Content Fingerprinting Using Wavelets

Shumeet Baluja, Michele Covell

Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA. 94043
{shumeet, covell}@google.com

Keywords: Audio Recognition, Fingerprinting, Wavelets

Abstract

In this paper, we introduce *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.

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 content-management systems, it can help follow the use of music and other audio material. This ability is increasingly important as more content is repurposed and recombined [6][13]. It provides the ability to automatically identify and cross-link background audio, such as songs. Tagging of songs with the performing artist's name, album or other metadata can be automatically accomplished [7][14].

Audio fingerprinting also enables numerous applications in the realm of enhanced television, from interactivity without imposing extraneous hardware constraints [4] to automatic advertisement detection and replacement [2]. Unlike many competing technologies, the goal of audio fingerprinting is to perform the recognition without the use of any extraneous information such as watermarks or pre/post transmission of inaudible (to the human ear) data.

There are a number of issues that make fingerprinting a challenging task. The simplest approaches, directly comparing the audio, or spectrograms of the audio, will not work. The query and stored version of a song may be aurally similar while having distinct bit representations. For example, they may be recorded at different quality settings, compression schemes, equalization settings, reference codecs, *etc*.; any of these factors would render naïve comparisons ineffective.

Numerous difficulties exist when moving to techniques that do not require exact bit-level matches. First, in many applications, the system must be able to function with only short snippets of audio content because often a full song will not be available. Second, since often only song fragments are given, in the vast majority of realistic scenarios, there is no fine or even coarse alignment of the audio content; it may occur anywhere within a song. Other difficulties arise in real-world usage. Often, songs are 'sampled' in other songs, thereby making the matches more ambiguous. When identifying songs played on a radio, a new set of difficulties arise because radio stations may change the speed of a song [12] to fit their programming requirements. Finally, there are difficulties introduced through the numerous 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.

2 Previous Work

Many audio-fingerprinting techniques use low-level features that attempt to compactly describe the audio signal without assigning higher-level meaning to the features. This approach is used in our study. Three of the most widely referenced approaches in this class are described here. One of the most widely used systems [8], uses overlapping windows of mono-audio from which to extract interesting features. Overlapping windows must be used to maintain time-shift invariance for the cases in which exact time alignment is not known. The spectral representation of the audio can be constructed in a variety of manners (by measuring the energy of the Mel-Frequency Cepstrum Coefficients (MFCCs) or Bark Frequency Cepstrum Coefficients (BFCCs)), BFCCs are used in their study. In their study, 33 BFCC bands are used that lie in the 300-2000 Hz range. Every 11.6 milliseconds, a sub-fingerprint is generated that covers a frame of 370 milliseconds. The large overlap in successive frames ensures that the sub-fingerprints vary slowly over time. The sub-fingerprints are a vector of 32-bits that indicate whether the difference in successive BFCC bands increases or decreases in consecutive frames. These sub-fingerprints are largely insensitive to small changes in the audio signal since no actual difference values are kept; instead, only the signs over consecutive frames compose the sub-fingerprint. Given this fingerprint (the sequence of sub-fingerprints), comparisons are simple; the difference between the frames is simply the Hamming distance of the fingerprints. The sub-fingerprints used in their study are compact and fast to compute.

A recent extension to the above work was presented in [10]. Ke used the same basic architecture of [8], but introduced a learning approach into the feature-selection process. An important insight provided by [10] is that the 1-D audio signal can be processed as an image when viewed in a 2-D time-frequency representation. The learning approach, based on AdaBoost, is often used in computer-vision applications such as face detection [16]. They presented a version of AdaBoost that learns features that integrate the energy in selected frequencies over time. The duration and frequencies are selected via the AdaBoost algorithm; they are similar to the "boxlet" features used in [16] (average intensities of rectangular sub-regions of the spectrogram image). The basis of selection for the features is the discriminative power of the rectangular region in being able to differentiate between when 2 frames are the same (when one is degraded by noise) and when they are different. 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 [8]. For lookup of new queries, their system processes the audio-image (similarly spaced to [8]), to create the 32-bit sub-fingerprint using the learned features. Then, all sub-fingerprints within a Hamming distance of 2 bits are searched for in the database. A measure of temporal coherence is provided by a simple transition model.

An alternate approach is explored in [1]; their work introduces Distortion Discriminant Analysis (DDA), a method to extract noise-tolerant features from audio. The features are more complex than in the studies by [8,10], but also summarize longer segments of audio than in that 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. In contrast, Principal Components Analysis finds a set of orthogonal vectors that maximize the signal variance. OPCA yields a set of potentially non-orthogonal vectors that account for noise statistics [1]. Their experiments have found that the fingerprints are resistant to problems with alignment and types of noise not found in the training set.

3 System Overview

Our system builds on the insights from [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 [9] for efficiently performing image queries in large databases. To make the algorithm scale, we employ the hashing work from the field of large-scale data-stream processing. The sub-fingerprints that we develop will be more comprehensive than used in either Haitsma or Ke's work since they will represent a longer time period, in a manner closer to the work presented in [1]. Each component of our system will be described in detail in this section.

