

Scalability Issues in Fingerprinted Audio Searching

C. Bellettini and G. Mazzini[†]

ENDIF – University of Ferrara

Email: {carlo.bellettini, gianluca.mazzini}@unife.it

Abstract—This paper discusses important scalability issues exhibited by the high-dimensional, automatic audio recognition problem. The emphasis is especially put on a well-known, robust fingerprint algorithm. Extensive tests on a very large database show the importance of the parametrization of the algorithm, in order to afford an efficient search strategy. We also quantitatively verify the capital attention that should be devoted to the caching issue, in order to avoid disk access as much as possible. Comparisons and insights are provided also with the respect to the brute-force approach.

Index Terms—Audio fingerprinting, features extraction, audio indexing, scalability, music information retrieval.

I. INTRODUCTION

The availability of more computing power at steadily decreasing costs certainly plays a key role in the current, widespread interest in many applications related to signal processing, among which we find automatic audio recognition. Though related to speech recognition, this truly multiform and challenging topic is actually addressed in other ways. In this paper, we work only on the waveform representation of the signal, without any further semantic provided.

The possible approaches are manifold, heavily depending on the researcher’s background and goals, but the first step invariably involves isolating a sequence of “features” in a piece of audio, as more as longer is the piece. This set of features is said to be the “fingerprint” of the piece: in order to be useful, it must be both fast to compute from e.g. a PCM (*Pulse Code Modulation*) signal, and sufficiently distinguishing, so that two fingerprints can be reliably told apart or regarded as similar. It should also achieve a significant reducing factor in size (roughly ≥ 1000) with respect to the original PCM samples.

While solo instrumental music may find a good solution in separating single notes composing a piece [1], its employment limitations are often unacceptable. For the more general case, different audio features have been proposed in the literature [2], usually chosen on the basis of heuristic considerations, with a few exceptions [3]. Of particular interest is the possibility of combining more of them into one single identification model [4], although at the moment it would require too expensive computation times.

As will be shown, the usual system sees the building of a large fingerprint database (tenth of thousandth entries, for practical uses), 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. Moreover, the fingerprint and search algorithm must prove robust, i.e. exhibit a high reliability even when various kinds of distortions occur, such as equalizing, white noise [5], pitching [6] and so on.

The tight search time requirements (a gain factor of roughly 100 should generally be attained over real time) and its high complexity also led to put considerable effort into the matching problem [7]. In [8], 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.

As to applications [9] [10], it is easy to see that such a framework may prove effective in numerous scenarios, ranging from metatagging digital music, to filling in a list of songs recorded from the radio or in open space, to spotting ill-used copyrighted music.

[†]We are both grateful to Knowmark s.r.l. [12], for their financial and technical support and their valuable suggestions.

II. SYSTEM ARCHITECTURE AND ALGORITHMS

The usual high-level perspective is summarizable as in Fig. 1. Blocks FA and SA stand for “fingerprint” and “search” algorithm respectively, while block DB is the reference database. For this work, we processed through FA an MP3 library comprising 10.000 pieces, in less than 20 seconds each (Pentium 4 @ 2.3 GHz). This lengthy, off-line stage must be performed only once, at least until FA does not change.

The everyday use of the system is pictured in the lower branch. Some source provides the piece of audio we wish to identify and whose duration can range from a few seconds to many hours, depending on the particular application. In any case, 3 or 4 seconds are usually enough to perform a reliable identification. Through SA, we try to match (a part of) the fingerprint of the unknown piece of audio (computed again with FA) against the pre-built database.

A. Fingerprinting

For FA we focused our attention on a simple yet very robust algorithm, whose operation was first described by Haitsma *et al.* in [11] and discussed also in [5]. The feature it extracts is intimately related to the audio signal energy. High robustness apart, this algorithm proves attractive also for the convenient output it produces (a bit matrix) and for its flexibility.

The input is assumed to be mono PCM samples @ 44.100kHz and 16bit: this may imply one or more preliminary conversion stages, depending on the particular source available. A STFT is then computed, applying a Hann window whose duration is about 372ms and with an overlap between frames slightly above 96%. For each temporal segment n of the N available, a band meaningful for the human ear (up to 2kHz) is divided up into $b+1$ sub-bands, each of them responsible for a computed energy contribution $E_{n,m}$ where $0 \leq m \leq b$ is the sub-band. The choice of sub-bands follows [6] (thus made according to the 12-TET tuning system), while the choice $b = 32$ will be cleared later in III-A. A vector of b bits is then computed according to the rule:

