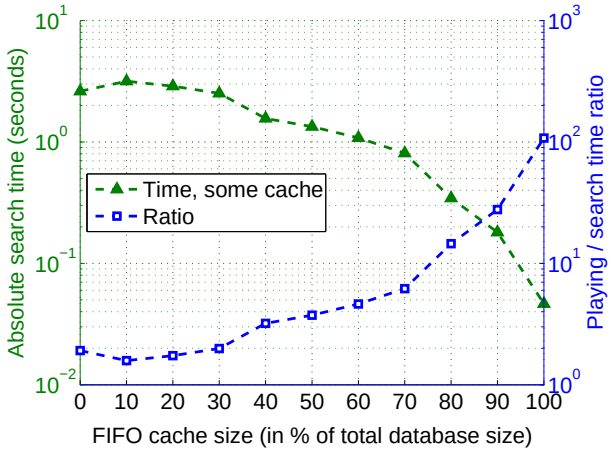


Fig. 20. Efficiency of the cached indexing approach, in terms of absolute and relative search time, as a function of the cache size.

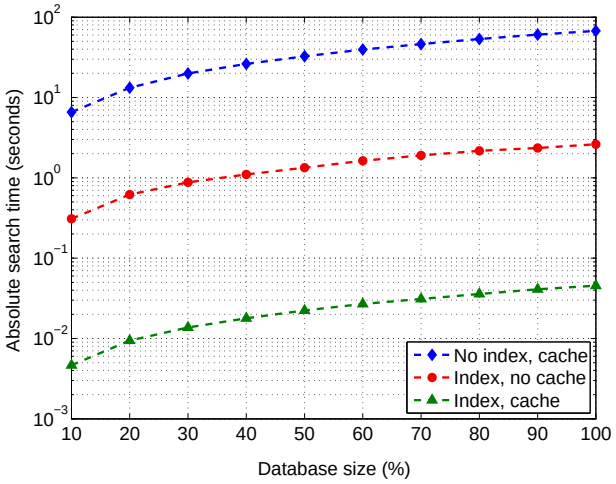


Fig. 21. Absolute search time as a function of the database size and comparing exhaustive search (which has database on memory), indexing and fully-cached indexing. The index parameter k is 24.

By looking at Fig. 21, we already notice a good improvement if we just index (thus accessing the disk at each comparison), with a time gain factor that ranges from about 21 for 1000 database entries, growing to over 25 when there are 10 000 of them. First of all, as seen, indexing leads to a reduced number of comparisons, which is in the order of $2 \cdot 10^8$ for brute force (this value is lower than the number of stored feature vectors because some of alignments at the end of each entry are actually not tried, since they would not allow a full distance evaluation with the unknown excerpt). In the second place, we believe that the growing trend of the gain may be well justified by the fact that fewer comparisons imply fewer disk accesses, which constitutes the real bottleneck, so their impact is progressively reduced.

A further, and significant, performance boost can come from the use of caching. If we load on memory all the entries that will be involved in comparisons, then the gain factor over brute-force is constant for every database size and evaluated in over $1,500$. The steadiness of the value well reflects the overcoming of the disk bottleneck and constitutes a valid indication for evaluating the benefit of the presented indexing scheme. On the other hand, the gain factor with the respect to the number of comparisons is around $8,500$ (see again Fig. 18 when $k = 24$).

Finally, Fig. 22 focuses instead on the ratio between playing and search time. The order of the curves is of course the opposite of that found in Fig. 21, and the full- and no-cache approaches may be compared with Fig. 19 when $k = 24$. Considering the whole sub-database, we observe a gain of almost 110 for the cached-index approach, in comparison to a value of about 2 if no cache is present and less than 0.1 when no index is

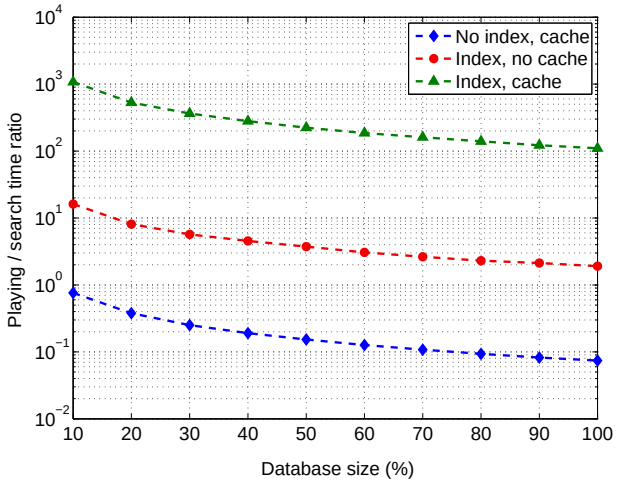


Fig. 22. Ratio between search time and playing time of the excerpt, as a function of the database size and comparing exhaustive search (which has database on memory), indexing and fully-cached indexing. The index parameter k is 24.

present, i.e. the search time is a hundred times longer than the playing time.