We start our processing by converting the audio input into a spectrogram. We create our spectrograms using parameter settings that have been found to work well in previous audio-fingerprinting studies [8]. As a result, 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. One important consequence of the slice length/spacing combination of parameters (371ms slices each 11.6 ms) is that the spectrogram varies slowly in time, providing matching robustness to position uncertainty (in time). The use of logarithmical spacing in frequency was selected based for 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 this choice later in this section. Extracting known-length spectral images from the spectrograms allows us to create sub-fingerprints that include some temporal structure without being 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 directly comparing the "pixels" of the image, we will use a representation based on wavelets.

For each of the spectral images that we create, we extract the top wavelets according to their magnitude. Wavelets are a mathematical tool for hierarchically decomposing functions. They allow a function to be described by its overall shape, plus successively increasing details. Like Fourier decompositions, wavelets provide some degree of separation according to spatial frequency. Wavelets have the additional property of localized support, with each wavelet's support covering a consistent number of frequency cycles of that wavelet's frequency band. A good description of wavelets can be found in [15]. The motivation for using wavelets in audio-retrieval is based on their successful use in creating an image-retrieval system [9]. In the Jacobs system [9], rather than comparing images directly in the pixel space, they first decomposed the image through the use of multi-resolution Haar-wavelets. For each image, a wavelet signature of the image is computed; the wavelet signature is a truncated, quantized version of the wavelet decomposition of the image. Their system supported query images that were hand-drawn or low-quality sketches of the image to be retrieved. The results were better than those achieved through simple histogram or pixel differences.

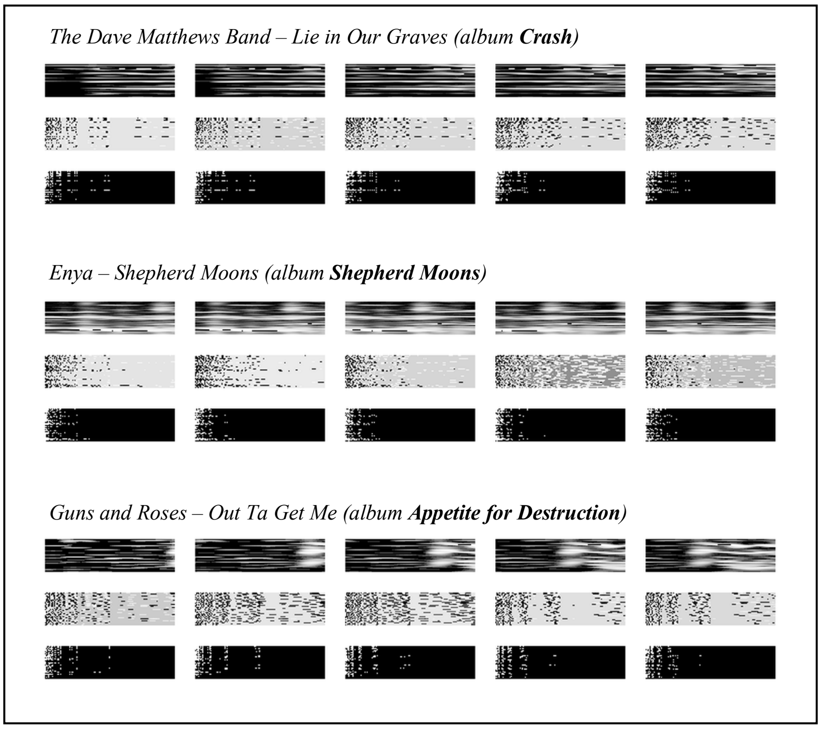


Figure 1. The representation for three songs – 5 consecutive frames shown for each, skipping 0.2 seconds. For each song, the top row is the original spectrogram image, the second row is the wavelet magnitudes, the third row shows the top-200 (*t*=200) wavelets. Note that the top wavelets have a distinctive 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).

To describe an $m \times n$ image with wavelets, $m \times n$ wavelets are returned: there is no compression. By itself, the wavelet-image is not resistant to noise or audio degradations; there will be changes to these values due to small changes in the sound (i.e. a small bit of noise, echo, other sounds in the background, being played over a cell phone, etc). Instead of using the entire set of wavelets, we only keep the ones that most characterize the song. We simply select the t top wavelets (by magnitude), where t << $m \times n$. 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 (see Figure 1).

One of the interesting findings in the Jacobs study was that they did not need to retain the coefficients for the top wavelets. Instead, they simply needed to retain the sign of the wavelet. This representation is used in this study, as memory usage must be carefully monitored. The most important feature of this bit vector is that it is sparse. Sparsity makes it amenable to further dimensionality reduction through the use of Min-Hash [18].

The final step of the sub-fingerprint creation process is to take the sparse wavelet-vector described above and create a compact representation of it. We explore the use of Min-Hash to compute sub-fingerprints for these sparse bit vectors. A fundamental requirement of the sub-fingerprints is 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.

