Exhibit 54

Pattern Recognition 41 (2008) 3467–3480

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Waveprint: Efficient wavelet-based audio fingerprinting

Shumeet Baluja*, Michele Covell

Google, Inc., Mountain View, CA, USA

ARTICLE INFO

Article history:
Received 17 July 2007
Received in revised form 6 March 2008
Accepted 2 May 2008

Keywords:
Audio retrieval
Applications
Image/video retrieval
Pattern analysis

ABSTRACT

In this paper, we present Waveprint, a novel method for audio identification. Waveprint uses a combination of computer-vision techniques and large-scale data-stream processing algorithms to create compact fingerprints of audio data that can be efficiently matched. The resulting system has excellent identification capabilities for small snippets of audio that have been degraded in a variety of manners, including competing noise, poor recording quality and cell-phone playback. We explicitly measure the tradeoffs between performance, memory usage, and computation through extensive experimentation. The system is more efficient in terms of memory usage and computation, while being more accurate when compared with previous state of the art systems. The applications of Waveprint include song identification for end-consumer use, copyright protection for audio assets, copyright protection for television assets and synchronization of off-line audio sources, such as live television.

1. Introduction

Audio fingerprinting provides the ability to link short, unlabeled snippets of audio content to corresponding data about that content. There are an immense number of applications for audio fingerprinting. In the consumer space, it provides the ability for consumers to automatically identify unknown audio, such as songs [1,2]. Songs can be tagged automatically with the performing artist's name, album or other metadata. Other applications include broadcast monitoring for not only songs, but also other non-musical content. One such application is to continuously monitor the number, time, and duration of broadcasts of pre-recorded programs and advertisements. An application that is rapidly growing in importance is the immediate identification of copyrighted material. As the ease and popularity of music and video sharing increases [3,4], the need to recognize copyrighted content grows. For this task, audio fingerprinting is proving to be useful for recognizing not only audio material, but also video content (through the use of the audio track). Similarly, automatically recognizing ambient audio signals from television broadcasts has also proven to be much easier than recognizing the video stream. This has enabled numerous applications ranging from enhanced television [5] to automatic advertisement detection and replacement [6].

There are a number of issues that make fingerprinting a challenging task. The simplest approaches, directly comparing the audio samples, will not work. The query and stored version of a song may be aurally similar while having distinct sample values. Even direct comparisons of spectrograms are susceptible to changes in quality settings, compression schemes, equalization settings, reference codecs, etc. Further, if radio broadcasts are included in the probe set, tempo and sometimes pitch changes may be introduced, since radio stations often change the speed of a song to fit their programing requirements [7]. Finally, there are numerous issues introduced by the many forms of playback available to the end-consumer. Music that is played through a cell-phone, computer speakers, or high-end audio equipment will have very different audio characteristics that must be taken into account.

In this paper, we describe a system which can handle signals that have been degraded by echoes, passed through a cell-phone codec, recorded in the presence of structured noise, and modified in its timing, with respect to either/both pitch and tempo. The system is always tested where even coarse offsets into the matching song are unknown (for example, cues such as "the snippet occurs with the first 30 s" will not be used). This offset-invariant testing is required for broadcast monitoring and for edited-media identification. Most open-source music-track identification systems (such as Foosic libFooID [8] and MusicBrainz Picard Tagger [9]) are not designed for use under this assumption. Instead, we compare ourselves to another state-of-the-art open-source system [10], allowing us to compare memory usage and computational load as well as accuracy.

For our explorations, there are practical system-design constraints. The most restrictive are the memory requirements. To minimize the number and size of disk reads, we will keep as much information in a computer's memory as possible. Therefore, the fingerprints need to be compact. Second, the system must be designed to be easily parallelizable to multiple machines. New audio content

* Corresponding author.
E-mail addresses: baluja@gmail.com, shumeet@google.com (S. Baluja), covell@google.com (M. Covell).

may be continuously added to the system. Third, the system must be able to work with non-music information, such as the audio track of television programs. The least restrictive of our requirements is the computation speed; we need to recognize an audio-track as it is being played. Therefore, it must be at least real-time.

