FAST HAMMING SPACE SEARCH FOR AUDIO FINGERPRINTING SYSTEMS

Qingmei Xiao **Motoyuki Suzuki** **Kenji Kita**
Faculty of Engineering, The University of Tokushima
Tokushima 770-8506, Japan
xiaoqingmei@iss.tokushima-u.ac.jp, moto@m.ieice.org,
kita@is.tokushima-u.ac.jp

ABSTRACT

In music information retrieval, a huge search space has to be explored because a query audio clip can start at any position of any music in the database, and also a query is often corrupted by significant noise and distortion. Audio fingerprints have recently attracted much attention in music information retrieval, for they provide a compact representation of the perceptually relevant parts of audio signals. In this paper, we propose an extremely fast method of exploring a huge Hamming space for audio fingerprinting systems. The effectiveness of the proposed method has been evaluated by experiments using a database of 8,740 songs.

1. INTRODUCTION

Just as fingerprints are used for identifying human beings, audio fingerprints can be used to identify music. Audio fingerprints, together with a music information database, can be used to derive information about an unknown audio clip automatically, such as the names of the song, artist and album. Gracenote [1] and Midomi [2] are two well-known commercial services. They retrieve a song by using a few seconds of music clip caught by such as a PC or mobile phone, display the title of the song and other information, and also enable the user to download the song from a web-based music store. In recent years, audio fingerprints have also attracted attention as a technique for copyright protection of music, such as detecting the distribution of copyright-infringing songs on the Internet.

In general, an audio clip is given as a search query, and it does not necessarily start at the beginning of the song. Therefore, a retrieval method should consider any time as a starting position, but this requires a long computation time. In order to solve this problem, fast and effective retrieval methods are necessary. Some efficient retrieval methods based on audio fingerprints have been proposed, including a method using a hash table [3][4] and a tree-structured representation of fingerprints [5].

A query is an excerpt of a song, but it may be “corrupted” by being mixed with environmental noise, or it may have been modified by a low-pass filter. As a result, retrieval methods should be able to handle queries which are similar to, but not exactly the same as a song in the database. Locality-Sensitive Hashing (LSH) is an emerging technique for solving large-scale similarity retrieval in high-dimensional spaces, and has been applied in extensive research fields [6-8].

In this paper, we propose a fast method for exploring a huge Hamming space which is suitable for audio fingerprinting systems building on the ideas of LSH. There have been several previous proposals on Hamming space retrieval methods based on LSH, however, our method uses less memory. The effectiveness of the proposed method is demonstrated by evaluation experiments using a database of 8,740 songs. The paper is organized as follows: Section 2 outlines music retrieval based on audio fingerprints. We propose a fast music retrieval method particularly suitable for audio fingerprinting systems in Section 3, and evaluate the method in Section 4. Finally, we conclude the paper in Section 5.

2. OVERVIEW OF MUSIC RETRIEVAL BASED ON AUDIO FINGERPRINTS

Audio fingerprinting is a kind of message digest (one-way hash function), and it converts an audio signal into a relatively compact representation by using acoustical and perceptional characteristics of the audio signals. For message digesting methods used for authentication and digital signatures (e.g. MD5), slight difference in the original objects results in totally different hash values. This means that two hash values mapped from an original audio signal and a corrupted one are completely different, which drastically decreases the retrieval performance for “corrupted” queries. However, in audio fingerprinting, similar inputs are hashed to similar hash values.

Music retrieval based on audio fingerprinting involves some key problems: (1) which type of audio fingerprints to use, (2) how to define the distance between two fingerprints, and (3) how to retrieve from a huge database. We review these problems next.

2.1 Audio Fingerprint Extraction

A variety of audio fingerprint extraction algorithms have been proposed based on different acoustic features, such as Fourier coefficients [9], Mel frequency cepstral coefficients [10], spectral flatness [11] and so on. In particular, the fingerprint extraction algorithm by Haitsma and Kalker [3] uses a feature of the energy difference between frequency bands as follows.