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

where $F_{n,m}$ (for which $0 \leq m \leq (b-1)$) is the m -th bit relative to frame n and α is the energy difference among contiguous sub-bands, namely $\alpha_{n,m} = E_{n,m} - E_{n,m+1}$. The whole input is processed one

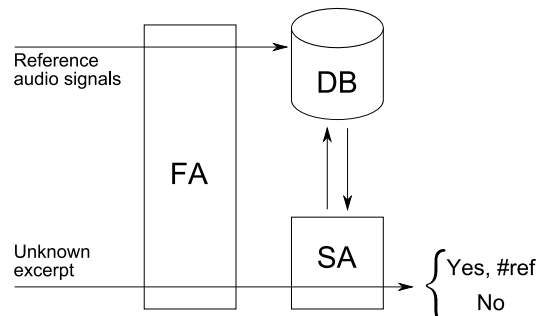


Fig. 1. Scheme of the audio recognizing system.

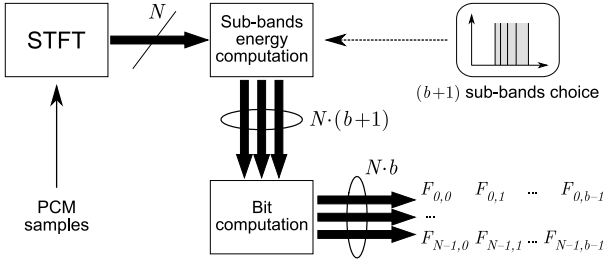


Fig. 2. The fingerprint algorithm (FA).

segment at a time and eventually we obtain a sequence of bit vectors, which is the actual fingerprint of the given piece of audio.

B. Matching

In order to declare a match between the fingerprint of the unknown excerpt and one or more stored references, some distance metric must be defined: the bit matrix form induces to use the normalized Hamming distance. If an exhaustive search is performed, the fingerprint to be identified is “slid” along all reference fingerprints, evaluating the distance for each possible alignment. The lowest outcome may then be proposed as match. To take into account the case of a missing match, however, a distance threshold $T = 0.35$ is defined, providing a hard decision on the reliability of the result. This value may be tuned to trade off false positives against false negatives, and is heuristically chosen [11]. We verified it is almost always appropriate.

It is clear that the brute-force approach, though highly effective (if not error free, with good-quality inputs), is not at all efficient, requiring an extremely high number of comparisons, too high even for a high-end PC to be performed in real time. With the proposed parameters, we extract about 5168 4-byte feature vectors per playing minute. If the average song length in our 10 000-song database in MM:SS is 4:18, then we have a total storage of 846MiB, or 87KiB per track, and $2.22 \cdot 10^8$ feature vectors stored. Even with an optimized implementation, comparing even a very short excerpt with as many possible alignments proves feasible only in relatively large times, linear with the database size, as will be shown.

Following the rationale suggested in [11], we thus implemented a more practical solution. If we suppose that the unknown fingerprint exhibits at least one feature vector exactly alike one belonging to the correct matching song, then it makes sense to build a database index. Ideally, it would list all possible patterns, along with their location in the database. Querying the index for the patterns computed from the unknown excerpt will give a number of promising alignments, thus drastically reducing the needed comparisons.

In order to achieve the desired efficiency, the index must reside in the fast, volatile memory (RAM). Since a 2^{32} -row table would largely exceed the standard RAM equipment of modern PCs, we in fact resorted to a shorter lookup table, using a number (L) of the least significant bits of the pattern as the search key. Values from $L = 16$ to $L = 26$ at step of 2 have been tried. In our current implementation, the pattern is first hashed with a general and very fast 32bit-to-32bit function, which alleviates to a great extent the issue of the strong non-uniformity of the pattern distribution. In other words, the key used for building and then querying the index is computed as $h(x) \otimes (2^L - 1)$, where h is the hash function, x the input pattern and \otimes denotes the bitwise *and* operator. We also point out that, from the hash table point of view, the location list associated with each key could also be seen as the *separate chaining* collision resolution method.

C. Caching

In III, will be quantitatively justified the strong need to operate as much as possible in RAM. In order to better drive implementation

decisions, a variable-size cache was implemented. Although there may exist more refined replacement policy, we used a simple FIFO (First In, First Out) ring buffer. Each time that part of an entry is needed, it is loaded in its entirety in the first available bin, for future reference. 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 step of 10%.