In Section 2, we review previous approaches to audio identification, with special attention to those systems that influenced our own approach. In Section 3, we present and explain the methods we have used for our system, termed *Waveprint* [11]. Section 4 explores the tradeoffs across computational complexity, memory usage and recognition accuracy, as a function of the parameter settings. Section 5 provides detailed performance results and compares the performance to the best previously published system [10]. In recognition of the importance of scaling properties, Section 6 analyzes, in practice, the accuracy and computation required as a function of known-audio database size. Finally, Section 7 concludes the paper and outlines several directions for further exploration.

2. Previous approaches

A good overview of the audio fingerprinting field can be found in Refs. [12,13]. Most track-based approaches, such as Foosic libFooID [8] and MusicBrainz Picard Tagger [9], often make use of starting-time constraints in matching fingerprints. Some commercial products are targeted at broadcast or edited content [2,14–17].

One of the most widely used systems [18] uses overlapping windows of audio to extract interesting features. Thirty-three Bark-frequency cepstral coefficient (BFCC) bands, covering 300-2000 Hz, are used for the spectral representation, with spectral slices taken every 11.6 ms and with each slice based on 370 ms of audio. This large overlap ensures that the sub-fingerprints slowly vary over time, providing fine-grain shift invariance to the representation. A vector of 32 bits, called a sub-fingerprint, are extracted from each slice position according to the increasing/decreasing difference pairs across successive bands and successive spectral slices. These sub-fingerprints are largely insensitive to small changes in the audio signal since no actual difference values are kept; instead, only the sign bits compose the sub-fingerprint. Comparisons with these fingerprints are efficient; a simple Hamming distance can be used.

For a database of 10,000 songs of average length 5 min, approximately 250,000,000 sub-fingerprints were generated [18]. During retrieval from this reference set, the entire database cannot be examined. Instead, the authors of Ref. [18] assume that, even with potential audio degradations, at least one sub-fingerprint from each query sound will have a (correct) exact match in the database. This allows them to use a hash table to find exact copies. Once an exact match is found, the temporal sequencing of the indexed song is used to analyze the surrounding sub-fingerprints. This allows a simple computation of the Hamming distance over the full snippet length. The only "search" done is over candidates that are retrieved by one or more exact matches. If extreme distortions are expected, such that even a single exact match is not guaranteed, their approach is modified to search for vectors that are small Hamming distances away from the original sub-fingerprints.

A recent extension to the above work was presented in Ref. [10]. Based on Ref. [18], Ke introduced a learning approach into the feature selection process. An important insight provided by Ref. [10] is that the 1-D audio signal can be processed as an image when viewed in a 2-D time–frequency representation. Their learning system finds features that integrate the energy in selected frequencies over time via AdaBoost learning [19]. The basis of feature selection is the discriminative power of the region in differentiating between two matching frames (within a distortion set) and two mismatched frames. Thirty-two "boxlet" features are selected, each yielding a bi- the 32-bit features found by Ref. [18]. Temporal coherence is measured by a simple transition model.

An alternate approach is explored in Refs. [20,21]. Their work uses a perceptually weighted log spectrogram as the feature set. This log spectrogram is sampled only once every 186 ms and uses 372 ms of data to provide 2048 frequency samples between DC and 5.05 kHz. Burges et al. extract noise-tolerant fingerprints from this spectrogram using distortion discriminant analysis (DDA). The fingerprints are more complex than in the studies by Refs. [10,18], but also summarize longer segments of audio than in the other work. DDA is based on a variant of linear discriminant analysis (LDA) called oriented principal components analysis (OPCA). OPCA assumes that distorted versions of the training samples are available. OPCA selects a set of directions for modeling the subspace that maximizes the signal variance while minimizing the noise power. OPCA yields a set of potentially non-orthogonal vectors that account for noise statistics [21]. The final result of their system effectively maps 110 K inputs into 64 outputs. These 64 outputs are the sub-fingerprints that are matched. The experiments conducted in Refs. [20,21] have found that the fingerprints are resistant to problems with alignment and types of noise not found in the training set.

3. Description of the Waveprint system

Our system builds on the insight from Ref. [10]: computer-vision techniques can be a powerful method for analyzing audio data. However, instead of a learning approach, we examine the applicability of a wavelet-based approach developed by Ref. [22] for efficiently performing image queries in large databases. To make the algorithm scale, we employ hashing approaches from large-scale data-stream processing. The sub-fingerprints that we develop are more comprehensive than used in Refs. [10,18] since they will represent a longer time period, in a manner closer to the work presented in Ref. [21].

