DUPLICATE DETECTION AND AUDIO THUMBNAILS WITH AUDIO FINGERPRINTING

Christopher Burges, Daniel Plastina, John Platt, Erin Renshaw, and Henrique Malvar

March 2004

Technical Report
MSR-TR-2004-19

Audio fingerprinting is a powerful tool for identifying either streaming or file-based audio, using a database of fingerprints. This paper presents two new applications: duplicate detection, whose goal is to identify duplicate audio clips in a set, even if they differ in compression quality or duration, and thumbnail generation, which aims at providing a representative short clip of a music track. Each application is self-contained in that it does not require an external database of fingerprints. Thanks to the robustness of the fingerprinting engine, both applications perform well.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1. INTRODUCTION

Audio fingerprinting (AFP) has recently emerged as a powerful method for identifying audio, either in streams or in files [1]. Several companies now offer music services based on audio fingerprinting. These services require that one or more fingerprints be extracted from the audio to be identified, and that these fingerprints be checked against a large database of previously-computed fingerprints. In this paper, we explore two new AFP applications: duplicate detection and audio thumbnail generation. In duplicate detection, we aim to identify duplicate audio files based only on the audio data, even if one is a noisy version of the other or if they have different durations. In audio thumbnail generation, the task is to find a short (we use 15 seconds) representative section of the music – a "thumbnail." Duplicate detection is useful for automatically cleaning large audio databases, e.g. to help users identify duplicate copies of songs on their PCs, and audio thumbnails facilitate audio browsing.

We build these two applications using the RARE (Robust Audio Recognition Engine) AFP system [2], which converts a segment of audio to 64 floating-point numbers (a fingerprint). RARE has two main features: its fingerprints are very robust to distortions of the original audio [2], and the AFP lookup method uses a new technique that is about a factor of 50 faster than the fastest competing method [3]. For each created fingerprint, a normalization factor is also created, so that the mean Euclidean distance from that fingerprint to a large collection of fingerprints computed from other audio is one. We refer the reader to [2, 3] for details. In the following, "trace" means any kind of fingerprint extracted from audio, and "fingerprint" means a reference fingerprint against which traces are compared to determine the audio identity. The normalization factor is always associated with the fingerprint, so Euclidian distances $D(\cdot,\cdot)$ between traces and fingerprints are normalized.

2. THE RARE DUPLICATE DETECTOR

The RARE duplicate detector *DupDet* works as in the basic diagram of Fig. 1, recursively processing all audio files in a directory tree. It creates a set of traces for each file, and checks it against a set of fingerprints created for the other audio files. If $D(\cdot,\cdot)$ between a trace and a fingerprint falls below a threshold [2], the associated audio files are declared to be duplicates. For each file, the fingerprints are computed at a fixed location L in the file, and the traces are computed in a search window W around L; L and W are user defined.

DupDet can simultaneously create fingerprints and check for duplicates in one pass, as indicated in Fig. 1. When the first audio file is read, a 6s fingerprint at location L is computed and saved. When the second audio file is loaded, traces that begin in the window $W - L$ to $W + L$ are computed in time order (each trace computed 1/6 s after the previous one). If one of these traces is a match, the file is declared a duplicate and added to the list of duplicates for that fingerprint. No external database of fingerprints is needed, and the amount of data loaded at run time is of order 2 MB. If no match is found for the entire set of traces in the search window, then the fingerprint (already compute at location L in the audio) is saved in the database, representing a (so far) unique clip. Finally, the system also uses 6 'veto fingerprints,' which are fingerprints collected from noise (e.g. silence, sound cards with no input, etc.) Audio files that match a veto fingerprint can also be labeled as 'junk files.'

