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

A R T I C L E   I N F O

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

A B S T R A C T

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).

0031-3203/$30.00 © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2008.05.006

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 binary value. These 32 bits are then used in an analogous procedure to 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 under such coarse sampling (only 32 samples across frequency).

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 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.

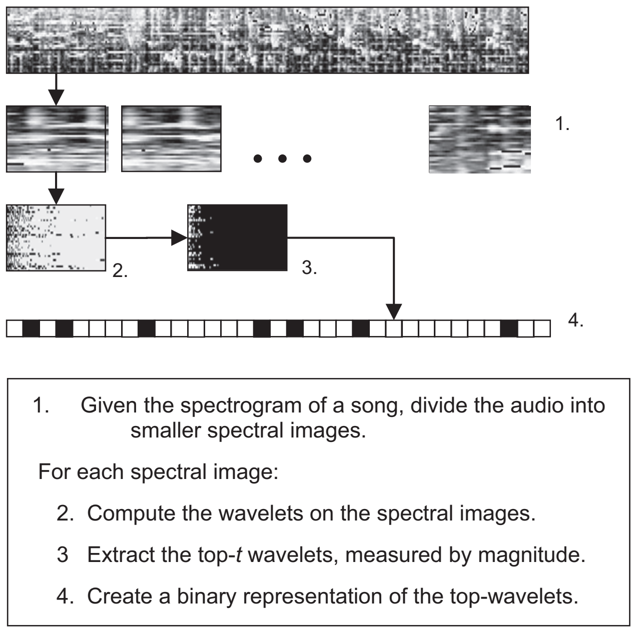
1.
2.
3.
4.
1. Given the spectrogram of a song, divide the audio into smaller spectral images.
For each spectral image:
2. Compute the wavelets on the spectral images.
3 Extract the top-*t* wavelets, measured by magnitude.
4. Create a binary representation of the top-wavelets.

Fig. 1. Overall architecture for fingerprint creation.

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

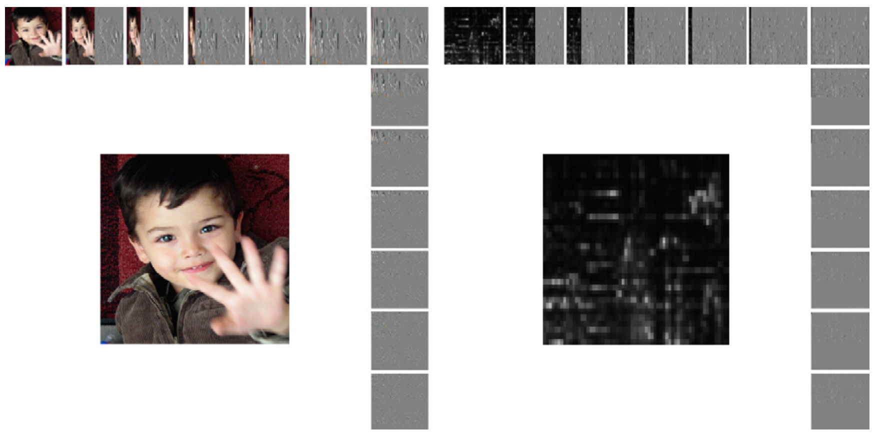

Fig. 2. Wavelet examples for a typical image (left) and a spectrogram image (right). These are computed in the "standard" manner—one dimension at a time.

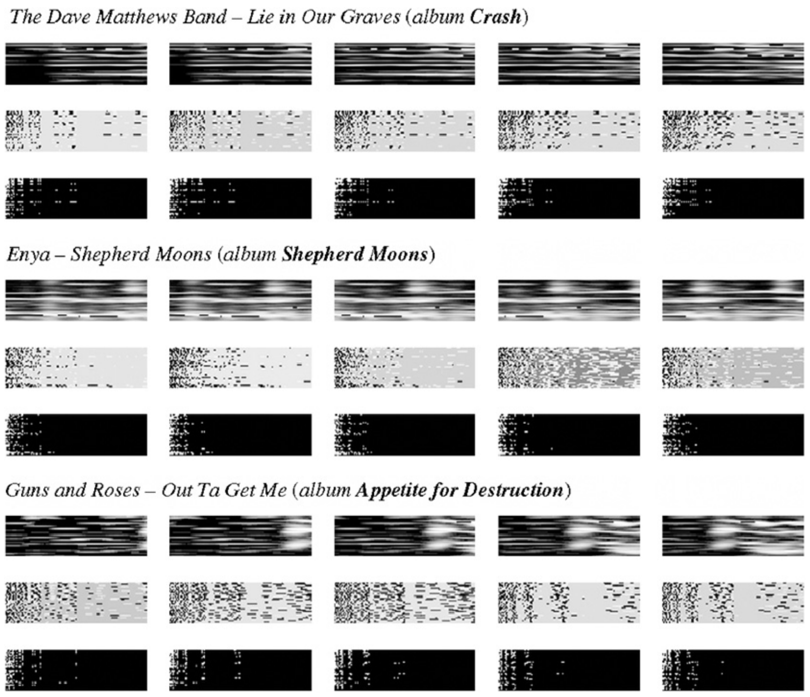

Fig. 3. The representation for three songs—five consecutive frames shown for each, skipping 0.2 s. For each song, the top row is the original spectrogram image, the second row is the wavelet magnitudes, the third row shows the top-200 ($t = 200$) wavelets. Note that the top wavelets have a distinctive 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 described in the previous section. Instead, we store the Min-Hash 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 problems.

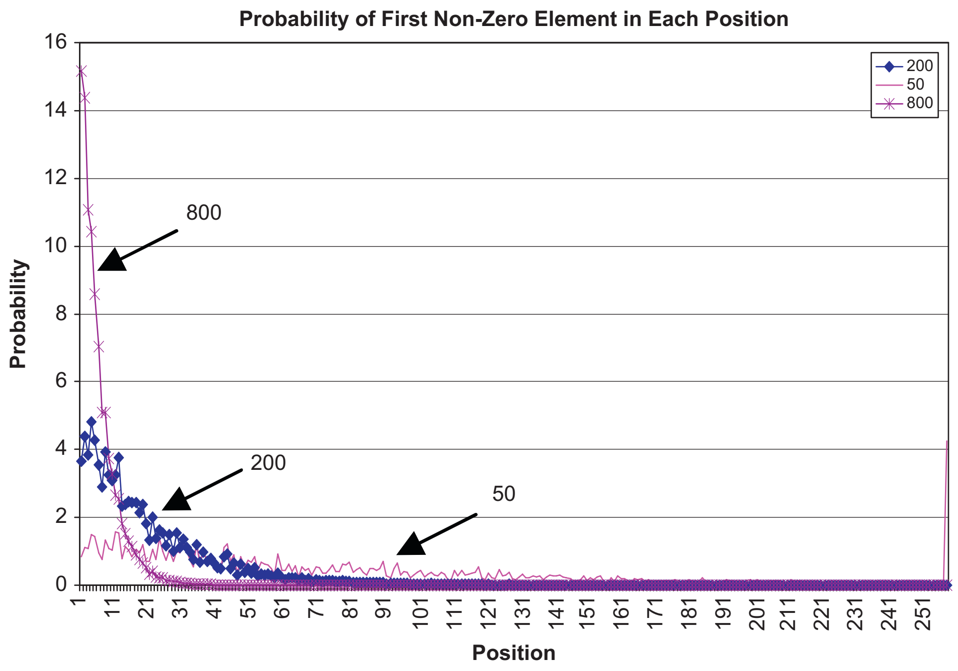

Fig. 4. Probability of the first non-zero element occurring in each bit position, measured over all of the snippets in the database. The curves shown are for keeping the top 200 wavelet coefficients (as we will use in our system), keeping only 50, and keeping 800. Note that with 50 top coefficients, there is a large percentage after the truncation point of 255 (the spike at the right edge of the graph). When keeping 800 coefficients, notice that the probability of a "1" in the first position examined increases; this is because the sparseness of the bit vectors has decreased significantly.

3.3. Database-creation process

To this point, we have reduced the amount of information from each spectrogram through three steps. First, we kept only the top wavelets of the spectral images associated with the spectrogram. Second, we reduced each of the top wavelets to only two bits. Third, we used the Min-Hash procedure to reduce the resulting bit vector to p values, which became the sub-fingerprint that was stored for the audio segment.