We structure our discussion of the Waveprint system in three stages. The first is the creation of a compact representation of songs that will be inserted into our database for retrieval. The second stage is the creation and organization of that database. The third stage is an efficient procedure to lookup a candidate match when a new query audio snippet is received.

3.1. *Fingerprint creation*

The overall architecture of the fingerprint creation procedure, described in this section, is shown in Fig. 1. Fig. 1 also shows a typical spectrogram and its decomposition into a sparse representation that will be converted into the signatures stored in our database.

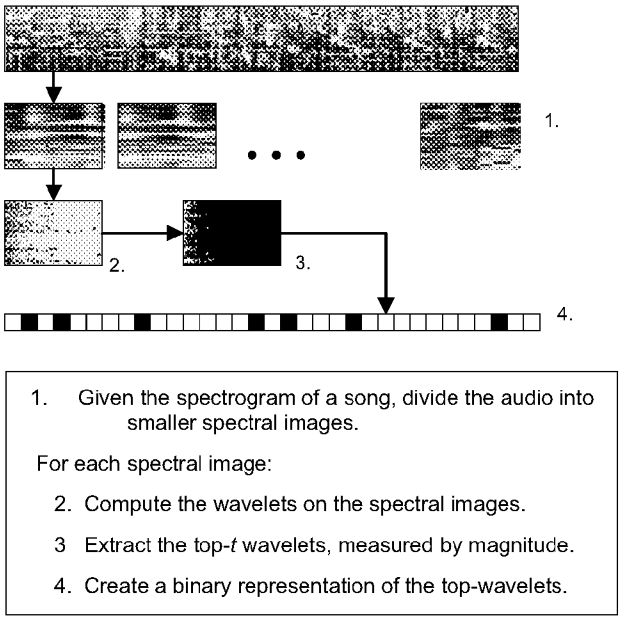
3.1.1. *Spectral-image creation*

We begin by converting the audio input into a spectrogram [23]. The simplest spectrogram extracts overlapping segments of audio, tapers each audio slice to reduce the sensitivity to end effects, takes the Fourier transform of each audio slice to give a short-time frequency representation, and then discards the phase component, keeping only Fourier magnitudes on the positive frequency bands. We create the spectrograms using parameter settings that worked well in previous audio fingerprinting studies [18]. We use slices that are 371 ms long, taken every 11.6 ms, reduced to 32 logarithmically spaced frequency bins between 318 Hz and 2 kHz. An important consequence of the slice length/spacing combination of parameters is that the spectrogram varies slowly in time, providing matching robustness to position uncertainty (in time). The use of logarithmic spacing in frequency was selected based on simplicity, since the detailed band-edge locations are unlikely to have a strong effect

We then extract spectral images, $11.6 * w$ ms long, each sampling offset apart. The sampling offsets that we use are constant in the database-creation process (s sec separation) but are non-uniform in the probe sampling process. We discuss the parameter choices (s and w) further in Section 4. Extracting known-length spectral images from the spectrograms allows us to create sub-fingerprints that include some temporal structure without been unduly susceptible to gradual changes in timing. At this point in the processing, we treat the spectral images as if they were components in an image-query system. Rather than performing retrieval by directly comparing the "pixels" of the spectral image, we will use a representation based on wavelets.

3.1.2. *Wavelets on spectral images*

Wavelets are a mathematical tool for hierarchically decomposing functions. They allow a function to be described by its overall shape, plus successively increasing details. A good description of wavelets can be found in Ref. [24]. We use wavelets in this audio-retrieval task



1. Given the spectrogram of a song, divide the audio into smaller spectral images.

For each spectral image:

2. Compute the wavelets on the spectral images.
3 Extract the top-*t* wavelets, measured by magnitude.
4. Create a binary representation of the top-wavelets.

Fig. 1. Overall architecture for fingerprint creation.