First, an input audio signal is segmented into frames, and then 32-bit sub-fingerprints are extracted from each overlapping frame. Sub-fingerprints are actually calculated in the frequency domain. Each frame is first converted into a frequency domain by using FFT, and then segmented into 33 non-overlapping frequency bands. Next, a sub-fingerprint is calculated by checking the sign (plus or minus) of the energy difference between two successive frequency bands. Haitsma and Kalker [3] used a frame length of 0.37 second with an overlap factor of 31/32, so a sub-fingerprint was extracted for every 11.6 milliseconds.

The sub-fingerprints are calculated as follows: let $E(n, m)$ be the power of frequency band m of frame n, then the m-th bit of frame n, $F(n, m)$, is determined as:

$$F(n,m) = \begin{cases} 1 & \text{if } ED(n,m) > 0 \\ 0 & \text{if } ED(n,m) \leq 0 \end{cases}, \quad (1)$$

where

$$ED(n,m) = E(n,m) - E(n,m+1) - (E(n-1,m) - E(n-1,m+1)) \quad (2)$$

Haitsma and Kalker [3] demonstrated that the sign of power differences between successive frequency bands was effective for identifying music, and was also robust against various “corrupted” inputs such as compressed or delayed music. The Haitsma and Kalker algorithm can be implemented by simple arithmetic, while maintaining compact representation for generated audio fingerprints.

2.2 Distance between Audio Fingerprints

The sub-fingerprint is a 32-bit feature extracted from a frame in an input audio, and one sub-fingerprint does not have enough information to identify the audio. To obtain sufficient information, a fingerprint block, which is a sequence of sub-fingerprints, is used for matching audio sub-fingerprints. A fingerprint block consisting of 256 sub-fingerprints was used in the experiments in [3].

Bit error rate is used as the distance between two fingerprint blocks. Let $F_A(n, m)$, $F_B(n, m)$ be the sub-fingerprints extracted from audio clips A and B respectively. The bit error rate of fingerprint block $BER(A, B)$ of length N is formally defined as:

$$BER(A,B) = \frac{\sum_{n=1}^{N}\sum_{m=1}^{32}[\, F_A(n,m)^\wedge F_B(n,m) \,]}{32N} \quad (3)$$

The operator ^ denotes bitwise operation XOR (exclusive or). The numerator of Equation (3) calculates the Hamming distance between two fingerprint blocks, which is divided by the bit length of fingerprint blocks ($32N$). $BER(A,B)$ is the error rate per bit.

2.3 Audio Fingerprint Search

Most music retrieval methods based on audio fingerprinting have the following stages. First, fingerprint blocks are extracted from each song in the database. Because of the unknown position of the query, all variations of starting point should be considered. Therefore, each song allows extracting quite a number of fingerprint blocks by shifting all the frames to fingerprint blocks one by one. When a query is given, many fingerprint blocks are also extracted from the query. Thus, music retrieval involves finding the fingerprint block in the database that is most similar to the fingerprint block derived from the query.

The search space of audio fingerprinting is huge. For example, a fingerprint database containing 10,000 songs each with an average length of 5 minutes would result in approximately 250 million fingerprint blocks in total using the algorithm in [3]. The number of distance calculations would be several to several dozen times as large as 250 million by brute-force search taking account of matching the fingerprint blocks. Many ways of reducing the number of calculations have been proposed, such as using a hash table (lookup table) for sub-fingerprints [3], a tree-structured representation of sub-fingerprints [5], and a hash table consisting of peak values in the frequency domain and duration between the two peaks [4]. However, with these methods the size of the hash table grows rapidly with the bit error rates between the query and songs in the database increasing.

3. FAST HAMMING SPACE SEARCH FOR AUDIO FINGERPRINTS

In this section, we propose a fast retrieval method for audio fingerprinting systems. Suppose that audio fingerprints are represented by binary bit vectors, and the Hamming distance is used for the distance between two audio fingerprints. We first outline the search methods for Hamming space based on LSH in Section 3.1, and then propose a new retrieval method in Section 3.2.