After these steps, each spectral image is represented by a series of p 8-bit integers, the sub-fingerprint. Even with this compression, efficiently finding near-neighbors in a p-dimensional space is not a trivial task (when $p > 50$); naive comparisons are not practical. Instead, we use a technique termed locality-sensitive hashing (LSH) [26]. It is not only efficient in the number of comparisons that are required (a small fraction of the data set will be examined for a lookup), but also provides noise-robustness properties.

In standard hashing, the query's sub-fingerprint is used, in its entirety, as a hash key that indexes into a bin that contains elements that have also been hashed into the same bin [27]. The elements that are found in that bin are then considered candidate matches, and can be compared in detail (for example, by comparing the full signatures) to obtain the final matches. The implicit assumption in using the entire fingerprint is that the procedures employed to create the sub-fingerprint already fully accounted for the noise that the system would encounter. In contrast to the standard hashing procedure, LSH performs a series of hashes, each of which examines only a portion of the sub-fingerprint. The goal is to partition the feature vectors into l subvectors and to hash each into l separate hash tables, each hash table using one of the subvectors as input to the hash function. In this study, because the Min-Hashes are signatures in which the constituents are determined randomly, we simply assign non-overlapping, consecutive, p/l elements for each hash. Candidate neighbors can then be retrieved by partitioning the probe feature vector in the same manner, and collecting the entries in the corresponding hash bins. The final list of potential neighbors can be created by vote counting, with each hash casting votes for the entries of its indexed bin, and retaining the candidates that receive a minimum number of votes, v. If $v = 1$, this takes the union of the candidate lists. If $v = l$, this takes the intersection.

Unlike standard hashing schemes, in which the entire fingerprint must be exactly the same for candidate retrieval, LSH relaxes this restriction, requiring as little as p/l bytes of the sub-fingerprints to match. It should be noted that the number of hash tables, l, can vary independently from the size of the sub-fingerprint, p. We experiment with a variety of setting for l, while keeping the size of the Min-Hash signature constant.

When creating the audio fingerprint database, we extract a sub-fingerprint of length p, once each s seconds. We retain these sub-fingerprints in the observed time-order sequencing, with a pointer to a song identifier. The time ordering is encoded implicitly in our system by the sub-fingerprint index within a linear array. In addition to the time-ordered representation of the sub-fingerprints, each sub-fingerprint is inserted into the l hash tables of our LSH according to the hash-function mapping of the corresponding p/l bytes to a hash bin. This provides us with the complete song database from which we will retrieve matching songs; this is described in the next section.

3.4. Retrieval process

In the previous two subsections, we described the technique to create sub-fingerprints and to store them in a database for fast retrieval. In this section, we describe the process to find similar snippets once given this database and a new query snippet. The complete procedure is shown graphically in Fig. 5. Note the first few steps (steps 1–5) of retrieval process follow the database sub-fingerprint creation process closely; this is to ensure that the query and stored data are processed in the same manner. After the candidate matches are found (steps 6–8), a measure of temporal coherence is used to verify the findings (not shown in the diagram).

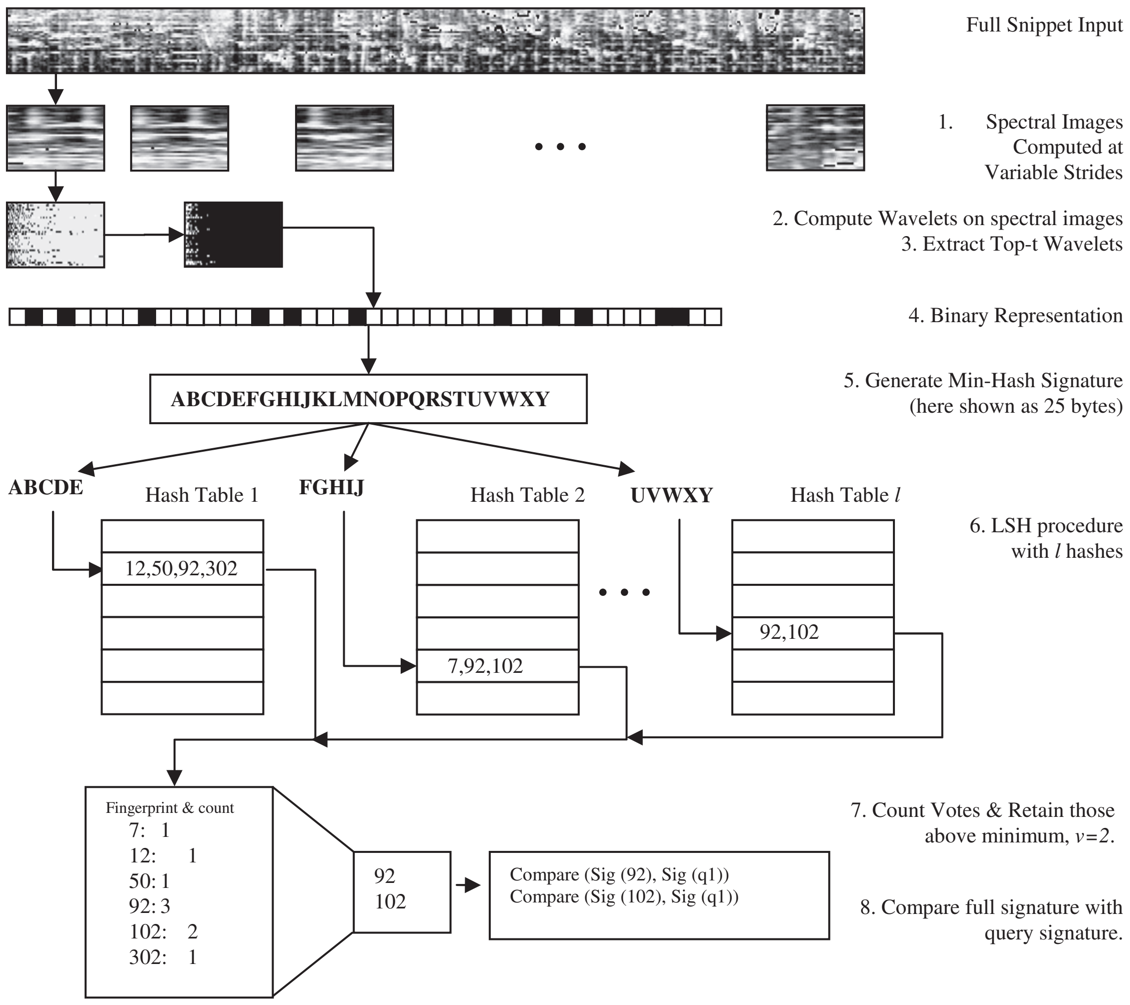
Full Snippet Input
1. Spectral Images Computed at Variable Strides
2. Compute Wavelets on spectral images
3. Extract Top-t Wavelets
4. Binary Representation
ABCDEFGHIJKLMNOPQRSTUVWXY
5. Generate Min-Hash Signature (here shown as 25 bytes)
ABCDE
Hash Table 1
FGHIJ
Hash Table 2
UVWXY
Hash Table l
12,50,92,302
7,92,102
92,102
6. LSH procedure with l hashes
Fingerprint & count
7: 1
12: 1
50: 1
92: 3
102: 2
302: 1
92
102
Compare (Sig (92), Sig (q1))
Compare (Sig (102), Sig (q1))
7. Count Votes & Retain those above minimum, v=2.
8. Compare full signature with query signature.

Fig. 5. Overall architecture of the retrieval process. Dynamic-time warping not shown.

The first difference in the retrieval process, in comparison to the fingerprint generation process, is that the song is divided into randomly overlapping segments rather than uniformly overlapping segments. Randomly selecting the stride amount avoids problems of unlucky alignments; if the sampling of the probe is kept constant, it may repeatedly find samples that have uniformly large offsets from the sampling used to create the database. After the audio snippet is created, its signature is computed in exactly the same manner as described in the database generation process (steps 2–5). The next steps 6–9 are described in the remainder of this section.