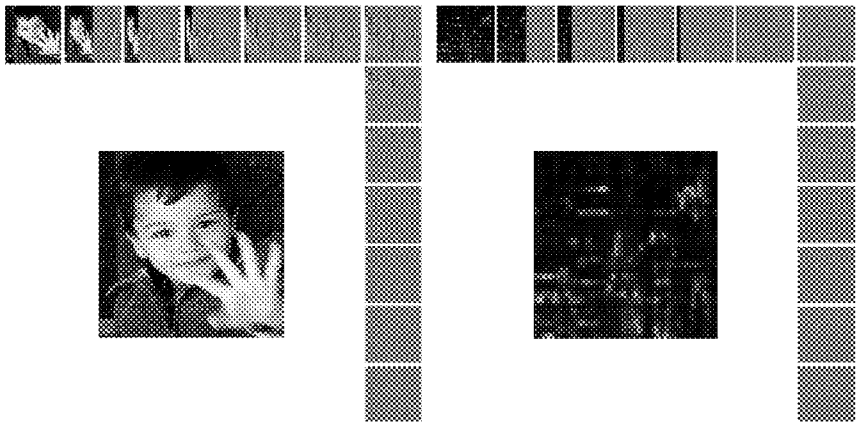
due to their successful use in image retrieval [22]. In Ref. [22], rather than comparing images directly in the pixel space, they first decomposed the image through the use of multi-resolution Haar wavelets. Samples of the wavelet decomposition, for a typical image and for a spectrogram image, are shown in Fig. 2. Their system supported query images that were hand drawn or low quality sketches of the target image. The results were better than those achieved through simple histogram or pixel differences.

In our system, for each of the spectral images, Haar wavelets are computed. By itself, the wavelet image is not resistant to noise or audio degradations. To provide this resistance, instead of using the entire set of wavelets, we only keep the ones that most characterize the image, by selecting the *top-t* wavelets (by magnitude), where $t \ll$ *image_pixels*. When we look at the wavelets for successive images for two songs, we see easily identifiable patterns both in the wavelet space and even more clearly when the *top-t* wavelets are kept. This is shown in Fig. 3.

One of the important findings by Jacobs [22] was that only sign bits (not the full coefficients) for the top wavelets were needed. This allows a bit-vector representation in which each wavelet was represented as only two bits—which is ideal for applications, such as this one, with stringent memory requirements. For each of the *top-t* magnitude wavelets, it is labeled as 10 (for positive values) or 01 (for negative values). The majority of wavelets, which are *not* in the *top-t* set are labeled with 00. This representation makes the resulting bit vector extremely sparse and amenable to further dimensionality reduction. For the next step of dimensionality reduction, we use Min-Hash, which is described in the next section.

3.2. *Min-Hash-based sub-fingerprints*

The final step of sub-fingerprint creation is to determine a compact but nearest-neighbor indexable representation of the sparse wavelet-vector described in the previous section; for this we explore the use of Min-Hash [25]. To support efficient nearest-neighbor retrieval, we require that sub-fingerprint $v1$ and sub-fingerprint $v2$ are highly similar if and only if wavelet signature ($v1$) and wavelet signature ($v2$) are highly similar. Because we retain only the *top-t* wavelet coefficients, we determine similarity based on those top wavelets, without rewarding matches on the zeroed positions. For the purposes of this discussion, given two vectors $v1$ and $v2$, we refer to match types as being of four types a, b, c, d, as shown in Table 1, depending on the corresponding bits in the vectors. Given these types of matches/mismatches, we note that for sparse vectors, most of the