III. DISCUSSION OF EXPERIMENTAL RESULTS

In all the following considerations, a random excerpt of length 3.333s (which equals 256 feature vectors) is randomly chosen from a random song whose fingerprint is in the database, iterating at least 500 times for each point, with sufficiently smooth outcomes. Even if in this work we are not investigating the reliability of the matching algorithm, it is worth noting that we have in all cases a success rate close to 95% (the excerpts are undistorted), meaning that only for a small fraction of the trials the song was not correctly identified. In these cases, though, the computed distance was always above the fixed threshold $T = 0.35$, thus leading to a false negative, rather than to a false positive, which is often regarded as positive. For the sake of comparison, the brute-force approach is virtually error free, with a success rate in excess of 99.9%. The simple but effective approach to improve reliability discussed in [11] is currently under study, together with possible alternative methods.

A. Indexing

First of all, let’s focus on Fig. 3, where we report both the absolute search time and the number of comparisons effectively made (i.e. the number of alignment locations tried in the database). We used C++ on a standard home PC, with clock @ 2.3GHz and 2GiB of RAM. Both non-cached and fully-cached indexing are reported, while we discuss intermediate values in III-B. The most striking outcome is that a 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 73.39s to 1.84s for $L = 16$ (gain of about 40), and from 0.59s to less than 0.03s for $L = 24$ (gain of about 20). Due to memory limitations, it was not possible to evaluate the trend for both $L = 26$ and a 10 000-entry cache.

As expected, a lower number of comparisons (from 813 500 for $L = 16$ to 8 629 for $L = 24$) leads to a significantly inferior 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.

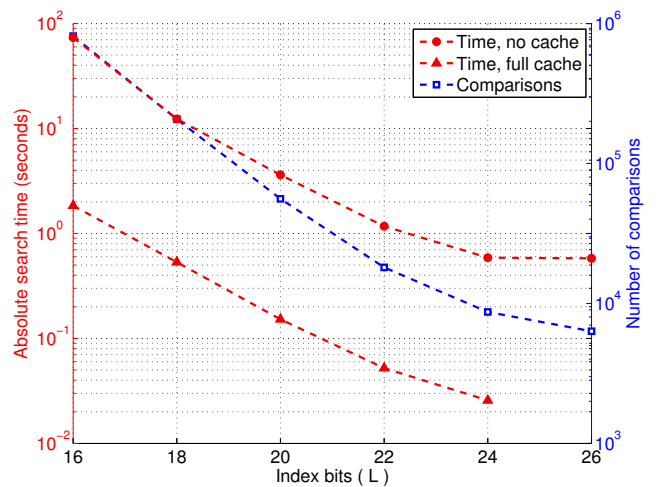


Fig. 3. Absolute search time as a function of the index parameter L and highlighting the number of actual comparisons made. A point is missing for lack of memory.

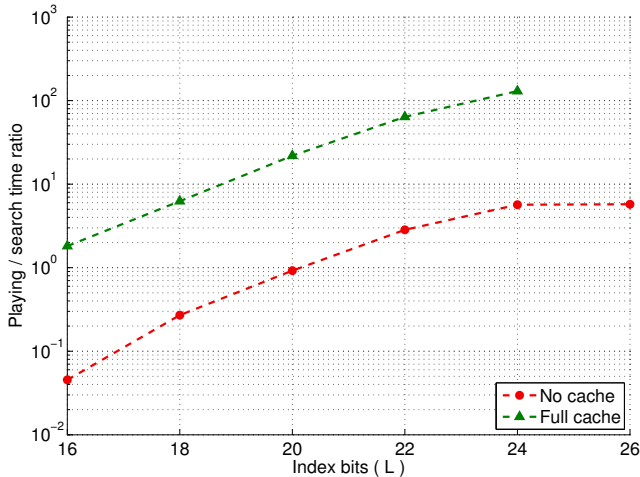


Fig. 4. Ratio between search time and playing time of the excerpt, as a function of the index parameter L . A point is missing for lack of memory.

In particular, the decreasing of the no-cache curve soon slows down, as enhanced by the logarithmic scale. This could be due the fixed, unavoidable seeking time associated with the reading from the hard disk, relatively as more relevant as the less 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.

We can now justify the choice $b = 32$, in contrast to our previous works [5] [6], where we took $b = 16$. From the presented results, it is clear that $L = 16$ 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 database, there would be about $2.22 \cdot 10^8 / 2^L$ candidate positions for each feature vector of the unknown fingerprint, thus suggesting to take L as large as possible. If we would have chosen $b = 16$, no value of L 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 (see III-C) 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. 4, which is nothing more than 3.333 divided by the absolute times of Fig. 3. In the case $L = 24$, we have a value of about 130 in the full-cache case, compared to a bare 6 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 I shows the actual index sizes and their associated loading time, for the tested values of L . Both measures linearly scales with the database size, but the index size may easily grow to 1.7 ($L = 24$) or even 4 ($L = 26$) times the associated 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 be performed only once, at the beginning. Note also that, without index, the size gain factor of the 846MiB over the corresponding PCM material (44.100Hz, 16bit, stereo) is 512, which is not very high. If we add the size of the index built for $L = 24$, the gain decreases to a