3.1 Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a hashing scheme for probabilistic searches of large-scale high-dimensional data, rather than a specific algorithm. It includes the hashing method for Hamming distance using bit sampling [6], the method for Jaccard distance using min-wise independent permutation [12], the method based on random projection for cosine distance [13], and the method using p-stable distribution for *Lp* distance [14]. The concept of LSH is to map the high-dimensional vector data into hash values so that similar data are mapped to the same hash values with high probability. Generally, we cannot find a hash function which gives the same hash values for similar high-dimensional data. LSH can maintain certain retrieval accuracy by using multiple hash functions.

There are a few Locality-Sensitive Hashing schemes proposed to reduce the problem in the Hamming space. Indyk and Motwani proposed an LSH algorithm for Hamming space based on the Point Location in Equal Balls (PLEB) problem [15], and Charikar [13] and Ravichandran [16] improved the algorithm by using random permutations of binary vectors.

The concept of random permutations is as follows: given a set of n vectors $D = \{d_1, d_2, \ldots, d_n\}$, where each vector consists of k binary bits, permutation σ is defined as a bijection on $\{1, 2, \ldots, k\}$, and then we can define that the bit vector $b_{\sigma(1),} b_{\sigma(2)}, \cdots, b_{\sigma(k)}$ is a permutation of $b_1, b_2, \cdots, b_k$. The number of permutations for k bits is $k!$, hence a random permutation is a random selection from these $k!$ permutations.

We can now create the data set D_σ by permuting all bits by using σ for all elements in the data set D, and also calculate the new query vector q_σ from the query vector q in the same way. The most similar vector to q_σ can be found in the data set D_σ by doing the following steps: Sort D_σ in lexicographic order, and then perform the binary search. The binary search is carried out from the first bit to the last bit, so if a different bit is located in the upper side (near the first bit), then the search makes a mistake. On the other hand, if a different bit is located in the lower side, the search can find the nearest vector. We expect to find the most similar vector by making a number of random permutations σ and corresponding data set D_σ, and searching for all data sets. This is an overview of the LSH for Hamming space proposed by Charikar [13] and Ravichandran [16]; the details of the theories and experimental analysis of this method are discussed in [13] and [17].

3.2 Fast Hamming Space Search Method for Audio Fingerprints

The principle of the Hamming space search based on random permutations is simple. The binary search can certainly find the exact vector if there exists one vector the same as the query. A similar vector which has a few different bits in the lower side can be found, too, but the problem is that sometimes it cannot find a similar vector which has a few different bits in the upper side. To address this problem, random permutations are used. In general, LSH-based methods use multiple hash functions. In the Hamming space search based on the random permutation method, multiple random permutations can be regarded as multiple hash functions.

The greatest disadvantage of the retrieval method based on random permutations is the requirement for a huge amount of memory in order to perform many random permutations on the original database in advance. This increases the size of database required to at least several to several dozen times larger than the size of the original database.

If we could only multiplex the query vectors without multiplexing the database vectors, then Hamming space searching would require little memory. Based on this assumption, we propose a new search method by modifying the query vector into many similar vectors.

The scheme of the search method based on random permutations is shown in Figure 1, and that of the proposed method is shown in Figure 2. In the random permutation method, multiple random permutations (σ_1, σ_2 and σ_3 in Figure 1) are applied to both the original database and query vector in order to solve the problem of search omissions. On the other hand, in the proposed method, only the query is multiplexed through the functions (φ_1, φ_2 and φ_3 in Figure 2). The definition of functions φ_i is necessarily application-dependent.

The proposed method is based on the sub-fingerprint matching scheme, and functions φ_i create the multiplexed search queries of sub-fingerprint sequences from the query audio clip. Many sub-fingerprints are extracted by shifting the query into frames. Moreover, there exists a great similarity between the overlapping sub-fingerprints in the sequence of sub-fingerprint, so that multiplexed sub-fingerprints with slight differences can be obtained as starting time of frame moving down. These sub-fingerprints are used for queries multiplexing, which make it possible to search for a song without modifying the original database by using random permutations.