For the purposes of this discussion, given two vectors *v1* and *v2*, we will refer to match types as being of four types *a, b, c,* and *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 bit positions will be of type *d*. We will define the similarity of two vectors to be the relative number of rows that are of type *a*: i.e., Sim(*v1*, *v2*) = *a*/(*a* + *b* + *c*).

An immediate approach to this problem is to simply randomly select a set of bit positions, and use them as the signature; however, that will not work. Because the vectors are sparse, the resulting signatures will likely be similar because they will mostly be composed of 0's; however, that will not give a true indication of similarity, because rows of type *a* are of most interest.

The Min-Hash technique works as follows. Permute the bit positions to some random (but known) re-ordering. Then, for that permutation, measure for each vector in which position the first '1' occurs. It is important to note that the probability that first_one_occurrence(*v1*) = first_one_occurrence(*v2*) is the same as the probability *a*/(*a* + *b* + *c*): the hash values agree if the first position with a 1 in either bit vector is of type *a*, and they disagree if the first such position is of type *b* or *c*. *Note this probability is the same as Sim(v1, v2) – which is what we need.*

We can repeat the above procedure multiple times, each time

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

with a new permutation of bit positions. If we repeat the process p times, with p different permutations, we get p 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 above. Instead, we store the Min-Hash computed signature; this is the final sub-fingerprint of the audio-image. A more thorough description of the process is given in [18]. Methods to make the matching process efficient, based on Locality-Sensitive-Hashing (LSH), are presented with the description of the retrieval process.

In this study, Min-Hash reduces the size of the signatures from the intermediate wavelet representation described in this section to a compact representation of p values. There are numerous other techniques that are commonly used for dimensionality reduction – among them techniques such as Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA). We chose to use 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 [1]. Since our top-wavelet signatures are already a sparse-vector representation, we decided to explore the use of techniques that were explicitly designed to handle probabilistic matching and discrimination across sparse vectors. Min-Hash is such a method and has been used extensively in data-stream processing. By employing this method, we avoid the continuous-valued modeling that would be required for LDA. This LDA conversion to continuous-valued modeling would require two transformations in our processing stream: from the top-wavelet signature bit stream to continuous values (for modeling) and then back to discretized values (for efficient nearest neighbor look up).

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 the top wavelets to only two bits. Third, we used the Min-Hash procedure to explicitly reduce the resulting bit-vector to p values, which became the final sub-fingerprint.

After these steps, each spectral image (or equivalent-length audio segment) 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) [5]. It is not only efficient in the number of comparisons that are required (a small fraction of the dataset will be examined), but also provides noise-robustness properties.

In contrast to more standard hashing, 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 point into l separate hash tables, each hash table using one of the subvectors as input to the hash function. Candidate neighbors can be efficiently retrieved by partitioning the probe feature vector 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 some minimum number of votes, v. If $v = 1$, this takes the union of the candidate lists. If $v= l$, this takes the intersection.