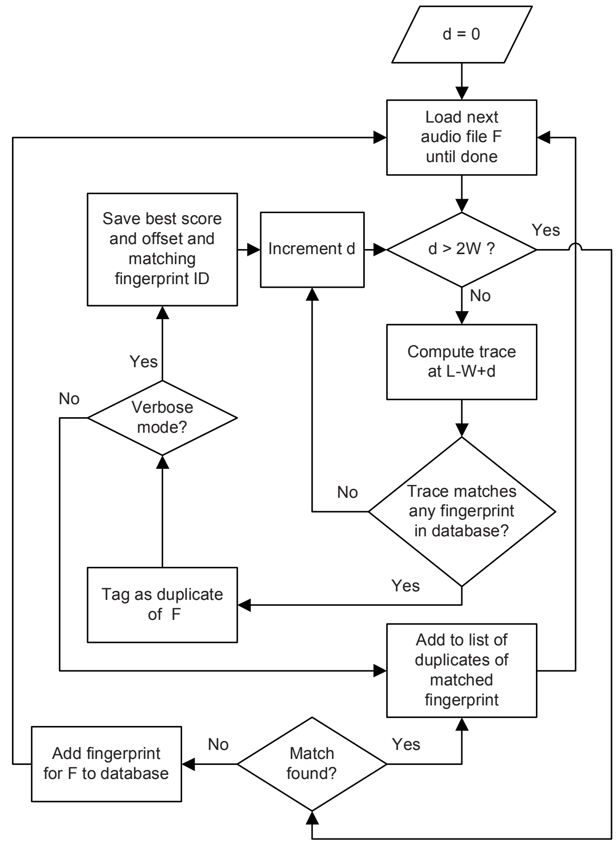

Fig. 1. Basic flowgraph of the duplicate detector.

We ran *DupDet* on 41,490 audio files. We selected fingerprints at 40 s into the music and a ± 5s search window. The threshold squared D_t for which two audio clips are identified as duplicates is 0.1 [2]. Of all files, 436 were unreadable, 63 loaded but were identified as noise thanks to the veto fingerprints, and the results on the remaining files are shown in Fig. 2. The top panel shows a histogram of the number of duplicates found. The log linear plot shows the Poisson nature of the distribution: the occurrence of duplicates is well-modeled as the limit of a binomial random process. The center panel is a histogram of optimal matching scores. We find that

95% of matches occur with score less than 0.026, and 99% with scores less than 0.067. (This is to be compared with 0.14, the threshold score the RARE engine currently uses to identify audio [2]). The highest score was 0.09948, which corresponded to the two copies being different mixes of a Beatles song. The bottom panel of Figure 1 shows a histogram of offsets, in seconds, where the center bar (of height 8,450) has been removed for clarity. Here, 95% of matches occur at absolute offsets less than 0.557s, and 99% at absolute offsets less than 2.04s.

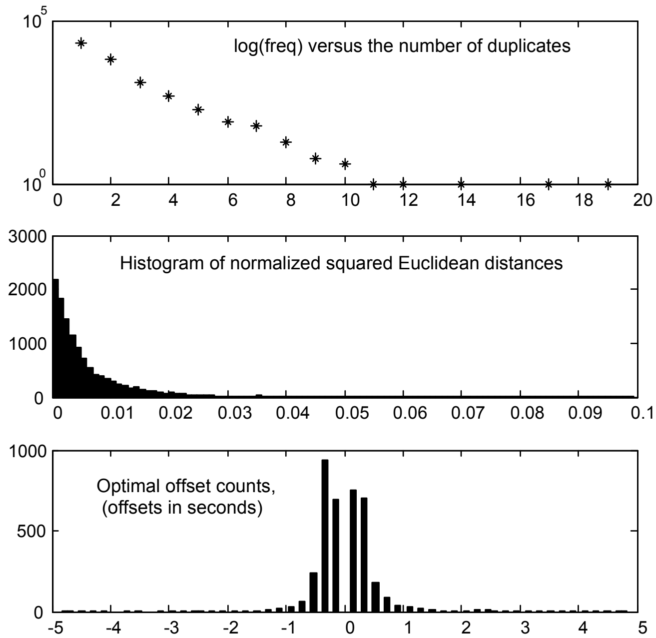

Fig. 2. Results of duplicate detection on 40,991 audio files.