3.4.1. *Lookup using LSH*

We described the basic characteristics of LSH in the previous subsection. Each component hash table votes for the sub-fingerprints that were retrieved using its p/l-bytes. Then, only those sub-fingerprints with a minimum of v component votes are retained. For example, by setting $v = 1$, a candidate must only match in one of the subregions in order to be included in the final candidate list. At the opposite extreme, by setting $v = l$, the candidate must match on all subregions and the LSH operates identically (although inefficiently) to a standard hash table.

The fingerprints that have at least v votes are then compared with the query sub-fingerprint (we refer to this as a *full compare*). Because each byte of the sub-fingerprint is a Min-Hash signature, we simply look at the number of bytes (out of p) that match exactly. The sub-fingerprint with the maximum of this score is the best match for the query spectral image.

3.4.2. *Dynamic programming for temporal ordering*

To this point, we have discussed matching individual sub-fingerprints from the probe into the database. In this section, we describe methods to accumulate evidence across sub-fingerprints over the duration of the probe snippet.

The simplest way to combine evidence is voting without regard to temporal information. With this approach, we keep a similarity counter for each song. At the start of a new probe snippet, all of these counters are zero. Then, for each candidate subfingerprint match that passed the required hash-voting threshold, v, we increment the corresponding song counter by the Hamming similarity between the probe and candidate sub-fingerprints. The advantage of this approach is its simplicity and low memory requirements. The disadvantage is that evidence for a song accumulates without regard to temporal ordering of the sub-fingerprint matches. Pairs of sub-fingerprints $p(t)/p(t + \Delta t)$ within the probe are rewarded equally for matching song sub-fingerprints $s(t)/s(t - \Delta t)$ as they are for matching $s(t)/s(t + \Delta t)$: there is no penalty for time reversal. Similarly, for matching $s(t)/s(t + 10\Delta t)$: there is no penalty for changing tempo.

We also explored dynamic-time warping (DTW) to accumulate evidence across time. DTW is a form of dynamic programming for imposing "tempo" constraints in mapping one sequence onto

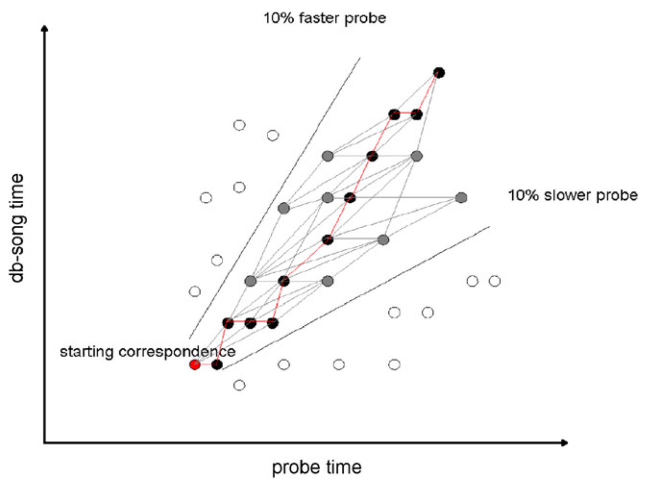

Fig. 6. Conceptual overview of dynamic-time warping, given a starting correspondence.

another [28]. Given a starting correspondence between the two sequences, the most probable path that satisfies the tempo constraints is efficiently found. We use both global-slope constraints (at most 10% change in tempo from the probe to the candidate match) and local-slope constraints (single-match location per probe subfingerprint without local-time inversions). Fig. 6 shows possible match paths under our constraints. When using temporal constraints, we took the figure-of-merit from the best DTW path as the match-score.

This leaves unaddressed the task of selecting the starting correspondence. Each probe sub-fingerprint can propose multiple matches in any given song. To avoid uncontrolled growth in numbers of possibilities, we impose two limits. First, we only allow each sub-fingerprint to propose a limited, small, number of potential matches for itself, across the full database of songs. Second, within each song, we use A^* pruning on the candidate space [29].

3.5. Summary of database creation and retrieval

Here we provide a summary of some of the more important considerations and resulting design decisions made to this point. First, we need to be resistant to noise, whether it occurs in the form of signal degradation/transformation, or whether it occurs because of misalignment. Noise is handled primarily in three portions of the system: (1) Retention of only the top few wavelet components: this leaves only the most significant time–frequency features and ignores small components. (2) Binary quantization of retained wavelet components: this reduces many normalization issues and avoids matching precise magnitudes. (3) Soft hashing of sub-fingerprints using LSH for candidate retrieval: parts of the top-wavelet pattern can be different across the candidate pairing. If we did not use LSH (or equivalently if $v = l$), an exact match of these signatures would be required.

Second, the system needs to be efficient in memory usage. The memory usage is affected by the following: (1) the database sampling stride, s (shorter sampling strides, for improved recall, require more memory), (2) the number of component hash tables, l (each component table must point to all of the snippets that it reverse indexes) (3) use/non-use of temporal tracking (the A^* pruning on candidate DTW tracks requires a noticeable amount of memory).

Third, the system must be computationally efficient. The obvious contributors to efficiency are the length of the signature that must be compared and the number of hashes that must be completed; small values are tried for each. A less obvious, but significant contributor to computational efficiency, is the number of votes required in the hashes (v). By increasing this number, we dramatically reduce the number of full comparisons that need to be conducted; the impact on accuracy will be carefully studied, as the likelihood of a missed match increases.

4. Parameter exploration

One of the intrinsic difficulties in designing systems with multiple components is conducting a thorough exploration of the parameter settings. Because of the need to deploy this system in a large-scale production setting, we extensively measure the effects of parameter values. The results reported in this section represent multiple decades of computational processing; over 50,400 different parameter combinations were tried to ensure that we select the best settings and understand the tradeoffs with each parameter. The parameters that we explored for the fingerprint and database generation and retrieval are shown in Table 2. To make the results more understandable, we also present some of our derived intuition about how the parameters affect computational complexity, memory usage and retrieval accuracy.

To explore this large parameter space, we used a 10,000 song database, with an average song duration of 3.5 min. Since our goal at this point is to understand the general system performance, we did not use heuristics such as unequal weighting of songs to influence the results. In the future, depending on the application, unequal representation of either specific portions of the song (i.e., the song beginnings and choruses) or unequal weighting to specific songs (i.e. the current most popular songs) can be used to reduce the memory usage or computation expense.

For the parameter setting experiments, we used 1000 independent probe snippets into this song database. Each of these probes included some form of distortion: time-offset only; added echo; spectral coloration; low-rate MP3 re-encoding; low-rate GSM cell-phone re-encoding; 10% time-scale modification; 2% linear-speed modification (LSM); or competing music mixed with original. We describe these distortions in more detail in Section 5. In this section, our goal was not a detailed understanding of how the distortions affect the performance but rather to present the system with a rich enough challenge set so as to accurately discriminate amongst the various possible parameter combinations.

Our accuracy results for the experiments in this section indicate the percentage of times we selected the correct song from the database using the distorted probe snippet. On this task, because we are operating against 10,000 equally weighted songs, random chance is 0.01% correct.

4.1. Selecting the parameter set

With over 50,400 parameter settings and three interesting attributes (retrieval accuracy, memory usage, computational load), there are numerous ways to report the results. In Fig. 7, we present our results for three different retrieval-accuracy settings. Fig. 7A shows the best accuracy achieved by our system. If the accuracy requirements are reduced, the computation and memory requirements can be dramatically improved. Fig. 7C shows the results for relaxing retrieval accuracy to over 80%.