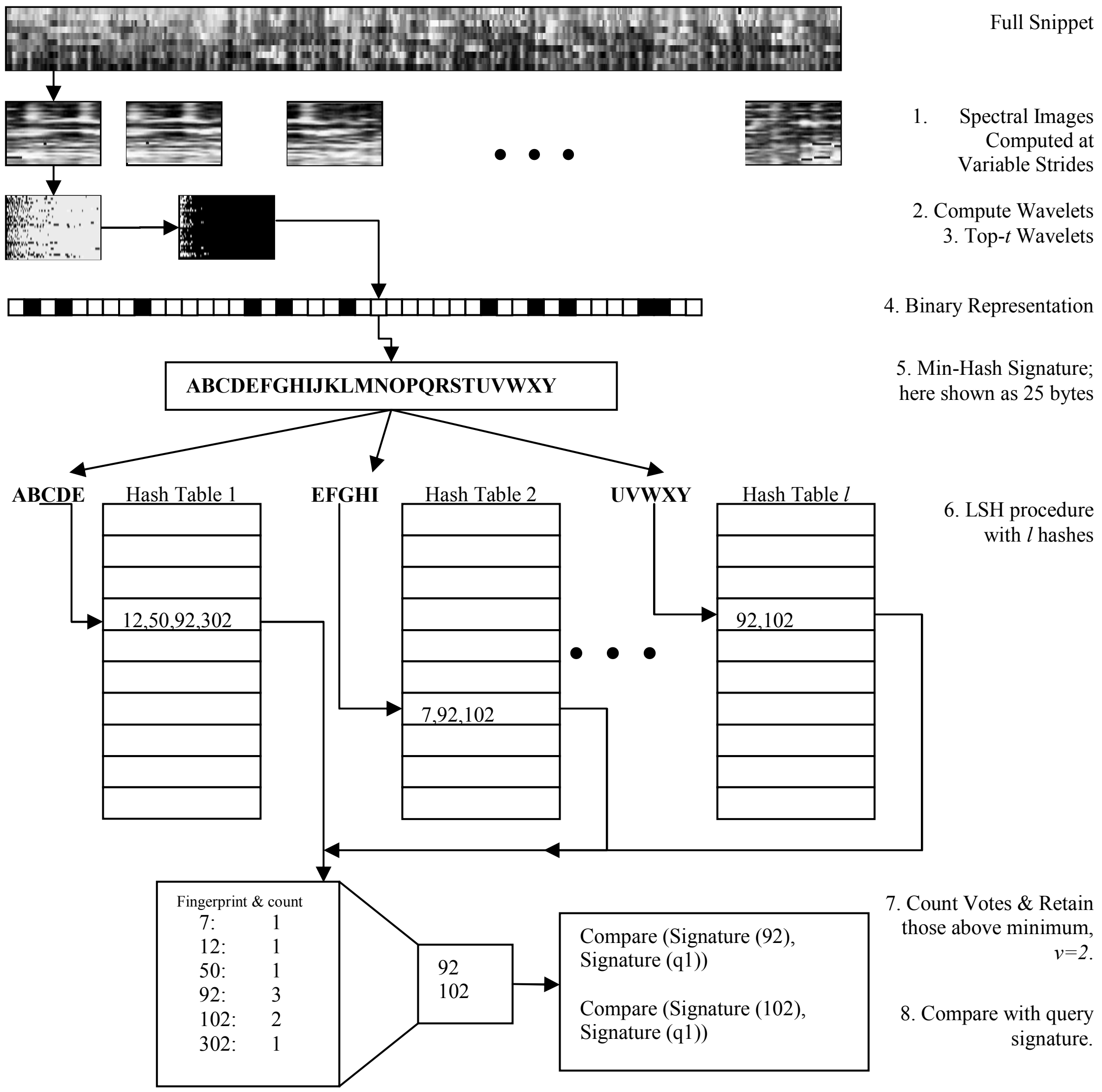
3.1 Retrieval

The overall retrieval process is shown graphically in Figure 2. The first difference in the retrieval process, in comparison to the database-generation process, is that the song is divided into randomly overlapping segments rather than uniformly overlapping segments. Randomly selecting the stride amount is important to avoid problems of unlucky alignments; if the sampling of the probe is kept constant, it may be possible to 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 6-8 describe an efficient mechanism for finding matches in the database and measuring their distances from the query, and are the subject of the next section.

We described the basic characteristics of LSH in the previous subsection. LSH also supports flexible constraints on which candidates from the individual component hash tables will be examined further, as part of the final list of candidate matches. This flexibility is achieved by adding a requirement for a minimum number of component hash-table votes. Under this development, 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 would be 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. 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 on that spectral image.


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

1. Given the audio spectra of a song, extract spectral images of 11.6**w* ms duration, with random spacing averaging *d*-ms apart.

For each spectral image:

2. Compute the wavelets on the spectral image
3. Extract the top-*t* wavelets.
4. Create a binary representation of the top-*t* wavelets.
5. Use min-hash to create a sub-fingerprint of the top-*t* wavelets (note that the same permutations used in the database-creation portion are used here)
6. Using Locality Sensitive Hashing, with *b* bins, *l* hash tables, find the sub-fingerprint segments that are close matches.
7. Discard the sub-fingerprints with less than *v* matches
8. Compute the Hamming distance from the remaining candidate sub-fingerprints to the query sub-fingerprint.
9. Use Dynamic Programming to combine the matches across time.

Figure 2. Overall architecture for the retrieval process. Step 9 not shown in the diagram.

3.1.1 Temporal Ordering Constraints

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

The simplest way to combine evidence is a simple voting scheme that does not take into account 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, at each probe sub-fingerprint, for each candidate that passed the required hash-voting threshold, 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+Δt) within the probe are rewarded equally for (respectively) matching song sub-fingerprints s(t) / s(t–Δt) as they are for matching s(t) / s(t+Δt): there is no penalty for time reversal. Similarly, for matching s(t) / s(t-+10Δt): there is no penalty for changing tempo. Even with these limitations, this simple voting approach is worth considering for retrieval tasks with overlapping temporal sampling in the database and with dense temporal sampling in the probe snippet.

We also explored using dynamic-time warping to accumulate evidence across time within a database song. Dynamic-time warping is a form of dynamic programming for imposing "tempo" constraints in mapping one sequence onto another [11]. 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 (over the length of the match, there can be only 10% change in tempo from the probe to the candidate match) and local-slope constraints (no single probe can match more than one song location within a single track and no local-time inversions are allowed).

This leaves unaddressed the question of selecting the starting correspondence. Each probe sub-fingerprint can propose multiple matches in any given song and any of these can be the correct starting correspondence. In addition, since we allow matches to be missing, the simple time-synchronized dynamic-program possible-state list grows over time. To avoid uncontrolled growth in the use of memory (and computation) to track all of these possibilities, we impose two limits. First, we only allow each sub-fingerprint to propose approximately twenty potential matches for itself, across the full data base of songs. This limit is only approximate since, if there are a set of identical-quality (according to Hamming distance) candidate matches that include one or more of the top 20 matches, then all of these ties are allowed as candidates. In the opposite direction, if fewer than 20 matches passed our previous criteria for sub-fingerprint matches, then only that smaller set is considered. Secondly, within each song, we use A* pruning on list of current hypothesized time sequences.