3. AUDIO THUMBNAILS

The RARE audio thumbnail generator *GenThumb* works as in the basic diagram of Fig. 3, to generate a representative short clip for an audio file. The basic idea behind *GenThumb* is to find parts of the audio that repeat within the audio clip. That way, if a song has a chorus and all chorus instances are similar, the system will be able to identify the chorus, and use that to construct a good thumbnail. *GenThumb* also uses a measure of spectral flatness and a measure of spectral energy to decide between different pieces of the audio that repeat. These measures also allow *GenThumb* to generate a thumbnail even if the audio contains no repeats.

GenThumb uses audio fingerprinting to find repeating sections, since we expect similar sections of music to generate similar fingerprints. Using the fingerprints rather than attempting to match the original audio has two advantages: (1) due to the robustness of RARE to

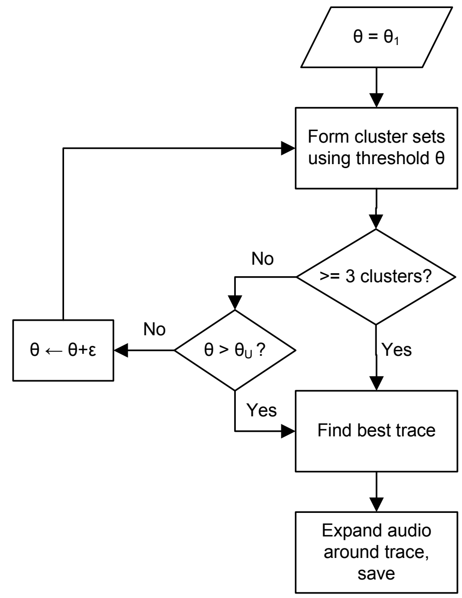

Fig. 3. Basic flowgraph of the thumbnail generator.

distortions, variations of the same segment within a song will often still give similar fingerprints, and (2) fingerprints are low-dimensional representations of the original music, so handling them instead of the audio is more efficient in terms of both memory and CPU usage. *GenThumb* computes three features from the audio to use for chorus detection: a 'cluster feature' F_C, which is a fingerprint and associated normalization, an 'energy feature' F_E, and a 'spectral flatness feature' F_F (with F_E and F_F computed from the same segment as F_C). The goal is to use these features to distinguish voiced choruses from purely instrumental repeated phrases, since the former are believe to be more mnemonic. Also, features F_E and F_F are used when the F_C features can't lead to a good chorus. *GenThumb* computes fingerprints that are approximately 3s long by concatenating 16 windows of 372 ms, each overlapping by 50% (the last layer of the DDA network was retrained for 3 second outputs [2]). All features F_C, F_E, and F_F are computed using these 372-ms frames. Three seconds was chosen as a good fit to the chorus detection task.

3.1. Feature Computation

Computation of the fingerprints follows [2]: in particular, for each frame, the spectral magnitudes are evaluated. The features F_E and F_F both use an average spectral magnitude as a normalization factor, so the features are independent of overall volume. To obtain F_E for each frame, a mean, normalized energy is computed by dividing the mean per-frequency-bin energy within the frame by the average of that quantity over all frames. This quantity is again averaged over the 16 frames that contribute to a given fingerprint. Thus, F_E measures

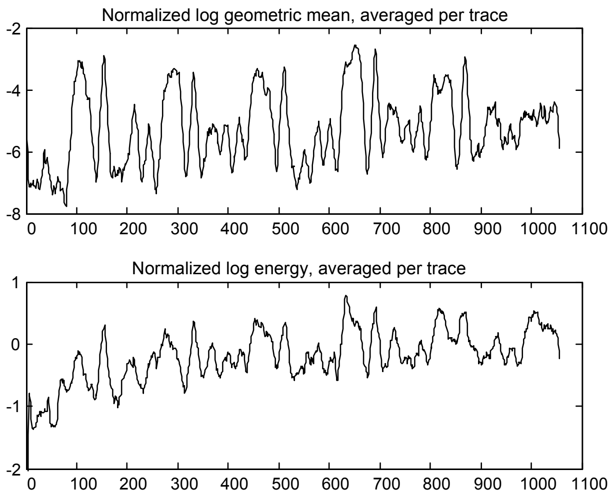

Fig. 4. Normalized log means for a song with 5 verses.

spectral energy per fingerprint.