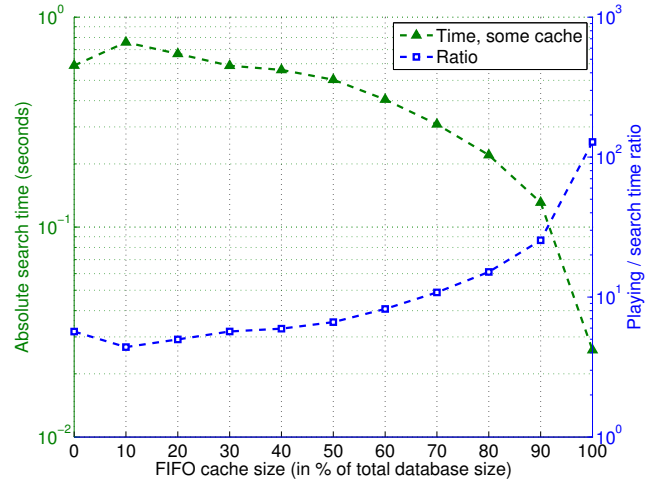


Fig. 5. Efficiency of the cached indexing approach, in terms of absolute and relative search time, as a function of the cache size.

mere 190 (we suppose that no compaction algorithm is applied). One interested in mobile applications should be aware of this issue.

On the other hand, there may be scalability issues with larger databases. In this work, ours is made of 10 000 entries, which are not that few, but are not certainly enough for a commercial employment, which would require around 10 times more. If we are not to have a huge RAM disk available, as is often the case, loading only a fraction of the index is clearly needed. However, from the data in table I, we see that either a more optimized index structure must be planned, or a completely different approach must be sought. As for the latter, the ideas presented in [7] and [8], though much more complex, have the virtues of being sufficiently general and might be good candidates to optimize for speed, besides their very good robustness.

In Fig. 5 we report the impact of the cache size on the absolute and relative search time. On the basis of the previous discussion, L 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 worsening 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 we achieve a very good gain of almost 130 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 database size, thus providing scalability hints. For the stated reasons, we again set $L = 24$. The exhaustive search is performed after the database is loaded in memory in its entirety, and very long times (unreported here) are achieved if this condition is not met.

By looking at Fig. 6, we already notice a huge improvement if we

L	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

TABLE I
INDEX REQUIREMENTS IN TERMS OF STORAGE AND LOADING TIME, AS A FUNCTION OF L . NOTE THAT THE SOURCE DATABASE SIZE IS 846MiB.

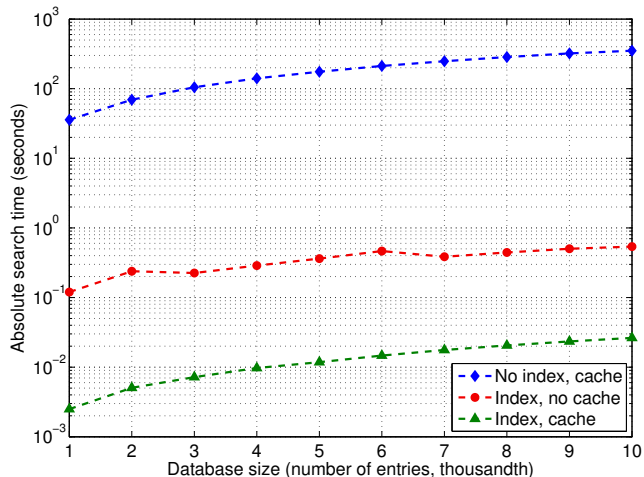


Fig. 6. Absolute search time as a function of the database size and comparing exhaustive search (which has database on memory), indexing and fully-cached indexing. The index parameter L is 24.

just index (thus accessing the disk at each comparison), with a time gain factor that ranges from about 300 for 1000 database entries, growing to over 650 when there are 10000 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 (see III-B), so their impact is progressively reduced. The small fluctuations seen in the middle curve may be due to the concurrent access to the common disk in the course of simulations.

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 just over 14000. 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 just above 24000 (see again Fig. 3 when $L = 24$), leaving room for a more optimized implementation of the index and/or the cache.

Finally, Fig. 7 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. 6, and the full- and no-cache approaches may be compared with Fig. 4 when $L = 24$. Considering the whole database, we observe a gain of almost 130 for the cached-index approach, in comparison to a value of about 6 if no cache is present and of just below 0.01 when no index is present, i.e. the search time is more than a hundred times the playing time.