Doing A* pruning on unequal length partial matches requires a quality-of-match measure that has the length of that partial match largely normalized out. For example, we do not want to prune out a new match track that has only a few but strong matches, due to the presence of a uniformly mediocre match that extends over large number of sub-fingerprint. As a result, we selected a figure-of-merit that is the similarity of the best sub-fingerprint match in the sequence plus the average across all other sub-fingerprint matches within the sequence. This reduces to the sub-fingerprint match score in the case of a single-length match track and to twice that value if all the sub-fingerprints are equal strength, but it otherwise does not grow with track length changes. Our figure-of-merit also includes a small penalty for global tempo changes (within the allowed ±10%). In summary, the score for each song is:

- For accumulation without temporal constraints: The sum of the Hamming similarity of the candidate sub-fingerprints within that song in the database.
- For accumulation with temporal constraints: The score of the best temporal track within each song (as described above), where sub-fingerprint matches within each temporal track must:
  - not introduce local time inversions (no backtracking within the database-song),
  - not match a single probe sub-fingerprint to more than one database-song sub-fingerprint (the opposite is allowed, due to unequal sampling rates),
  - not include probe-database sub-fingerprint pairs that, when measured along the database-song axis, lie outside more than a database sampling stride outside of the ±10% tempo cone as defined by the starting probe-database sub-fingerprint pair.

4 Experiments

One of the intrinsic difficulties in designing a large-scale system is conducting a thorough exploration of the parameter settings. In this section, we report the results of extensive testing of parameter sets. Over 50,400 different parameter combinations were tried to ensure that we select the best settings and understand the tradeoffs with each parameter.

To explore this large parameter space, we used a 10,000-song database, with an average song duration of 3.5 minutes. Since our goal at this point is to understand the general system performance, we did *not* use heuristics such as unequal protection over time (i.e., protecting song beginnings and choruses more heavily) to reduce the amount of memory usage or computation.

We used 1000 independent probe snippets into this song data base. Each of these probes included one of the following distortions, with each distortion getting equal representation:

1. Time-offset only: This uses a clean signal as the probe but with an unknown starting time offset, relative to the database sampling. This unknown sampling offset is also included with all the other distortions, below.
2. Echo: The added echo retains 90% of the original signal level and arrives 100 ms after the original sound.
3. Equalization: This process effectively passes the signal through an equalizer with the settings of [8], which double the volume on some frequency bands at the same time as halving the volume on others.
4. MP3-32 Kbps: We encode and decode the test probes at 32-kbps using constant bit-rate MPEG2 layer-3 audio.
5. GSM-Adaptive Multi-Rate (AMR): We encode and decode the test probes to 4.75-kbps-mode GSM AMR audio.

6-7. Noise: This adds structured noise (Enya's *Watermark I* or To Die For's *Veil of Tears Epilogue*) to the probe at a fixed (probe-content-independent) RMS-volume.

8-9. Linear Speed-up Modification: We simply change the playback speed of the sound, speeding it up, or slowing it down by 2%.

10-11. Time-Scale Modification: This increases/decreases the tempo by 10% without changing the pitch.

In Section 4.1, we report retrieval results on a forced-choice task (this assumes that the song exists somewhere in the database). The results reflect the percentage of times we selected the correct song from the database, using the distorted probe snippet. On this task, since we are operating against 10,000 equally probable songs, random chance is 0.01% correct.

4.1 Empirical Results

With over 50,400 parameter settings, and three interesting attributes (retrieval accuracy, memory usage, computational load), there are numerous manners in which to report the results. In the following 2 graphs, we present the results on the recognition task for 2 different retrieval-accuracy settings. The results near the *best operating curve* are shown in Figure 3. This best operating curve is the set of points which, for the selected retrieval accuracy, requires the least memory for its computational-load operating point and uses the least computation for its memory-usage operating point. In the 2 graphs shown in the figure, we restricted the computation and memory range to be close to the best operating curve (for the selected accuracy) but left all experimental results that fell within the shown range of values, even if they were not on that operating curve.

There are many interesting points to note about the results. First, corresponding to the graph showing the best accuracy (Figure 3, top) only 122 out of the 50,400 experiments had accuracies of 97.5% or above. As many as 320 additional parameter combinations might have achieved accuracies in this range but were terminated early, due the impractically large amount of computation that they were consuming (more than 3 million comparisons per probe snippet). Of the 122 points that ran to completion, 85% used 20 or 25 hashes (with the remaining 15% split between 15 and 10 hashes). There also was an unequal distribution across the numbers of retained top wavelets: 400 and 200 top wavelets accounted for nearly 2/3^{rds} of the run-to-completion points, with the remaining 1/3 split across 50, 100 and 800 top wavelets.