There are many interesting points to note about our results. First, corresponding to the graph showing the best accuracy (Fig. 7A) only 122 out of the 50,400 experiments had accuracies at or above 97.5%. As many as 320 additional parameter combinations might have achieved accuracies in this range but were terminated early, due the impractically large computational load (more than 3M comparisons per probe snippet). Of the 122 points that ran to completion in this accuracy range, 85% used 20 or 25 hashes (with the remaining 15% split between 15 and 10 hashes). There also was an

Table 2
Database generation and retrieval parameters

Name	Description (values explored)	Intuition
Database generation parameters		
Spectral image length (w)	This parameter controls the length of each spectral image. Each spectral image is $32 \times w$ elements (there are 32 frequency bands) [1.48, 5.94 s]	Increasing this value provides a longer sample to compare each sub-fingerprint. However, length increases computation load and sensitivity to timing changes (e.g., linear-speed modification)
Sampling stride between spectral images in database (s)	After inserting a sub-fingerprint into our database, we skip over s seconds before another sub-fingerprint is inserted into our database [0.232–53.824 s]	The longer the sample stride, the less memory that is required since fewer sub-fingerprints are stored. However, the shorter the stride, the more completely the song is represented
Top-wavelet coefficients kept (t)	This is the number of top coefficients that are kept from the wavelets; the remaining wavelet coefficients are set to 0 [50–800]	The more coefficients that are kept, the more accurately the snippet can be reconstructed. However, the larger the t, the more noise is represented. t also interacts with the choice to use Min-Hash. The more non-zero values in the wavelet-vector, the less efficient the Min-Hash procedure becomes
Number of Min-Hash permutations (p)	To create a Min-Hash signature, the vectors are randomly permuted and the first non-zero element recorded. p is the number of times this procedure is done [50–100].	Increasing the number of permutations used to create a signature increases the signature's fidelity to the data that is being represented. However, this directly impacts memory requirements
Number of Hash tables (l)	This is the number of hashes that are computed. If l hash tables are used, then each hash bin is computed as a function of p/l values [10–25]	This has a direct impact on the memory requirements. The more hash tables, the more memory will be required, since a pointer to each snippet must be maintained with in each hash table
Number of hash bins (b)	This is the number of hash bins in the hash tables [50 000–10 000]	This can have a direct impact on memory and the number of elements that are in each bin. Note, however, that because the distributions are not completely random, the bins are unevenly filled
Retrieval parameters		
Probe snippet length (y)	This is the length of snippet that we are retrieving. [10,30 and 60 s]	The shorter the snippet, the more difficult the identification problem
Average probe sampling stride (d)	The time skipped, on average, between sub-fingerprints used to probe the database [46.4–2970 ms]	The longer the stride, the more likely that an important feature will be skipped. However, the shorter the stride, the more computation will be required
Minimum votes required from component hash tables (v)	With l total hash tables, each indexed sub-fingerprint can be retrieved between 1 and l times. v represents the minimum required for a full compare of the sub-fingerprint to occur [0–90%]	This directly impacts computation. The lower this value is, the more full comparisons are conducted. The higher this value is, the less noise-tolerant LSH becomes

unequal distribution across the numbers of retained top wavelets: 400 and 200 top wavelets accounted for nearly two-third of the run-to-completion points, with the remaining one-third split across 50, 100 and 800 top wavelets.

Second, the computational load (y-axis) is displayed logarithmically: the amount of computation for the boundary points marked varies significantly. Computation is measured by the number of full compares required. Recall that a *full compare* is when the p constituents of the query sub-fingerprint must be compared with the p constituents of a database candidate. This number is then multiplied by the length of the probe snippet, the y parameter. The graphed number is the total number of full compares required to recognize a snippet.

Third, in Fig. 7A, the best retrieval accuracy on the best operating curve (bottom left edge of the points in the graph) achieves 97.9% accuracy, while the best retrieval accuracy over *all* the parameter setting was only 0.2% higher on this probe set. That 0.2% increase in accuracy required twice the memory and nearly 2000× the computation; therefore, was not used in our final system. The operating point which yielded 97.9% accuracy was used (from this point on, we call the system with these parameters *Waveprint*-1); the parameters to obtain this accuracy were: spectral image length = 1.48 s, 5% top wavelets (t=200), l=20 hashes, s=0.9 s stride in DB creation, y=60 s queries; d=46 ms stride in probe; 7% minimum component hash vote (v=2); T=dynamic programming for temporal constraints. The long (60 s) probe snippet and short (46 ms) probe sampling stride giving the best results is expected, since the longer and more heavily sampled the query, the more information there is for accurate retrieval. Similarly, the best results coming from DTW is also as expected.

Fourth, we can reduce the computation by an order of magnitude with little drop in accuracy. If we look on the operating curve (Fig. 7A), the bottom labeled point achieves close performance results (97.5%) using 1/13th of the computation and only 15–20% more memory. The following parameters were used for this point (from this point on, we shall call the system with these parameters *Waveprint*-2): l=25 hashes and 17% minimum component hash vote ($v = 5$). The increase in the number of hashes accounts for the increased memory. This increase in the voting threshold accounts for the reduced computation.

Finally, we examine the performance of systems below the 80% threshold (shown in Fig. 7C). The parameters change substantially and the computation and memory requirements are reduced dramatically. For example, for a system with 80.2% accuracy, the computation can be reduced by more than two orders of magnitude from our best case, and memory reduced by 3x. Interestingly, the number of hash tables used is 25, so the memory reduction is not achieved through reducing the number of hash tables. Instead, this memory reduction was achieved by increasing the database stride (s) from 0.9 to 7.4 s—thereby reducing the number of stored sub-fingerprints. In addition, the computation was reduced by having a large probe stride ($d = 185$) and keeping the larger voting threshold (17%; $v = 5$ votes).

In the next section, we analyze the sensitivity of the parameters for the operating point with the best performance (*Waveprint*-1). In Section 5, we will return to examining the two points (*Waveprint*-1 and *Waveprint*-2) to examine two possible accuracy–computation–memory trade offs across a variety of song degradations and database sizes.