IX. Conclusion

Building on a well-known fingerprint algorithm [5], we developed and discussed a more comprehensive framework for automatic audio recognition. We mainly supported our considerations and suggestions on the basis of careful experiments conducted on real and ample data, i.e. a 100 000-entry fingerprint database.

We would like to highlight that databases of this size are not frequently seen in the scientific literature, although there are notable exceptions in the commercial field, like the popular *Shazam*. A second important aspect is the *efficiency* of our implementation: although still quite demanding in terms in computing power, it achieves an appreciable performance, as discussed in the past sections.

A number of effective, and possibly efficient, strategies for selecting good algorithm parameters and for improving the overall performance of the system, has also been proposed. Special emphasis is put on the broadcast monitoring application, which also represents a growing market.

We believe the discussion we provided can help to a good extent not only the understanding of the topic, but also the design and use of such systems. Further improvements and refinements are yet needed for a less resource-demanding solution.

Acknowledgments

We would like to thank the editor and the anonymous reviewers, whose relevant remarks really helped improving the quality and the readability of the paper. We are also very grateful to Knowmark s.r.l. [40] for its technical and financial support.

References

[1] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *The Journal of VLSI Signal Processing*, vol. 41, no. 3, pp. 271–284, 2005.

[2] R. Typke, F. Wiering, and R. Veltkamp, "A survey of music information retrieval systems," in *Proceedings of the 6th International Conference on Music Information Retrieval*, 2005.

[3] P. Cano, E. Batlle, H. Mayer, and H. Neuschmied, "Robust sound modeling for song detection in broadcast audio," *Proceedings of the 112th AES Convention*, pp. 1–7, 2002.

[4] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *International Workshop on Content-Based Multimedia Indexing*. Citeseer, 2001.

[5] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system with an efficient search strategy," *Journal of New Music Research*, vol. 32, no. 2, pp. 211–221, 2003.

[6] H. Özer, B. Sankur, N. Memon, and E. Anarim, "Perceptual audio hashing functions," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 1, pp. 1780–1793, 2005.

[7] D. Jang, S. Lee, J. Lee, M. Jin, J. Seo, S. Lee, and C. Yoo, "Automatic Commercial Monitoring for TV Broadcasting Using Audio Fingerprinting," in *Proceedings of the 29th Audio Engineering Society Conference*, 2006, pp. 38–43.

[8] C. Burges, D. Plastina, J. Platt, E. Renshaw, and H. Malvar, "Using audio fingerprinting for duplicate detection and thumbnail generation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 2005.

[9] A. Ghias, J. Logan, D. Chamberlin, and B. Smith, "Query by humming: musical information retrieval in an audio database," in *Proceedings of the third ACM international conference on Multimedia*. ACM New York, NY, USA, 1995, pp. 231–236.

[10] Melodis Corporation, "midomi," http://www.midormi.com, 2008.

[11] V. Venkatachalam, L. Cazzanti, N. Dhillon, M. Wells, M. Inc, and W. Kirkland, "Automatic identification of sound recordings," *IEEE Signal Processing Magazine*, vol. 21, no. 2, pp. 92–99, 2004.

[12] C. Burges, J. Platt, and S. Jana, "Extracting noise-robust features from audio data," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2002*, vol. 1. Citeseer, 2002.

[13] A. Ramalingam, "Gaussian mixture modeling of short-time Fourier transform features for audio fingerprinting," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 4, pp. 457–463, 2006.

[14] C. Burges, J. Platt, and S. Jana, "Distortion discriminant analysis for audio fingerprinting," *IEEE Transactions on Speech and Audio Processing*, vol. 11, no. 3, pp. 165–174, 2003.

[15] L. Shen, Y. Guan, Y. Wu, and Y. Zhao, "Fast audio fingerprint search strategy for song identification," *Networking and Digital Society, International Conference on*, vol. 2, pp. 259–262, 2009.

[16] N. Orio, "Automatic identification of audio recordings based on statistical modeling," *Signal Processing (Elsevier)*, 2010.

[17] S. Baluja and M. Covell, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern Recognition (Elsevier)*, vol. 41, no. 11, pp. 3467–3480, 2008.

[18] S. Sukittanon and L. Atlas, "Modulation frequency features for audio fingerprinting," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 2002.

[19] L. Ghouti and A. Bouridane, "A robust perceptual audio hashing using balanced multiwavelets," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, May 2006.

[20] Y. Liu, K. Cho, H. S. Yun, J. W. Shin, and N. S. Kim, "Dct based multiple hashing technique for robust audio fingerprinting," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 0, pp. 61–64, 2009.

[21] F. Kurth and M. Clausen, "Full-text indexing of very large audio data bases," in *Proceedings of the 110th AES Convention*. Citeseer, 2001.

[22] F. Kurth, A. Ribbrock, and M. Clausen, "Identification of Highly Distorted Audio Material for Querying Large Scale Data Bases," in *Proceedings of the 112th AES Convention*. Citeseer, 2002.

[23] J. Goldstein, J. Platt, and C. Burges, "Redundant bit vectors for quickly searching high-dimensional regions," *Lecture notes in computer science (Springer)*, vol. 3635, p. 137, 2005.