Second, note that the y-axis (showing the amount of computation) on all the graphs is logarithmic; therefore, the amount of computation for the 4 boundary points marked on Figure 3(top) varies significantly. Computation is measured by the number the number of full-compares required. A full-compare is when the *p* constituents of the query sub-fingerprint must be compared with the constituents of a database candidate. Note that this number is then multiplied by the length of the probe snippet required by the system, the *y* parameter. The final number, shown on the graph, is the total number of full compares required to recognize a snippet.

Third, the best retrieval accuracy on the best operating curve (Figure 3, top) 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 2000x 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 shall call the system with these parameters *Waveprint-1*); the parameters to obtain this accuracy were:

- 5% of the wavelets (t=200) were kept; *l*=20 hashes, *s*=0.9 seconds stride in DB creation; *y*=60 second queries; this is expected as the longer the query is, the more information there is for accurate retrieval; *d*=46 ms stride in probe; this is a small stride when querying; the smaller the stride, the more accuracy is expected; 7% of the hashes (*v*=2) had to vote for a snippet in order for it to be considered as a potential match.; T=Dynamic programming for temporal constraints; again, this is expected as it is another source of information.

Fourth, the next best results on the best operating curve (97.8%) took an order of magnitude more computation time. This was due to two factors. First, the number of hashes increase, and second, the sampling stride for the query was reduced; thereby increasing the number of comparisons that must be done.

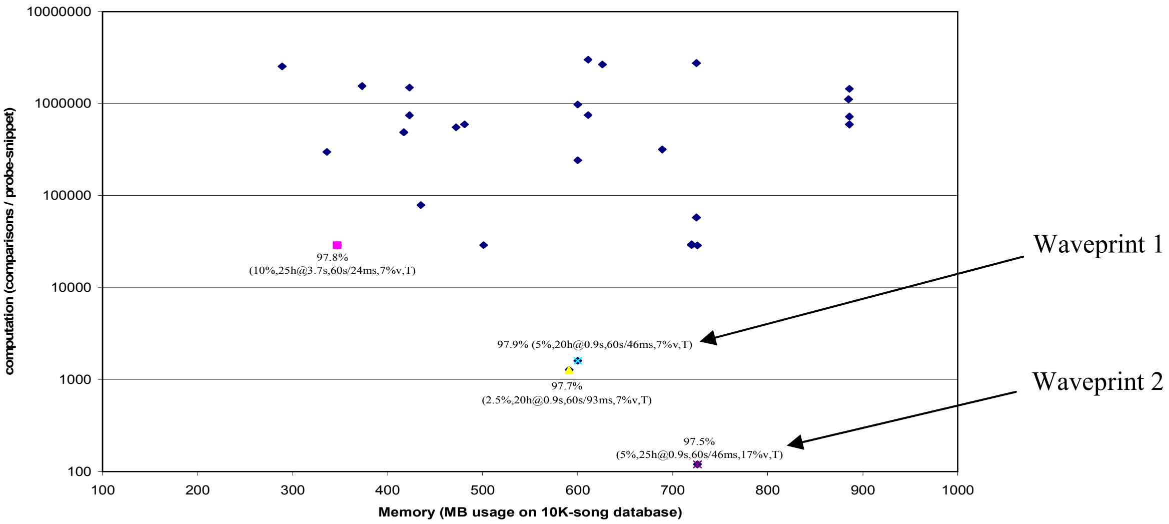
Fifth, another interesting point is that we can reduce the computation by an order of magnitude with little drop in accuracy. If we look on the operating curve (Figure 3, top), the bottom labeled point achieves close performance results (97.5%) using 1/13th of the computation and < 20% more memory. The following parameters were used for this point (from this point on, we shall call the system with these parameters *Waveprint-2)*:

- *l*=25 hashes; this increase in the number of hashes accounts for the increased memory; 17% of the hashes (*v*=5) had to vote for a snippet; this increase in the voting accounts for the reduced computation; all other parameters unchanged.

If we now look at the graph showing 50%-or-better performance (Figure 3, bottom), we see that the parameters have changed substantially and that the computation and memory requirements have been reduced dramatically. For example, looking at the case with 68.6% accuracy, note the computation has been reduced by almost 3 orders of magnitude from our best case, and memory reduced by 3x. Surprisingly, the number of hash tables that define the best operating curve for 50% accuracy use 25 hashes, 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 seconds to 7.4 seconds – thereby reducing the number of stored sub-fingerprints. In addition, the computation was reduced by having a large probe stride (d=186 ms) and keeping the larger voting threshold (17%; v = 5 votes).