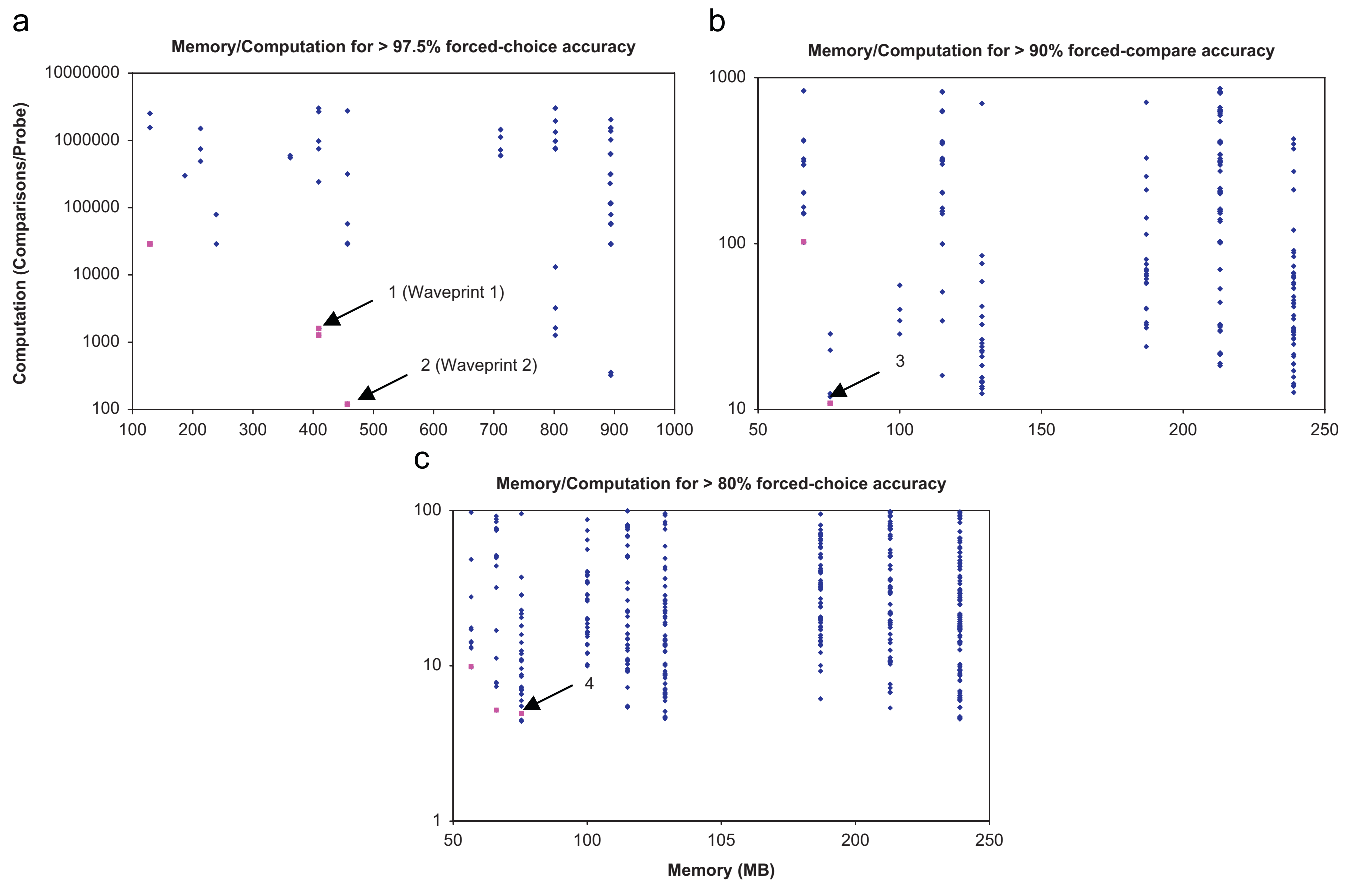

Fig. 7. Results for retrieval-accuracy settings > 97.5% (A), > 90% (B), and > 80% (C). The best operating cases are shown in gray (magenta in online version). *Waveprint*-1 *and Waveprint*-2 are labeled in (A). *X*-axis is the memory used in MB. *Y*-axis is a measurement of computation (the number of full-signature comparisons per probe). The striations seen are an artifact of the quantization of the parameter settings tested. A few of the interesting points on the performance boundaries are labeled with their performance and settings: 1: 97.9% (5% wavelets kept, 20 hashes, @0.9 s stride in DB creation, 60 s probe, sampled at/46 ms, 7% min votes) 2: 97.6% (5%,25h@0.9 s, 60s/48 ms, 17%*v*) 3: 90.0% (10%,25h@7.4 s,60s/46 ms, 17%*v*) 4: 80.2% (2.5%,25h@7.4 s,60s/185 ms, 17%*v*).

4.2. Parameter sensitivity

In this section, we examine the sensitivity of the settings chosen for each parameter. The base setting *Waveprint*-1 ($t = 200$, $l = 20$, $s = 0.9$, $y = 60$, $d = 46$ ms, $v = 2$ (7%), temporal on), is systematically changed in each dimension. Some of the more interesting results are shown here.

In our simplest analysis, we used all of the same parameter settings, but did not use dynamic programming to accumulate evidence in order to account for temporal coherence. This only dropped the performance from 97.86% to 97.12%, most likely due to the dense probe sampling. If the probe stride had been set longer (or fewer sub-fingerprints were stored in the database), temporal coherence would have yielded more information. Also, since changing the spectral image length (w) had little effect on performance, we kept it at the smallest value (1.48 s) for computational efficiency.

We next discuss three parameters with direct effects on the stored data distributions: the number of hash tables (l); the number of bins in each hash table (b); and the number of Min-Hash permutations (p) that were used in the l p/l-sized groups as inputs to a hash-key generating function for these b bins. In the final system, the number of hash bins, b, was set to 100,000. Changing this had only minor effects. For $b = 50,000$ and $b = 75,000$, the performance reduced slightly (to $\sim 97\%$) and computation increased (by $1.3\times$ and $1.1\times$, respectively). In contrast, reducing the number of Min-Hash permutations, p, dramatically increased the computation. Using $p = 75$ and 50 increased the computational load by more than $85\times$ and $1000\times$, respectively, because of the increased spurious collisions in the hash tables due to shorter hash keys. On the runs that did not exceed our "too-much-computation" termination condition, the accuracy remained largely unchanged. Fig. 8 shows the effects of the third parameter mentioned above: the number of hash tables (l). In general, the more hash tables used, the larger the likelihood of a match; however, more hash tables increase both computation and memory, as well as the possibilities for a false match. As can be seen here, the memory used is proportional to the hash tables used. However, the performance peaks with 20 hash tables; increasing the number of hash tables slightly degrades the performance. The lack of improved performance with 25 hash tables may be due to the increased chance of a false match that comes with looking at more of the full database of fingerprints.

In the best found setting, the probe length was set at 60 s. By reducing the probe length to 30 s, it is interesting to note that the accuracy drops by only 1.2% and reducing the length to 10 s drops the accuracy to 94.34%. The computation time drops proportionally to the length, as expected, since fewer sub-fingerprints must be matched.

In Fig. 9, we examine the effects of changing the minimum required votes (v). The most salient features in this analysis is the effects of setting the minimum number of votes to 1 (i.e., if a match is found anywhere in the bins, it is considered). This increases the computation by $1000\times$; although the accuracy does not change drastically, it does go down slightly. This indicates several important features about our system. First, the large increase in computation

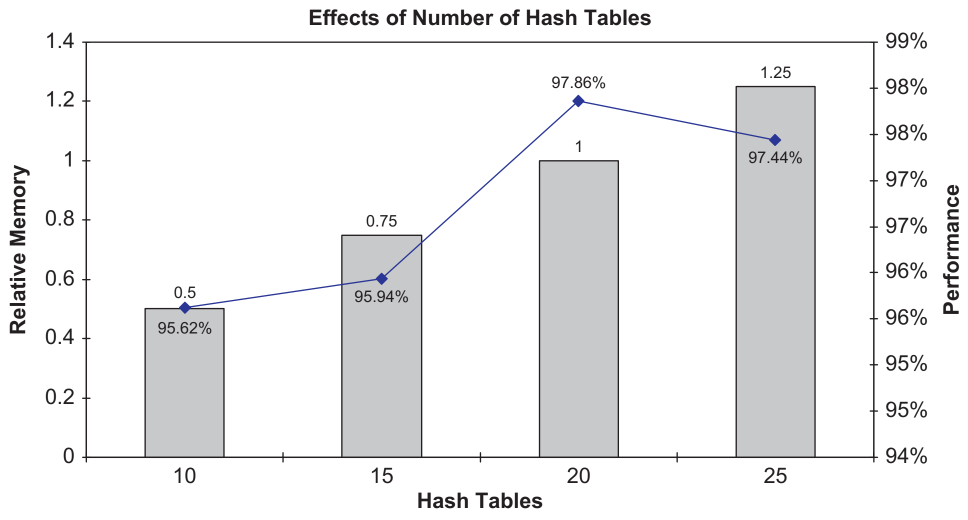

Fig. 8. Effects of number of hash tables (memory and performance) The line indicates accuracy, the bars indicate the relative memory usage. Hash tables = 20 is the setting used in the study.

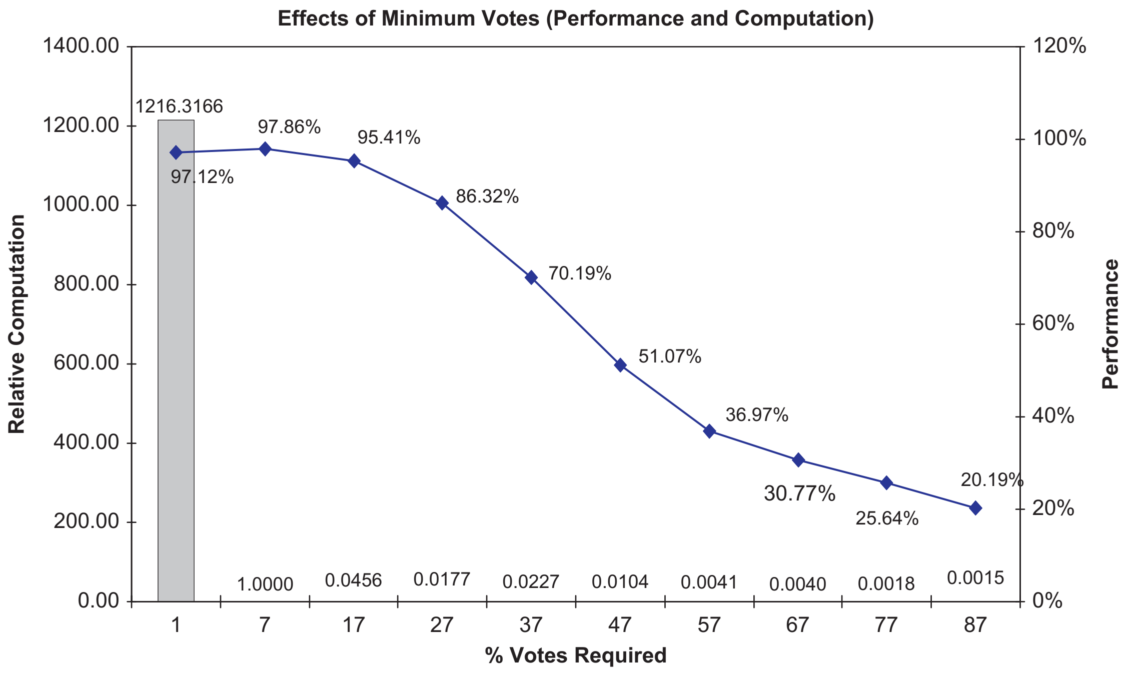

Fig. 9. Effects of the minimum number of votes required (computation and performance) The line indicates accuracy, the bars indicate the relative computation. Min-votes=7% is the setting used in the study.

indicates that there are many collisions in the database. We have verified this; the distribution of items in the bins is far from uniform (this is expected given the structured nature of songs). Second, the slightly reduced accuracy suggests that there are confounding sub-fingerprints, rather than simply random collisions. The voting provides a way to ensure that the similarity is above a certain threshold.