IV. CONCLUSIONS AND FUTURE WORK

In this work we discussed relevant scalability issues associated with the high-dimensional, automatic audio recognition problem, and especially with regards to a well-known, robust fingerprint algorithm. In particular, precise tests show the importance of the parameter b , in order to afford an efficient search strategy. We also point out the capital focus that should be put on the caching issue, so to not compromise the indexing benefits because of the slow disk access.

Careful comparisons and insights were provided on the basis of extensive simulations on a very large, 10000-entry database, also in comparison to the trivial, exhaustive search approach. Future work will

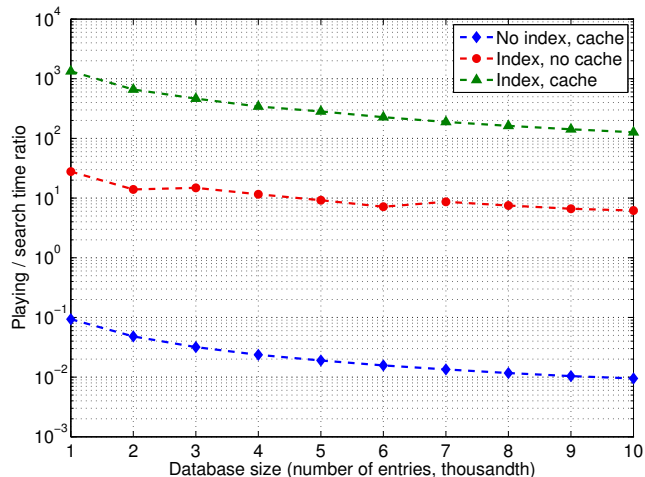


Fig. 7. Ratio between search time and playing time of the excerpt, as a function of the database size and comparing exhaustive search (which has database on memory), indexing and fully-cached indexing. The index parameter L is 24.

focus on a better evaluation of the overall reliability of the indexing technique and on the eventual implementation of possibly improved search strategies. A further step, which could be of interest for certain high-end applications, will be that of planning a parallel deployment of the framework.

REFERENCES

- [1] G. Velickic, E.L. Titlebaum, M.F. Bocko, "Musical note segmentation employing combined time and frequency analyses", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Montreal, Canada, May 2004.
- [2] P. Cano, E. Batlle, T. Kalker, J. Haitsma, "A review of algorithms for audio fingerprinting", *International Workshop on Multimedia Signal Processing*, St. Thomas, Virgin Islands (USA), December 2002.
- [3] C.J.C. Burges, J.C. Platt, S. Jana, "Distortion discriminant analysis for audio fingerprinting", *IEEE Transactions on Speech and Audio Processing*, vol. 11, pp. 165-174, May 2003.
- [4] A. Ramalingam, S. Krishnan, "Gaussian Mixture Modeling of Short-Time Fourier Transform Features for Audio Fingerprinting", *IEEE Transactions on Information Forensics and Security*, vol. 1, pp. 457-463, December 2006.
- [5] C. Bellettini, G. Mazzini, "On audio recognition performance via robust hashing", *International Symposium on Intelligent Signal Processing and Communication Systems*, Xiamen, China, November 2007.
- [6] C. Bellettini, G. Mazzini, "Reliable Automatic Recognition for Pitch-Shifted Audio", *International Conference on Computer Communications and Networks*, St. Thomas, Virgin Islands (USA), August 2008. (to appear)
- [7] F. Kurth, M. Clausen, "Full-Text Indexing of Very Large Audio Data Bases", *Proceedings of the 110th AES Convention*, Amsterdam, The Netherlands, May 2001.
- [8] P. Cano, E. Batlle, H. Mayer, H. Neuschmied, "Robust Sound Modeling for Song Detection in Broadcast Audio", *Proceedings of the 112th AES Convention*, Munich, Germany, May 2002.
- [9] D. Jang, S. Lee, J.S. Lee, M. Jim, J.S. Seo, S. Lee, C.D. Yoo, "Automatic commercial monitoring for TV broadcasting using audio fingerprinting", *AES 29th International Conference*, Seoul, Korea, September 2006.
- [10] A. Ribbrock, F. Kurth, "A full-text retrieval approach to content-based audio identification", *IEEE Workshop on multimedia signal processing*, St. Thomas, Virgin Islands (USA), December 2002.
- [11] J. Haitsma, T. Kalker, J. Oostveen, "Robust audio hashing for content identification", *International Workshop on Content-Based Multimedia Indexing*, Brescia, Italy, September 2001.
- [12] <http://www.knowmark.it>