[24] J. Haitsma and T. Kalker, "Speed-change resistant audio fingerprinting using auto-correlation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, 2003.

[25] F. Balado, N. Hurley, E. McCarthy, and G. Silvestre, "Performance analysis of robust audio hashing," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 2, pp. 254–266, 2007.
[26] P. Doets and R. Lagendijk, "Distortion Estimation in Compressed Music Using Only Audio Fingerprints," *IEEE Transactions on Audio Speech and Language Processing*, vol. 16, no. 2, p. 302, 2008.
[27] A. Wang, "An industrial strength audio search algorithm," in *Proceedings of the 4th International Conference on Music Information Retrieval*, 2003.
[28] S. Kim and C. Yoo, "Boosted binary audio fingerprint based on spectral subband moments," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, April 2007, pp. I–241–I–244.
[29] D. Jang and C. D. Yoo, "Fingerprint matching based on distance metric learning," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 0, pp. 1529–1532, 2009.
[30] C. Bellettini and G. Mazzini, "On audio recognition performance via robust hashing," in *International Symposium on Intelligent Signal Processing and Communication Systems*, 2007, pp. 20–23.
[31] T. Wang, "Integer hash function," http://burtleburtle.net/bob/hash/integer.html, March 2007.
[32] R. Jenkins, "Hash table lookup," http://burtleburtle.net/bob/c/lookup3.c, May 2006.
[33] T. Li, M. Ogihara, and Q. Li, "A comparative study on content-based music genre classification," in *Proceedings of the 26th International Conference on Research and Development in Information Retrieval*. ACM New York, NY, USA, 2003, pp. 282–289.
[34] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.
[35] F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano, "An experimental comparison of audio tempo induction algorithms," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1832–1844, 2006.
[36] C. Panagiotakis and G. Tziritas, "A speech/music discriminator based on RMS and zero-crossings," *IEEE Transactions on Multimedia*, vol. 7, no. 1, pp. 155–166, 2005.
[37] C. Bellettini and G. Mazzini, "Reliable Automatic Recognition for Pitch-Shifted Audio," in *Proceedings of 17th International Conference on Computer Communications and Networks*, 2008, pp. 1–6.
[38] Bernsee, "Pitch shifting using the fourier transform," http://www.dspdimension.com/admin/pitch-shifting-using-the-ft/, May 2008.
[39] Audacity, http://audacity.sourceforge.net/.
[40] Knowmark s.r.l., http://www.knowmark.it.

Carlo Bellettini was born in Portomaggiore (FE), Italy, on May 13th, 1982. He received his BSc (summa cum laude) and his MSc (summa cum laude and distinction) degrees in Telecommunications Engineering from the University of Ferrara, Italy, in 2004 and 2006 respectively. His Master's thesis was done at Telecom Italia Lab (TILAB), in Turin, Italy, where he worked within the European Community project ENABLE. He was visiting research scholar to UCLA Network Research Lab (NRL), Los Angeles, CA, USA from March to September 2009. Since 2007, he has been part of the Telecommunications group at the Engineering Department of Ferrara (ENDIF), where he is pursuing his PhD in Telecommunications Engineering.

He has been graduate student member of IEEE since 2007 and he served as reviewer for many IEEE conferences. His main research focuses are on telecommunications networks and signal processing. He is or has been teaching assistant at the University of Ferrara for the courses of Signal Theory; Electrical Communications; and Telecommunications Networks. He also teaches there the course of Digital Signal Processing.

Gianluca Mazzini was born in Bologna, Italy, on January 3, 1968. He received the Laurea degree in Electronic Engineering (with honors) and the Ph.D. in Electronic Engineering and Computer Science from the University of Bologna, Bologna, Italy, in 1992 and 1996, respectively. In 1996 he joined the University of Ferrara, Italy, where he currently is an Associate Professor of Telecommunications. He teaches Telecommunications Networks; Internet and Wireless Systems; Telecommunications Network Laboratory; Internet Security; Digital Transmission Systems; and Multimedia Communications. His research interests are related to: spread spectrum communications; application of chaotic system to telecommunications; non-linear dynamical system modeling; next generation of cellular/ambient systems; wireless LAN architectures, routing and protocols; sensor networks; Internet mobile computing; and routing and security.

He served as technical organizer for NDES2000 and NOLTA2006 conferences and he was editor of the IEEE Proceeding May 2002 special issue on "Applications of Nonlinear Dynamics to Electronic and Information Engineering". Since 2002 he is an Associate Editor for IEEE Communications Letters. He is author or co-author of more than 200 papers. He is an IEEE senior member.

A description of his research group in the University of Ferrara is on www.tlc.unife.it. He is also involved in the Electronic System for Information and Communication Technologies [ARCES] research group in the University of Bologna (www.arces.unibo.it), in the National Inter-University Consortium for Telecommunications [CNIT] (www.cnit.it), in the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni [IEIIT-CNR] (www.ieiit.cnr.it).