Finally, we examine the effects of sampling stride, for both database creation and for probes. It is interesting to note that reducing the computation by 50% by doubling the probe sampling stride results in only a small degradation in recognition accuracy. In fact, we can stay above 90% accuracy even when the sampling stride is increased by 16×. When looking at the database sampling stride, it is interesting to note that increasing the number of samples stored in the database (by reducing the stride) does not necessarily increase the performance of recognition. Both doubling and halving the database sampling stride reduce the performance by approximately 1.5%. As expected, doubling the database sampling stride reduces the computation and memory by 50%.

In summary, the results indicate that there can be large variations along the three dimensions of accuracy, memory and computation usage when the parameters are varied by large amounts. However, when changed independently, small changes in the parameter values will not change the performance drastically. Given the small performance changes that some of the parameter settings had, it is likely that similar parameter settings would work equally well in large real-world tests.

Table 3
Results with 10,000 Songs for Waveprint 1 ("WaveP 1"), Waveprint 2 ("WaveP 2"), and modified Ref. [10] ("Ke")

Degradation source	System	10 s query	30 s query	60 s query
Time-offset only	WaveP 1	100.00	100.00	100.00
	WaveP 2	100.00	100.00	100.00
	Ke	99.60	99.61	99.60
Echo-90%	WaveP 1	99.90	99.59	99.68
	WaveP 2	97.98	99.59	99.68
	Ke	99.00	99.11	99.25
Equalizer	WaveP 1	99.80	99.69	99.79
	WaveP 2	98.18	100.00	99.79
	Ke	99.13	99.14	99.30
MP3-32K	WaveP 1	97.47	98.76	99.15
	WaveP 2	97.37	98.44	98.50
	Ke	94.34	93.55	93.17
Noise-"Enya"	WaveP 1	86.97	95.96	98.40
	WaveP 2	85.45	96.17	98.51
	Ke	60.49	71.72	75.86
Noise-"Veil of Tears"	WaveP 1	82.29	84.37	85.18
	WaveP 2	81.01	84.16	85.25
	Ke	73.09	75.61	77.99
AMR: GSM Codec	WaveP 1	95.86	99.28	99.79
	WaveP 2	86.16	96.69	99.04
	Ke	94.87	96.97	97.65
LSM-102	WaveP 1	80.30	92.32	96.15
	WaveP 2	60.30	80.81	90.17
LSM-98	WaveP 1	93.74	95.56	96.95
	WaveP 2	90.81	94.21	94.78
TSM-90	WaveP 1	99.40	99.69	99.79
	WaveP 2	90.81	94.21	94.78
TSM-110	WaveP 1	99.09	99.69	99.78
	WaveP 2	98.89	100.00	99.68

5. System performance

With the two operating points, *Waveprint*-1 and *Waveprint*-2, we perform a comprehensive set of tests with a large number of signal degradations, and compare our system to that of [10].[1] Since all of our tests include an unknown time-offset in sampling, we were unable to compare ourselves directly with open-source Foosic libFooID [8] or open-source MusicBrainz Picard Tagger [9]. The results of our comparison to Ref. [10] are shown in Table 3. For each task, we lookup randomly selected, and degraded, samples from a database of 10,000 songs. We evaluate recognition accuracy with 10, 30 and 60 s snippets. The accuracy is the percentage of times the correct song was found as the top match, measured over 1000 trials for each task (33 total task were tried: 11 degradations * three probe lengths). The following signal degradations are evaluated on the probe set:

1. Time-offset only: no signal degradation, only time-offset is unknown. (Note that time-offset was also included in all of the tested cases with distortions 2–11.)
2. Echo: 90% of the original signal level and arrives 100 ms after original.
3. Equalizer: passes the signal through an equalizer with the settings of [18].
4. MPEG2 layer-3 at 32 kbps CBR: encodes and decodes the test probes.
5. GSM-adaptive multi-rate AMR at 4.75 kbps CBR: encodes and decodes to 4.75 kbps-mode AMR audio (cell-phone emulation).

6. and 7. Noise: adds Enya's "Watermark I" or To Die For's "Veil of Tears Epilogue" to the probe at a fixed volume. This ranged from $\frac{1}{2}$–2× the volume of the probe recording.

8. and 9. LSM: changedthe playback speed ±2% [30].

10. and 11. Time-scale modification (TSM): tempo ±10% without changing the pitch [30].

For each of the degradations, we attempt to retrieve 1000 samples. The *Waveprint* results shown in Table 3 display uniformly improved performance over the state-of-the-art system presented in Ref. [10]. The performance difference between *Waveprint*-1 and *Waveprint*-2 was less significant, with the best performance often switching between the two. However, as we will see in the next section, the scaling properties of the two are very different.

As can be seen by these results, the hardest case for the *Waveprint* system was the introduction of competing noise. We artificially set the structured noise fairly loud to make the case challenging. For many applications, however, in which more realistic, sporadic, noise is present (such as speech), we have found that good results are achievable in all the systems: for example, see Ref. [5] where the system from Ref. [10] was used as the audio-recognition component of a system built to recognize ambient-sound signals from a TV in a realistic home environment.

It is also interesting to note that severely degrading the signal via poor MP3 encoding, or passing the signal through AMR encoding, does not significantly change the detection accuracy when 30 s or longer are used for the recognition. At 10 s, the ability to recognize recordings through AMR degradation decreases to approximately 96%.

In the bottom 4 rows of Table 3, the results for the degradation through time-shifting are presented (LSM and TSM). We do not give the results for Ke's system [10] for time-varying distortions. In our use of Ke's system, it performed far below the Waveprint system when challenged with temporal distortion. Moreover, the performance of their system *degraded* with increasing snippet length, suggesting that their system did not include an extension to handle timing variations.[2] We see that pitch shifts can have a large impact on our recognition accuracy. This is expected, considering that shifting pitches will change the wavelets that are computed. Without pitch shifting (TSM-90 and TSM-110), recognition accuracy stays comparable to no degradation. However, when even a small amount of pitch shifting is introduced (± 2%), performance drops significantly. Haitsma and Kalker [18] also noted the same degradation in performance with pitch shifting. Two simple methods to solve this problem without introducing extra machinery are (1) to insert pitch-shifted versions of the songs into the database, in addition to the original version or (2) to attempt multiple queries for each song, varying the pitch shift for each query. The first method increases the memory; the second method increases the computation. These extensions are left for future work.

Finally, when we look at the computation and memory requirements of Waveprint, we again find a benefit, as described below. Memory usage is
$O(l * N * M/s) + O(l * b) + O(p * N * M/s) +$ Temporal_Constraint_Overhead_and_Bookeeping

- $O(l{*}N{*}M/s)$: This number is the number of pointers stored across all hash tables; these point to the actual sub-fingerprints. (M/s) is the number of sub-fingerprints that are created for each song. N is the number of songs, and l is the number of hash tables used. In practice, since these are pointers, they are 4 bytes.

[1] Ke's system is available on-line at the following address: http://www.cs.cmu.edu/~yke/musicretrieval/. Note that for these experiments, we modified their amplitude normalization by using a smoothly varying normalization computed on a sliding window of the surrounding 5 s. This uniformly improved the results from their system across all of our experiments.

[2] Since the extension of Ke's Bernoulli–Markov temporal model to include timing variations should be fairly straightforward, we chose to omit the artificially pessimistic performance numbers that we observed from this paper.