For F_F, we compute the log normalized geometric mean of the magnitudes, where the normalization is performed by subtracting the per-frame log arithmetic mean of the magnitudes. The idea is that if the spectral energy is spread evenly throughout the frequency bins, then this quantity will be much larger than if it is concentrated across a few frequency bins. Finally, just as for the spectral energy F_E, this quantity is computed per fingerprint, by averaging over all frames that contribute to that fingerprint. Thus, F_F is a measure of spectral flatness per fingerprint. For some kinds of audio, we have found that high values of this quantity indicate a full sound (for example, when vocals dominate the sound this quantity tends to be high). Fig. 4 shows the per-trace quantities computed for the song "Buckets of Rain" by Bob Dylan; the top curve is F_F, the bottom is F_E. In this case F_F tracks the voice well: the song consists of 5 verses, and each verse is split temporally in two by a short instrumental. However, F_F is not always predictive of voiced music (for example, if the instruments dominate in a vocals/instrumental mix). For this reason *GenThumb* primarily uses the features F_C; features F_E and F_F are only used to distinguish cases for which F_C does not give a clear choice.

3.2. Cluster Computation

Traces for the whole song are computed, together with their normalization factors [2]. Traces are then added to 'cluster sets' $\mathcal{C}_i$. A trace T_1 is added to a cluster set $\mathcal{C}_i$ if there is a trace T_2 that is a member of $\mathcal{C}_i$ and that satisfies two conditions: (1) $D(T_1, T_2) < \theta$, where θ is a threshold, and (2) T_1 must be temporally separated from T_2 by a fixed minimum duration Y (we use 6s). Condition (2) is required to prevent adding traces that are similar just because they occur nearby. If a trace does not meet both conditions, for all cluster sets created so far, then it is added to a new cluster set. In this way, the number of cluster sets is grown until all traces are accounted for, and each cluster set contains one or more clusters of traces. A cluster is defined to be a collection of traces which is separated from all other such collections by at least Y. Once a cluster set has been created, it is added to recursively, until no more traces can be added. Once all traces have been processed, we determine the multiplicity m_i as the number of clusters in $\mathcal{C}_i$.

The steps above are performed for an initial value of the threshold $\theta = \theta_1$. If the maximum multiplicity of the resulting cluster sets is at least three, then that collection of cluster sets is used; otherwise, θ is incremented by a small amount, and the above computation is repeated, as shown in Fig. 3. This procedure also stops if $\theta \geq \theta_U$ for a fixed upper bound θ_U. In this way, the search criteria for forming clusters is incrementally loosened until either at least three clusters are found, or until further search is unlikely to find good clusters.

3.3. Choice of Cluster Set

If the clustering procedure finds no cluster sets with $m_i > 1$, then we resort to using the energy measures alone: we consider only fingerprints whose F_E is in the top third of the values of F_E for the whole song, to avoid quiet parts of the song. For the traces that survive this test, that trace whose surrounding 6s has the highest F_F is taken to be the optimal trace. If the clustering did result in at least one cluster set with $m_i \geq 2$, the remaining tasks are (1) to choose a good cluster set (which is likely to contain a fingerprint index corresponding to a chorus or repeat instrumental), and (2) to use that fingerprint to pick a suitable 15s thumbnail.