Looking across both graphs, the number of retained wavelets seems to be consistently higher on the lower-accuracy best operating curves than what was seen for the 97.5%-accuracy best operating curve. On the 97.5%-accuracy curve, most of the best operating points retained 5% (200) of the wavelets. When examining similar curves at 80% and 90% accuracy levels (not shown here), all the best operating points keep 20% (800) of the wavelets, as do many of the best operating points on the 50% accuracy curves. The number of wavelets to keep is not a simple parameter to set and is dependent on the settings of many other parameters. If too many wavelets are kept, the sparsity of the binary vector is reduced; thereby rendering techniques like Min-Hash ineffective. If too few

Memory/Computation for > 97.5% forced-choice accuracy



Memory/Computation for > 50% forced-choice accuracy

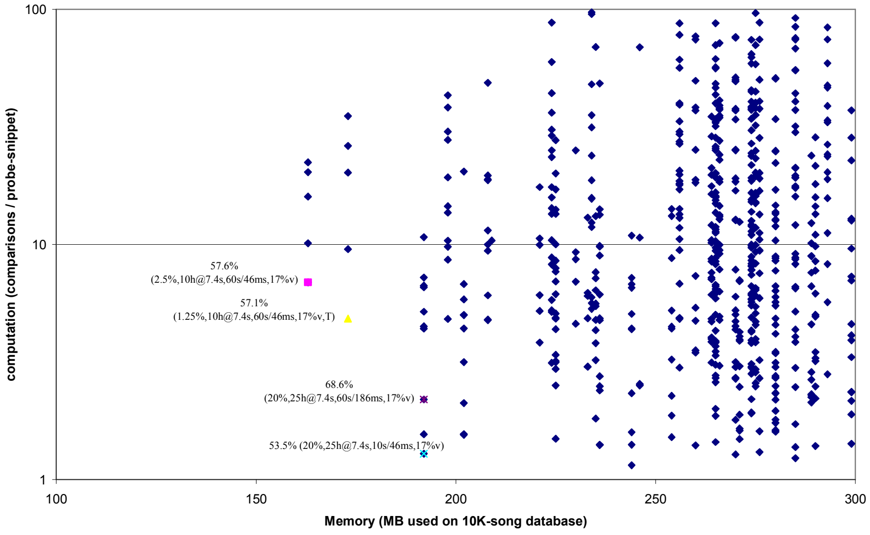


Figure 3. Results for retrieval accuracy settings > 97% (top) and > 50% (bottom). The best-operating cases are labelled. Two parameters are selected for further study: The first is at 97.9% accuracy with settings (t=200 retained wavelets, l=20 hashes, s=0.9-sec DB stride, y=60-sec queries, d=46-ms probe stride, v=7% voting, and temporal constraints); this is called *Waveprint-1*. The second has 97.5% accuracy with settings (t=200 retained wavelets, l=25 hashes, s=0.9-sec DB stride, y=60-sec queries, d=46-ms probe stride, v=17% voting, and temporal constraints); this is called *Waveprint-2*.

Table 2. Performance Comparison – Full Test Set

System	10 Sec	30 Sec	60 sec
Waveprint-1	94.3	96.5	97.9
Waveprint-2	89.5	96.2	97.5

wavelets are kept, the signature may not contain enough information about the underlying signal.

4.2 Comparisons

In this section, we analyze our system's performance. The retrieval performance of *Waveprint-1* and *Waveprint-2,* for a new test set, across three different probe lengths, is shown in Table 2.

For performance comparison, we use the extension to [8] that was developed by [10].[1] In addition to the exact system developed by Ke, we also tried a modification that simply changes the amplitude normalization to a smoothly varying normalization that is computed on a sliding window of the surrounding 5 seconds of audio. Unfortunately, their system was not designed to handle large timing variations, so we did not include time-based degradations in the next set of tests.[2] Instead, we created a new test set that replaced linear-time-scale modification and time-scale modification with other degradations. The results of all the systems on this modified-test set are given in Table 3.

In terms of memory, we can express our usage in terms of the parameters of the system. The number of songs stored in our database is N, the average length of a song is M.

Memory Usage =

$$O(l * N * M / s) + O(l * b) + O((p+\alpha) * N * M / s) + Temporal\_Constraint\_Overhead$$

Here, O ($l * N * M / s$) 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 represented with 4 bytes. 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).

Table 3. Performance Comparison – Test Set without Time-Scale- and Speed-Modification Degradation

System	10 Sec	30 Sec	60 sec
Ke - Original	80.1	83.0	85.0
Ke - Modified	84.8	88.1	90.0
Waveprint-1	93.8	96.4	96.9
Waveprint-2	90.8	96.4	96.9

O($(p+\alpha) * N * M /s$): for every spectral image examined, 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, each value is between 0-255, so it can be represented as 1 byte. α is bookkeeping storage to relate each fingerprint to its position into its song. *Temporal_Constraint_Overhead:* Although we will not explain this in great detail, there is a memory cost when using temporal constraints with dynamic programming. It is minor in comparison to the rest of the memory used elsewhere.

To make this concrete, we can estimate the memory required for the tests performed for a *memory-optimized* 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. Some were used in the experiments described in Figure 3 – which rendered it *not* memory optimal).

For *Waveprint-1*, we used l = 20, p = 100, s = .928 seconds, b=100,000 bins. We assumed a database of 10,000 songs, of average length 3.5 minutes. We estimated the *Temporal_Constraint_Overhead* at 10 megabytes. This yields a total of approximately 0.45×10^9 bytes of memory. Therefore, on a standard 2GB machine, we can store approximately 47,000 songs without touching disk for retrieval.