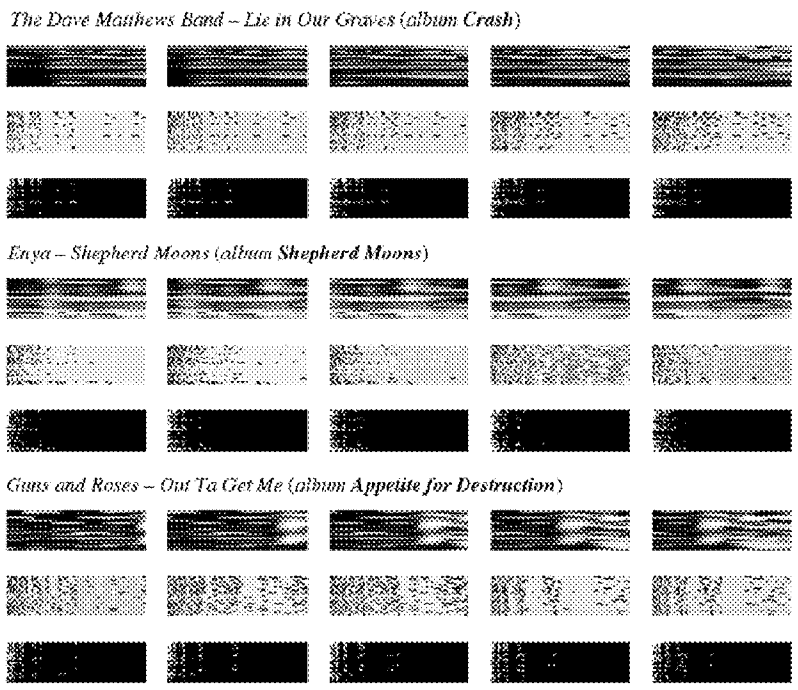


Fig. 3. The representation for three songs—five consecutive frames shown for each, skipping 0.2 s. For each song, the top row is the original spectrogram image, the second row is the wavelet magnitudes, the third row shows the top-200 ($t = 200$) wavelets. Note that the top wavelets have a distinctive pattern for each of the three songs. (For each song, the top 2 rows in the figure have been extensively visually enhanced to be visible when printed on paper.)

Table 1
Types of match/mismatch between single bits of two binary vectors

Type	Vector 1	Vector 2
a	1	1
b	1	0
c	0	1
d	0	0

bit positions will be of type *d*. To avoid computing similarity based on the uninformative zeroed rows, we will define the similarity of two vectors to be the relative number of rows that are of type *a* from the other non-zero rows: i.e., Similarity $(v1; v2) = a/(a + b + c)$.

The Min-Hash technique works with binary vectors as follows: Select a random, but known, reordering of all the vector positions. Reorder each vector by this permutation. With this new ordering, determine in which position the first "1" occurs. It is important to note for two vectors, $v1$ and $v2$, the probability that first_1_occurrence($v1$) = first_1_occurrence($v2$) is the same as the probability of finding a row that has a 1 in both $v1$ and $v2$, from the set of rows that have 1 in either $v1$ or $v2$. Therefore, for a given permutation, the hash values agree if the first position with a 1 is the same in both bit vectors, and they disagree if the first such position is a row where one but not both, vectors contained a 1. *Note that this is exactly what is required; it measures the similarity of the sparse bit vectors based on matching "on" positions.*

We repeat the above procedure multiple times, each time with a new permutation of bit positions. If we repeat the process *p* times, with *p* unique permutations, we get *p* largely independent projections of the bit vector. These *p* values are the signature for the bit vector. We can compare the similarity of the bit vectors by looking at the exact matches in the signatures of length *p*; for a large enough *p*, it will be very close to the similarity of the original vectors. In our system, we do not keep the intermediate bit representation computed signature; this is the final sub-fingerprint of the spectral image. We experimented with a variety of values for *p*; these are presented in Section 4. An in-depth description of the Min-Hash process is given in Ref. [25]. Methods to make the matching process efficient given these signatures will be presented with the description of the retrieval process.

On a pragmatic note, because memory efficiency is paramount for deployment, each signature element must be small. Instead of tracking the first position in which a "1" occurs, we truncate the computation at position 255. If the first 1 occurs after position 255, it is demarcated as if it occurred at position 255. This allows us to keep each component of the signature as a single byte. Fig. 4 shows a histogram of the position of the first 1 computed for the snippets in our database, for three values of *t* (the number of top wavelets kept). Note that the cumulative probability of occurring after 255 is very low when more than 50 top coefficients are kept; this indicates that the single-byte representation loses little accuracy.

In this study, Min-Hash reduces the size of the signatures from the intermediate wavelet representation described in this previous section to a compact representation of *p*-bytes. There are numerous other techniques that are commonly used for dimensionality reduction—among them techniques such as principal components analysis (PCA). We chose Min-Hash due to a chain of reasons. We require discriminative power across our top-wavelet signatures, not descriptive power. This requirement means that PCA may not be the best representation and instead has traditionally been handled by LDA-based methods [21]. Since our top wavelet signatures are already a sparse-vector representation, we employed techniques explicitly designed to handle probabilistic matching across sparse vectors, and did not require transformation to continuous values. Min-Hash is such a method and has been used extensively (and successfully) in data-stream processing for this class of

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.