The flow of the proposed method is as follows: first, estimate several candidates of starting position in the database those using sub-fingerprints obtained from the query. Then, calculate the Hamming distance (bit error rate) for the fingerprint blocks of query music data and estimated candidates. Usually one sub-fingerprint does not contain sufficient information for music identification, so a sequence of

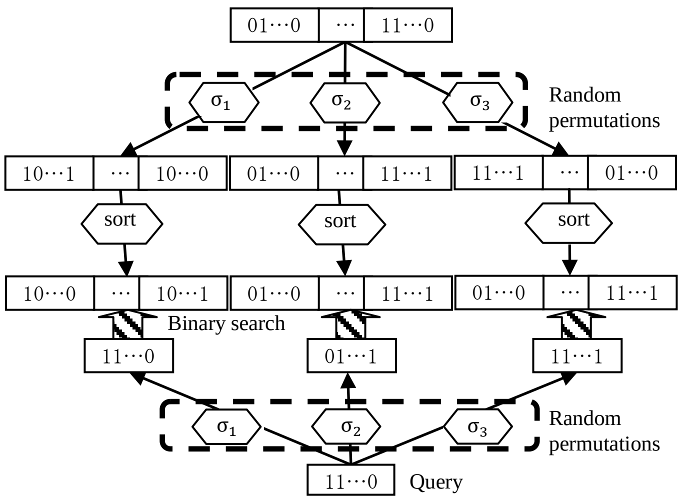

Figure 1. Schematic diagram of search based on random permutations

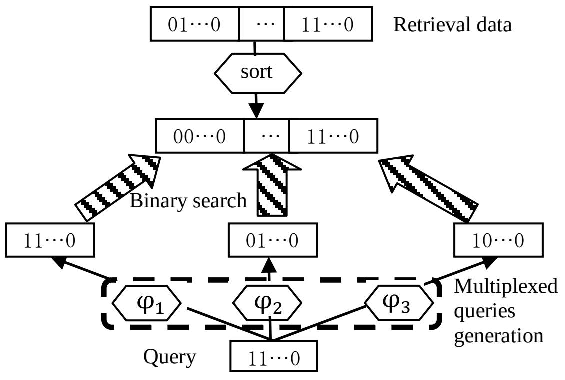

Figure 2. Schematic diagram of search based on query multiplexing

sub-fingerprints (SSF) is used. In this paper, the length of the SSF is set to 3 (3 sub-fingerprints, containing 96 bits in total).

A schematic diagram of the SSF search is shown in Figure 3. The sub-fingerprints obtained from all the songs of the database are denoted by $FP = (FP_1, FP_2, \dots, FP_n)$. As stated above, we can get many SSFs in certain length by changing the starting position of fingerprint. Let m be the length of each SSF, and the SSFs are constructed from FP in such a way that $SSF_1 = (FP_1, FP_2, \dots, FP_m)$, $SSF_2 = (FP_2, FP_3, \dots, FP_{m+1})$ and the i-th sub-fingerprint sequence $SSF_i = (FP_i, FP_{i+1}, \dots, FP_{i+m-1})$. All the SSFs are sorted by value and the sorted positions of SSFs are stored in a one-dimensional array $S = S_1, S_2, \dots, S_{n-m+1}$. Array S, similar to the suffix array [18], contains the indexes to FP and satis-