Doing the same analysis for *Waveprint-2* (l=25 instead of 20), and keeping the *Temporal_Constraint_Overhead* at 10 megabytes, we get 0.50×10^9 bytes of memory. Therefore, on a standard 2GB machine, we can store approximately 43,000 songs without touching disk for retrieval.

Next, we describe the speed of our system. The speed achievable is dependent on the accuracy desired: The longer the sample snippet, the more reliable the recognition is but the longer the processing takes. The timing results are as follows (measured by how much faster than real-time): 10-sec probe, 286×faster; 30 seconds, 94×faster; 60 seconds, 47×faster. [3] These timing results *do not* include the time required to create the spectrogram. Further, it should be possible to speed up the most computationally expensive portion of the process (computing and sorting the wavelets, which account for approximately 90% of the cost) by a factor of ~16-32x. This

[1] Ke's system can be downloaded from: (http://www.cs.cmu.edu/~yke/musicretrieval/).

[2] Since an extension of Ke's Bernoulli-Markov temporal model to include large timing variations should be possible, we omitted the pessimistic performance numbers that we observed of the [10] default system.

[3] The machine tests were performed on a 3.4 GHz Pentium-4 CPU, with 3 GB memory and 1 MB cache.

can be done by reusing partial results across successive sub-fingerprints, since much of the computation is repeated across the time-windows examined.

5 Conclusions & Future Work

In this work, we have presented the *Waveprint* audio identification system. The system builds on the insight of [10]: the task of audio recognition can be effectively addressed through computer-vision techniques. In this work, we extended the computer-vision work presented in [9] for retrieving near-duplicate images from a large corpus of image data to the task of audio retrieval. The accuracy of the resulting system remains high even when tested on severely degraded probe samples.

Immediate next steps include scaling the database. We have seen preliminary promising results, both in terms of accuracy and speed, especially in the *Waveprint-2* setting. Other future work includes exploring applications beyond music matching, such as using the system for matching television broadcasts. Finally, automatic methods for ascertaining and using the stability and distribution of points within the hash-bins and the top-wavelets are being explored.

Acknowledgements

We would like to acknowledge the help of Yan Ke, Derek Hoiem, and Rahul Sukthankar for providing their code for comparison. Additionally, this paper has benefited from conversations with many people, including Mayur Datar, David 'Pablo' Cohn and Sergey Ioffe.

References

[1] C. Burges, J. Platt, S. Jana (2003). Distortion Discriminant Analysis for Audio Fingerprinting, *IEEE Trans. PAMI* 11, (3).

[2] M. Covell, S. Baluja, M. Fink (2006). Advertisement Replacement using Acoustic and Visual Repetition, *Proc. IEEE Workshop on Multimedia Signal Processing.*

[3] J. Deller, Jr., J. Hansen, J. Proakis (1999) *Discrete-Time Processing of Speech Signals* Wiley-IEEE Press.

[4] M. Fink, M. Covell, S. Baluja (2006). Social- and Interactive-Television Applications Based on Real-Time Ambient-Audio Identification, *Proceedings of EuroITV.* .

[5] A. Gionis, P. Indyk, R. Motwani (1999), Similarity search in high dimensions via hashing. *Proc. International Conference on Very Large Data Bases*,.

[6] Google Video Team (2005). About Google Video. http://video.google.com/video_about.html.

[7] Gracenote Press Release (2006). Gracenote Global Media Database used by over 150 Million Music Fans Worldwide, Performing over 6 Billion Music Searches in 2005. http://www.gracenote.com/music/corporate/press/article.html/date=2006010502

[8] J. Haitsma, T. Kalker (2002). A Highly Robust Audio Fingerprinting System. *Proc. International Conf. Music Information Retrieval*.

[9] C. Jacobs, Finkelstein, A., Salesin, D. (1995) Fast Multiresolution Image Querying. *Proc. SIGGRAPH,*

[10] Y. Ke, D. Hoiem, R. Sukthankar (2005). Computer Vision for Music Identification. *Proc. Computer Vision and Pattern Recognition*.

[11] C. S. Myers, L. R. Rabiner (1981). A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60(7):1389-1409.

[12] J.S. Seo,, J.Haitsma, T. Kalker (2002). Linear Speed-Change Resilient Audio Fingerprinting. *Proc. IEEE Workshop on Model based Processing and Coding of Audio.*

[13] G. Sing (2004). Content Repurposing. *IEEE Multimedia* 11(1).

[14] Shazam (2006). Shazam Entertainment Limited. http://www.shazam.com/

[15] E. Stollnitz T. DeRose D. Salesin (1995). Wavelets for Computer Graphics: A Primer Part I. *IEEE Computer Graphics and Applications, 15(3).*

[16] P. Viola & M. Jones (2001). Robust Real-time Object Detection. *Proc. International Conf. Computer Vision.*

[18] E. Cohen, *et al.*. (2001) Finding interesting associations without support pruning. *Knowledge and Data Engineering*, 13(1).