The quality of the clustering in a given cluster set $\mathcal{C}$ is measured using a scaled Renyi entropy R, in order to favor clusters that are evenly spread in time over clusters that are not. R is computed by normalizing the duration of the entire song to 1, and then scaling the center of each cluster to lie in the interval $[0, 1]$. Let the time position of the ith cluster be t_i, and let $\mathcal{C}$ contain N clusters. Setting $t_0 = 0$ and $t_{N+1} = 1$, then R is defined as

$$R = \frac{N+1}{N}\left(1 - \sum_{i=1}^{N+1}(t_i - t_{i-1})^2\right)$$

Since $\sum_{i=1}^{N+1}(t_i - t_{i-1}) = 1$, and since $t_i \geq t_{i-1}$, the differences $t_i - t_{i-1}$ can be interpreted as probabilities, so R is linearly related to the Renyi entropy for the corresponding distribution. The offset and scaling factor have been chosen so that R takes the maximum value of 1 and minimum value of 0, for any number of clusters N. This allows us to compare the quality of the spread of sets of clusters even when those sets contain different numbers of clusters.

Sometimes the spectral flatness feature F_F doesn't predict voice sections well. In those cases F_F tends to not vary much through the clip. Thus, we weight the F_F feature by its standard deviation: let $s_{\max}$ and $s_{\min}$ be the maximum and minimum standard deviations of a set of validation songs (only the central part of each song is used, to skip quiet introductions and fades). Define the linear mapping (a, b) by $as_{\min} + b = 0$ and $as_{\max} + b = 1$. Suppose a test clip has standard deviation s, and compute $y = as + b$. Replace y by $\bar{y} \equiv \min\{\max\{y, 0\}, 1\}$, and linearly map all values of F_F for the clip to the interval $[0, \bar{y}]$. Finally, each cluster set is ascribed a mean spectral flatness quality, which is just the mean of the scaled values F_F for the fingerprints in that set. Thus each set now has two numbers associated with it: one measures cluster spread quality, and varies from 0 to 1, and the other measures spectral spread quality, and varies from 0 to $\bar{y}$, where $\bar{y}$ is at most 1, and where it is large for those songs whose variance in their spectral spread is large. The best set is chosen to be that one for which the sum of the square of these two numbers is the highest. Once a set has been chosen, that trace with largest surrounding spectral energy in the set is chosen, and the thumbnail is taken as the 15s of audio surrounding that.

3.4. Results

To test *GenThumb*, we wrote a testing tool which presents two thumbnails to a user, who then rates them each on a scale of 0 to 5, corresponding to the thumbnail containing 'Voiced Title', 'Repeating Voiced Words', 'Any Other Vocals', 'Instrumental Only, Repeating', 'Instrumental Only, Not Repeating' and 'Other (e.g. Applause)'. The second method used to generate the thumbnail was to take the 15s starting 30s into the song, which was found to work well in many cases. For any given song the user is presented with the two thumbnails blindly, to prevent bias. *GenThumb* was tested on 68 songs, with lengths greater than 30s. *GenThumb* achieved a score of 68, whereas the default method scored 94 (lower score indicates higher quality). We also performed a Wilcoxon signed rank test on the score, to determine if the difference is statistically significant. This gave a z value of 2.219 in favor of *GenThumb*, meaning that *GenThumb* does better than the default at a confidence level of 99.9%, validating the significance of the results.

3.5. Conclusions

Audio fingerprinting has uses beyond the simple identification of music. We have shown that it can be used to detect duplicate audio files in large databases, even if the duplicates are compressed differently, or have different durations; in fact in the latter case, by aligning the matching fingerprints, the locations where the two files differ can be automatically detected (and if necessary checked with further fingerprint matching). We have also shown that by searching for repeated musical phrases within a single piece of music, representative sections of the music can be found automatically, which can then be used to create thumbnails that greatly facilitate browsing.

4. REFERENCES

[1] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of algorithms for audio fingerprinting," in *International Workshop on Multimedia Signal Processing*, December 2002.

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

[3] J. Goldstein, J.C. Platt, and C.J.C. Burges, "Indexing high dimensional rectangles for fast multimedia identification," Tech. Rep. MSR-TR-2003-38, Microsoft Research, 2003.