Table 4
Timing results for matching against a 10,000-song (35,000-min) reference set

Probe (s)	Time (s)	Speedup faster than real-time (assuming average song length of 3.5 min)
10	0.7	285.6
30	2.2	93.6
60	4.5	46.6

- $O(l*b)$: We used very simple hash tables that were implemented as arrays; two elements are stored in each bin of the array, a pointer to its contents and a count of how many sub-fingerprints are stored in the bin. In practice, since these are a pointer and an integer, they are represented as 8 bytes (total).
- $O(p*N*M/s)$: For every spectral image examined (for which there are $N*M/s$ of them), we need to keep a sub-fingerprint. The sub-fingerprint is p elements long (the number of permutations used in the min-hash signature). In practice, this is a value between 0 and 255, so it can be represented as 1 byte.
- Temporal_Constraint_Overhead_and_Bookeeping: There is a minor memory penalty when using temporal constraints to the dynamic programming.

To make this concrete, we can estimate the memory required for the tests performed in a straightforward implementation of the system (note that there are many modifications that can be made in the coding of the procedures which yield time-memory tradeoffs, but they are beyond the scope of this paper). We set the number of songs (N) to 10,000, the average song length (M) is 210 s, and the temporal constrain overhead and bookeeping peaks at approximately 5 MB. For 10,000 songs using *Waveprint*-1, $l=20$, $p=100$, $s=.928$ s, $b=100,000$ bins. The memory required is

$l*N*M/s$:	4 bytes $*\ 20*10,000*210/0.928$
$l*b$:	8 bytes $*\ 20*100,000$
$p*N*M/s$:	1 byte $*\ 100*10,000*210/0.928$
Overhead:	5 Mbytes
Total:	409 Mbytes

Doing the same analysis for *Waveprint*-2 ($l=25$ instead of 20), we get 456 Mbytes—also less than 0.5 GB of memory. Therefore, we can store about 45,100–50,500 songs on a single 2 GB machine without touching disk for retrieval. In contrast, the [18] system is reported to be able to handle 20,000 songs on a “modern PC”. The results that they report include creating 250,000,000 fingerprints for 10,000 songs of 5 min average length; this corresponds to 175,000,000 fingerprints if the songs were 3.5 min. Simply storing these pointers (or alternatively the fingerprints themselves @ 4 bytes per fingerprint) requires 667.6 Mbytes, this is approximately (30,700 songs in 2 GB memory).

A fairly detailed analysis is required for determining computational load; it is not presented here due to space restrictions. To summarize the results, we determined that *Waveprint*-1 and *Waveprint*-2 both have average case performances that are either lower than or within the bottom 1% of the range of performances expected in Ref. [18], in terms of the number of byte comparisons expected for their system. With respect to real-time performance, the speed achievable is dependent on the accuracy desired. The longer the sample snippet, the more reliable the recognition is. The timing results are shown in Table 4 (the tests were performed on a machine with a 3.4 GHz Pentium-4 CPU, 3 GB memory and 1 MB cache). These numbers reflect performance unoptimized for speed; it is possible to speed up the computing and sorting of the wavelets, by a factor of ~ 16–$32\times$ by reusing partial results across successive sub-fingerprints, since much of the computation is repeated across the time-window being examined.

Table 5
Scaling Waveprint reference set size

Trial	*Waveprint*-1: accuracy (relative computation)		*Waveprint*-2: accuracy (relative computation)	
	5000 songs	15 000 songs	5000 songs	15000 songs
Offset	100.0 (0.58)	99.79 (1.39)	100.0 (0.88)	100.0 (0.98)
Echo	99.9 (0.54)	99.9 (1.43)	99.9 (0.97)	99.69 (1.06)
Equalizer	99.69 (0.56)	99.69 (1.43)	99.79 (1.06)	99.59 (1.01)
MP3	98.55 (0.58)	98.76 (1.43)	98.96 (0.99)	98.44 (1.08)
Noise	97.1 (0.52)	95.76 (1.47)	96.38 (1.00)	95.34 (1.03)
Noise2	84.58 (0.55)	84.99 (1.41)	84.26 (1.03)	84.16 (1.07)
AMR	99.79 (0.53)	99.59 (1.46)	97.41 (0.98)	96.58 (1.04)
LSM-102	92.84 (0.52)	90.98 (1.46)	79.46 (1.02)	79.77 (1.06)
LSM-98	95.56 (0.55)	95.66 (1.47)	94.21 (0.90)	94.52 (0.95)
TSM-90	99.8 (0.55)	99.29 (1.43)	99.9 (0.77)	99.49 (0.77)
TSM-110	99.9 (0.55)	99.58 (1.40)	99.79 (1.05)	99.79 (1.10)

6. Scaling waveprint

In our last experimental section, we examine the effect of the number of songs in the database on the computational performance of our system. To this point, we have maintained a database size of 10,000 songs to match previous studies. In this section, we examine the effects of both reducing and increasing the database size by a factor of $\frac{1}{2}$ and $2\times$, respectively. The memory requirements scale linearly with the number of songs in the database.

Table 5 shows the performance, in terms of computation and accuracy, for *Waveprint*-1 and *Waveprint*-2, respectively, as the number of songs in the database is changed. A few observations are given here. First, when the size of the database is scaled from 5000 songs to 15,000 songs, there is only a small effect on the recognition performance in any of the degradation tests. These changes are most likely due to the small size of the testing sets (1000 retrievals are represented by each entry): missing 1–2 samples will create the types of differences seen here.

The most salient point to be made is that, for *Waveprint*-1, there is a strong relationship between the size of the database and the computation required (relative to the 10,000-song system), but the dependence is quite different for *Waveprint*-2. For *Waveprint*-1, the relationship is almost linear. For *Waveprint*-2, there is little performance difference between 5000 and 15,000 songs. This is relative to a lower computational load to start with (from Fig. 7 note that *Waveprint*-2 takes less computation than *Waveprint*-1). The difference in the scaling of performance is largely due to the minimum-voting parameter (v). When a hash bin is retrieved, it contains pointers to the set of sub-fingerprints that were hashed into that bin. With l total hash tables, each indexed sub-fingerprint can be retrieved between 1 and l times; v represents the minimum count that is required for a full check of the sub-fingerprint to occur. In *Waveprint*-1, the number of votes required is fewer than in *Waveprint*-2; therefore, more full comparisons are being conducted than in *Waveprint*-1. However, by increasing the number of votes, it is possible to miss some of the correct candidates, which leads to the lower recognition rates. In *Waveprint*-2, the number of votes is high enough that the increase in the number of matches in a bin (due to an increase in the number of songs) does not effect how many matches gather enough votes to trigger a full comparison.

The number of full compares given here is an indication of the scaling factor of the system. Each compare is the simple computation of the Hamming distance of 100-element arrays (since the signatures are of length $p=100$). From Fig. 7, that the *Waveprint*-1 system requires under 2000 full compares and *Waveprint*-2 requires under 200 full compares for matching a full song, even when 60 s snippets are used. However, in terms of clock-time, these full compares comprise only a portion of the overall computation cost. A significant amount of the computation time is spent creating the signatures for the candidate songs. Reducing the wavelet computation time (through the methods described earlier) will have a large impact on the user-perceived clock-time.

Finally, on a practical note we simply point out that the constancy in recognition accuracy across the database sizes is vital; the scaling issues in terms of real computation time can easily be addressed with the addition of more machines (as would be required to keep all the songs in memory). The architecture of this system lends itself to simple parallelization where more songs can be added to new machines without needing to update/change previously installed machines. The only architectural change to this system with the distribution of songs across multiple machines is the addition of a single "master server" which gathers the best of the matches that each machine has found and returns the single best match.

7. Conclusions and future work

In this work, we have presented the *Waveprint* audio identification system. The system builds on the insight of Ref. [10]: the task of audio recognition can be effectively addressed through computer-vision techniques. In this work, we applied the computer-vision work presented in Ref. [22] in retrieving near-duplicate images from a large corpus of image data to the task of audio retrieval. The resulting system is more efficient in terms of memory usage than the state-of-the-art competing system while providing better recognition accuracy. The system also provides good scaling characteristics. When the database size is increased by 50%, we see that we can have a sub-linear computation increase while having no significant impact on recognition.

Looking forward, there are at least five directions for immediate exploration. First, we will expand this system to open-set detection, in which we must determine whether a song occurs in the database (not which one of the database to which it is most similar). We have started this work by applying classifiers [31–33] to the results of *Waveprint*, as well as using additional temporal constraints.