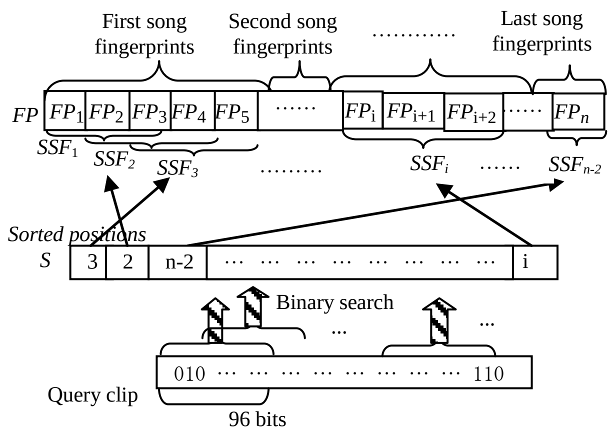

Figure 3. Schematic diagram of SSF search

fies the following:

$$S_j = i \quad \text{iff} \quad SSF_i = (FP_i, FP_{i+1}, \dots, FP_{i+m-1}) \text{ is the } j\text{-th SSF in sorted order.} \qquad (4)$$

In the search step, a binary search is performed on array S for all the SSFs extracted from the query audio clip. Most similar SSF can be found by checking the neighborhood positions of the searched block in array S.

Array S is used as an index for music retrieval. The size of the index is proportional to the length of the sub-fingerprint sequence in the database, so it requires much less memory/storage compared with the method based on random permutations.

The proposed method can be summarized as follows:

(1) Extract the sub-fingerprint sequence *FP* from query music.
(2) For all SSFs, find candidate positions by performing a binary search on array *S*.
(3) Set the start position of the *FP* to the candidate position, and calculate the Hamming distance (bit error rate) between *FP* and the fingerprint block corresponding to the SSF.
(4) Output the top n songs as the final results.

4. EVALUATION EXPERIMENTS

To evaluate the effectiveness of the proposed method, real music data were used for evaluation experiments. In these experiments, the algorithm proposed by Haitsma and Kalker [3] was used for extracting audio fingerprints in different acoustical analysis settings.

4.1 Music Data

The database had 8,740 songs in mp3 format from CDs or the Internet. The compression ratio was different for each song. There were many genres in the database such as pop, classical, and folk music.

Category		Notes	Audio	Accuracy
Original music	Non-noise	PV music faithful to the original music with little noise if any.	104	96.2%
	With noise	Declared to be original but with obvious noise.	22	100%
Live data		Live audio, most of which contain voices, cheering and applause, and other noise.	142	83.1%

Table 1. Results on evaluation data

Music clips uploaded to YouTube were used for the queries. Audio data were extracted from various types of videos, such as promotional video and live video. Many of the music data were of poor quality, including music following and followed by long silences, and music with various types of noise such as hand-clapping, cries of excitement, and other environmental noise. 268 songs were used for evaluation data, which are roughly classified by hand. Details of the evaluation data are shown in Table 1.

4.2 Acoustical Analysis Settings

After down-sampling to 4,000 Hz, the music data were segmented into frames by using a Hamming window. The frame length was set to 1.024 seconds and the frame shift to 32 milliseconds. All frames were converted into the frequency domain by FFT. The frequency domain was divided into 33 frequency bands, and 32-bit sub-fingerprint features were extracted. The length of the fingerprint block was set to 128.

Although these settings seem rough compared with those given by Haitsma and Kalker [3], these parameters were determined by many preliminary experiments and the resulting proposed algorithm gave a high accuracy. The total number of sub-fingerprints was about 70 million.

4.3 Experimental Results

Experiments were carried out on a PC (DELL Precision M6500) with an Intel Core i7 CPU (1.73 GHz) (8 cores) and 4 GB of memory. Retrieval times varied as the query music, and each song was retrieved in approximately 0.4 to 0.6 seconds.

The length of the sub-fingerprint sequence for one query music was approximately from 6,000 to 8,000 fingerprints. The proposed algorithm searches candidate positions by a binary search for all SSFs (length was 3) extracted from the query music before calculating the bit error rate of fingerprint blocks. Therefore, we believe that the algorithm is competent and fast since the retrieval time per SSF did not exceed 0.1 milliseconds.

The top-1 retrieval accuracy is shown in the right column of Table 1. The retrieval rate for "original music" was 98.6%, and accuracy for "live music" was 83.1%. The difference was due to the different melody of the live clip from that of the original music. The evaluation data of "original music" can be divided into two classes with regard to noise, but the results did not show any influence of noise.