A second direction is to examine alternative solutions to alignment uncertainty. The standard method of overcoming alignment uncertainty in audio processing is to oversample relative to the expected rate of change. A factor of 16–32 oversampling has been used in this and earlier work in audio fingerprinting [10,18]. For similar alignment ambiguities in image and video [34], explicit alignment is often useful. Finding ways to use a similar pre-alignment step could dramatically reduce the required degree of oversampling.

Third, we should consider the distribution of elements in the hash bins. The current, uneven distribution has an impact on computation and accuracy. While exploring different hashing functions can have an affect on this, larger gains can be achieved by intelligently selecting which signature elements are grouped into hash keys. Recall that, in the current system, each key is created by finding the first non-zero element after a random permutation has been applied to a sparse vector. Some permutations may yield correlated answers; therefore, intelligently selecting which sets of elements to include in a key, perhaps simply by minimizing their correlation, is open for future exploration.

Fourth, we need a method to determine wavelet stability. In this study, we retained the *top-t* wavelets and discarded the rest. The value of *t* is a compromise between selecting too few wavelets (the top ones are likely to be the same for many songs), and selecting too many (where the bottom ones are subject to noise). Exploring methods through which we can automatically ascertain which wavelets are stable to the types of distortions that we expect to encounter will help automatically set this parameter.

Fifth, the techniques developed here for processing audio were a combination of ideas from computer-vision literature with the ideas from large-scale data-stream processing literature. It is likely that some of the techniques will be useful in video and image retrieval tasks as well. We are currently exploring retrieval of non-music audio data through these same techniques.

References

[1] Gracenote Press Release, Gracenote global media ⟨http://www.gracenote.com⟩, 2006.
[2] Shazam, Shazam entertainment limited ⟨http://www.shazam.com/⟩, 2006.
[3] Google Video Team, About Google Video ⟨http://video.google.com/video_about.html⟩, 2005.
[4] YouTube Press Release, YouTube opens Internet Video to the Masses ⟨http://www.youtube.com/press_room_entry?entry = OcN9xXYar1g⟩, 2005.
[5] M. Fink, M. Covell, S. Baluja, Social- and interactive-television applications based on real-time ambient-audio identification, in: EuroITV, 2006.
[6] M. Covell, S. Baluja, M. Fink, Advertisement replacement using acoustic and visual repetition, in: IEEE Multimedia Signal Proceedings, 2006.
[7] J.S. Seo, J. Haitsma, T. Kalker, Linear speed-change resilient audio fingerprinting, in: Proceedings of IEEE Workshop on Model Based Processing and Coding of Audio, MPCA-2002, 2002.
[8] Foosic, libFooID Free Fingerprinting Library ⟨http://foosic.org/libfooid.php⟩, 2008.
[9] MusicBrainz, Picard, the next generation Musicbrainz tagger ⟨http://musicbrainz.org/doc/PicardTagger⟩, 2008.
[10] Y. Ke, D. Hoiem, R. Sukthankar, Computer vision for music identification, in: Proceedings of Computer Vision and Pattern Recognition 2005, pp. 597–604.
[11] S. Baluja, M. Covell, Audio fingerprinting: combining computer vision & data stream processing, in: Proceedings of the 2007 International Conference on Acoustics, Speech, and Signal Processing, 2007.
[12] P. Cano, E. Batlle, E. Gomez, L. Gomes, M. Bonnet, Audio fingerprinting: concepts and applications, in: Studies in Computational Intelligence (SCI), vol. 2, 2005, pp. 233–245.
[13] P. Cano, E. Batlle, T. Kalker, J. Haitsma, A review of algorithms for audio fingerprinting, in: International Workshop on Multimedia Signal Processing, 2002.
[14] Audible Magic, Music and video identification empowers online media service providers ⟨http://www.audiblemagic.com/products-services/contentsvcs/⟩, 2008.
[15] Barcelona Music and Audio Technologies, Match: track every bit of sound ⟨http://www.bmat.com/match⟩, 2007.
[16] Gracenote, Gracenote Mobile Music Identification ⟨http://www.gracenote.com/business_solutions/mobileMusic/⟩, 2008.
[17] Mediaguide, Mediaguide technology ⟨http://www.mediaguide.com/technology/⟩, 2008.
[18] J. Haitsma, T. Kalker, A highly robust audio fingerprinting system, in: Proceedings of International Conference on Music Information Retrieval, 2002.
[19] P. Viola, M. Jones, Robust real-time object detection, in: Proceedings of the CVPR, 2001.
[20] C. Burges, J. Platt, S. Jana, Extracting noise-robust features from audio data, in: ICASSP, 2002.
[21] C. Burges, J. Platt, S. Jana, Distortion discriminant analysis for audio fingerprinting, IEEE Trans. Speech & Audio Processing 11 (3) (2003) 165–174.
[22] C. Jacobs, A. Finkelstein, D. Salesin, Fast multiresolution image querying, in: Proceedings of SIGGRAPH 95, 1995.
[23] J. Deller Jr., J. Hansen, J. Proakis, Discrete-Time Processing of Speech Signals, Wiley, IEEE Press, New York, 1999.
[24] E. Stollnitz, T. DeRose, D. Salesin, Wavelets for computer graphics: a primer part I, IEEE Comput. Graphics Appl. 15 (3) (1995) 76–84.
[25] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J.D. Ullman, C. Yang, Finding interesting associations without support pruning, Knowl. Data Eng. 13 (1) (2001) 64–78.
[26] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, VLDB J. (1999) 518–529.
[27] J. Carter, M. Wegman, Universal classes of hash functions, in: Proceedings of the Ninth ACM Symposium on Theory of Computing, 1977, pp. 106–112.
[28] C.S. Myers, L.R. Rabiner, A comparative study of several dynamic time-warping algorithms for connected word recognition, Bell Syst. Tech. J. 60 (7) (1981) 1389–1409.
[29] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A^*, J. ACM 32 (3) (1985) 505–536.
[30] R. Veldhuis, H. He, Time-scale and pitch modifications of speech signals and resynthesis from the discrete short-time Fourier transform, Speech Commun. 18 (1996) 257–279.
[31] C. Cortes, V. Vapnik, Support vector networks, Machine Learning 20 (3) (1995) 273–297.
[32] T. Joachims, Making large-scale SVM learning practical, in: B. Scholkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge, MA, 1999.
[33] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, Berlin, 1995.
[34] M. Turk, A. Pentland, Eigenfaces for recognition, J. Cognitive Neurosci. 3 (1) (1991) 71–86.

About the Author—SHUMEET BALUJA, is currently a Senior Staff Research Scientist at Google, Inc., where he works on a broad set of topics ranging from image processing and machine learning to wireless application development and user interaction measurement. Shumeet was formerly the Chief Technology Officer of JAMDAT Mobile, Inc., where he oversaw all aspects of technology initiation, development and deployment. Previously, Shumeet was Chief Scientist at Lycos Inc., where he led the Research and Applied Technology Group in the quantitative and qualitative analysis of user behavior, including data mining and trend analysis, advertisement optimization and statistical modeling of traffic and site interactions. As Senior Vice President of R&D at eCompanies LLC, he spearheaded the creation of their wireless practice and was responsible for finding and evaluating new technologies and technology entrepreneurs. Shumeet has filed numerous patents and has published scientific papers in fields including computer vision and facial image processing, automated vehicle control, statistical machine learning and high dimensional optimization algorithms. Shumeet graduated from the University of Virginia with a Bachelor of Science, with high distinction, in 1991. He received his Ph.D. in Computer Science from Carnegie Mellon University in 1996.

About the Author—MICHELE COVELL received her Ph.D. from Massachusetts Institute of Technology in signal processing. She joined SRI International, in the area of active acoustic-noise control, and then Interval Research Corporation, where her research covered a wide range of topics in audio, image, and video processing. In 2000, she joined YesVideo and worked on faster-than-real-time video analysis. She moved to the Mobile Streaming Media group in HP Labs, as a key contributor in streaming-video services in 3G telephony networks. This work is listed as one of the top-40 accomplishments from HP Labs' 40-year history. She moved to Google Research in 2005, where she has focused on large-scale audio and video fingerprinting, identification, and retrieval.