5. CONCLUTIONS

In this paper, we have proposed a fast Hamming space search method for audio fingerprinting systems. Our method is inspired by Locality-Sensitive Hashing (LSH), a probabilistic algorithm for solving the nearest neighbor search in high-dimensional spaces. LSH uses multiple hash functions to maintain high retrieval accuracy and therefore requires a large amount of memory/storage for saving hash tables. For the Hamming space search, LSH must maintain multiple database sets created by random permutations. On the other hand, the proposed method created multiplexed search queries of sub-fingerprint sequence with different starting time, and does not require expansion of the database. As a result, a large amount of memory/storage is not needed. Experimental results showed that the proposed method delivers accurate, fast retrieval.

6. REFERENCES

[1] Gracenote: available from http://www.gracenote.com/.

[2] Midomi: available from http://www.midomi.com/.

[3] Jaap Haitsma and Ton Kalker: "Highly Robust Audio Fingerprinting System," *Proceedings of the 3rd International Conference on Music Information Retrieval* (*ISMIR 2002*), pp.107–115, 2002.

[4] Avery Li-Chun Wang: "An Industrial-Strength Audio Search Algorithm," *Proceedings of the 4th International Conference on Music Information Retrieval* (*ISMIR 2003*), pp.7-13, 2003.

[5] Matthew L. Miller, Manuel Acevedo Rodriguez, and Ingemar J. Cox: "Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces," *Journal of VLSI Signal Processing*, Vol. 41, No. 3, pp.285-291, 2005.

[6] Aristides Gionis, Piotr Indyk, and Rajeev Motwani: "Similarity Search in High Dimensions via Hashing," *25th International Conference on Very Large Data Bases* (*VLDB 1999*), pp.518–529, 1999.

[7] Brian Kulis and Trevor Darrell: “Learning to Hash with Binary Reconstructive Embeddings,” *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems* (*NIPS 2009*), pp.1042–1050, 2009.

[8] Brian Kulis and Kristen Grauman: “Kernelized LSH for Scalable Image Search,” *Proceedings of the 12th IEEE International Conference on Computer Vision* (*ICCV 2009*), pp.2130–2137, 2009.

[9] Dimitrios Fragoulis, George Rousopoulos, Thanasis Panagopoulos, Constantin Alexiou, and Constantin Papaodysseus: “On the Automated Recognition of Seriously Distorted Musical Recordings,” *IEEE Transactions on Signal Processing*, Vol. 49, No. 4, pp.898–908, 2001.

[10] Beth Logan: “Mel Frequency Cepstral Coefficients for Music Modeling,” *Proceedings of the International Symposium on Music Information Retrieval* (*ISMIR 2000*), pp.11–23, 2000.

[11] Eric Allamanche et al.: “AudioID: Towards Content-based Identification of Audio Material,” *Proceedings of the 110th AES Convention*, 2001.

[12] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher: “Min-wise Independent Permutations,” *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp.327–336, 1998.

[13] Moses S. Charikar: “Similarity Estimation Techniques from Rounding Algorithms,” *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp.380–388, 2002.

[14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni: “Locality-Sensitive Hashing Scheme Based on p-Stable Distributions,” *Proceedings of the 20th Annual Symposium on Computational Geometry*, pp.253–262, 2004.

[15] Piotr Indyk and Rajeev Motwani: “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality,” *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp.604–613, 1998.

[16] Deepak Ravichandran, Patrick Pantel, and Eduard Hovy: “Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering,” *Proceedings of ACL*, pp.622–629, 2005.

[17] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma: “Detecting Near-Duplicates for Web Crawling,” *Proceedings of the 16th international conference on World Wide Web*, pp.141–149, 2007.

[18] Udi Manber and Gene Myers: “Suffix Arrays: A New Method for On-line String Searches,” *SIAM Journal on Computing*, Vol. 22, No. 5, pp.935–948, 1993.