A Study on Music Retrieval based on Audio Fingerprinting

Qingmei Xiao

A Thesis submitted to the University of Tokushima in partial fulfillment of the requirements for the degree of Doctor of Philosophy

September, 2013

Department of Information Science and Intelligent Systems
Graduate School of Advanced Technology and Science
The University of Tokushima

Contents

List of Figures

List of Tables

Abstract

For an efficient and accurate music retrieval system, 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 fingerprinting has recently attracted much attention in music information retrieval, which provides a compact representation of the perceptually relevant parts of the audio signals. For the audio fingerprinting-based music retrieval methods, a large database is required in order to compare the fingerprints extracted from the query. To retrieve music information from such database, an efficient search method has to be developed.

In this thesis, we propose a fast Hamming space search method for audio fingerprinting based on query multiplexing, by using a suffix array. Our method is inspired by the Locality-Sensitive Hashing (LSH) algorithm, which is a probabilistic algorithm for solving the nearest neighbor search in high-dimensional spaces. The LSH algorithm employs multiple hash functions to maintain a high retrieval accuracy and therefore requires a large amount of memory/storage for saving the hash tables. For the Hamming space search, the LSH algorithm must maintain multiple database sets at the same time which have been created by random permutations. To solve such problem, the proposed method creates multiple multiplexed search queries of sub-fingerprint sequence with different starting time, and therefore does not require the expansion of the database. As a result, a large amount of memory/storage is saved. The effectiveness of our method has been evaluated by experiments using a database which contains 8,740 original songs, 800 artificially corrupted songs and 268 original queries.

We evaluate the performance of the audio fingerprinting system in real-noise conditions by adding noise data (ten types of noise data) to the music data in

accordance with signalto-noise ratios (SNR). For the reason that the query music could be distorted by external noise, such as the noise on a smart phone which can be used anywhere, including some fairly noisy places such as shopping malls or playgrounds (i.e., varieties of external noise are unavoidable and should be taken into account), it is necessary to examine the robustness of the audio fingerprinting system against the real noise in practical applications.

We also present a method for index compression using a compressed suffix array. This method first sorts the audio fingerprints, and then compresses the index by encoding the 8-bit data sequences with the Run Length Encoding. Vertical Code, which represents a smaller value in a smaller size, is used to compress the array, in which the positions of the sorted data are stored. Compared with the conventional method, our method only needs an audio fingerprints database taking 30% the original space as the music database consisting of 8,000 songs, and an index space of around 80% as the database of 1,000 songs. Moreover, the entire space cost is reduced to around 60%.

Chapter 1

Introduction

1.1 Backgrounds

With the development of the Internet and music compression technology, large amounts of music is available for us. A music retrieval system would be necessary and helpful to retrieve a song as we need from a large-scale music database. As one of music retrieval systems, text retrieval can retrieve music by using a song name, artist and so on. Music can be also retrieved by using lyrics. However, the text retrieval does not work if there is no information represented by the text. The retrival method using lyrics would also fail to work if there is no lyrics in music, such as an intrumental song.

A retrieval method based on audio signals can avoid failure in these cases. For example, a song suddenly catches your attention while you are watching TV. You like the song so much that you still would like to know more about this song. However it is the first time you heard the song and thus you do not know the song name or artist. In this case, the recorded music would help.

A music retrieval system enables users to easily obtain information about an unknown song such as song name, artist and album. Most music retrieval systems adopt audio fingerprinting algorithm proposed by Haitsma and Kalker [1]. An audio fingerprint is a condensed feature that can be used to identify a song or quickly locate similar songs in an audio database. Information for an unknown music clip can be derived by using audio fingerprints generated from the music clip, together with an audio database. Specifically, extract the audio fingerprints of the song you wish to know, and then compare the fingerprints with that of the music in the database; Among the songs in the database, select the music whose fingerprints are most similar as the song you are looking for.

Content-based music information retrieval has become one of the most attractive application services pursued by many companies in recent years. For instance, Shazama [2] and Gracenote [3] can return the artist, track, lyrics and more information from a few seconds of music clip captured by a smart phone. Even more, the song which has been identified can be also perchased directly.

1.2 Purpose and Significance of the Study

Audio fingerprints are not only used to retrieve music, but also for copyright protection, such as detecting illegal copying of digital contents and the distribution of copyright-infringing songs on the Internet. Illegal copying can be detected by comparing the similarity of audio fingerprints extracted from the original music and the music on the Internet respectively.

On the other hand, a large database to be searched is required for the identification of the music by the audio fingerprinting. However, it takes a long time to search in a brute-force method since the database to be explored is quite large. That is, there is a need for efficient database searching method.

Some efficient retrieval methods based on audio fingerprinting have been proposed, including a method using a hash table [4] and a tree-structured representation of fingerprints [5]. However, the space cost increases in proportion to the growing music database.

In this study, we propose a fast space-saving method for exploring a huge Hamming space which is suitable for audio fingerprinting systems building on the ideas of Locality-Sensitive Hashing. We also compress the index of the audio fingerprints by using the compressed suffix array to reduce the space cost by compressing the index of the database.

1.3 Thesis Organization

The thesis is organized as follows:

Chapter 2 outlines the music retrieval method based on audio fingerprinting. Beginning with introducing the audio fingerprinting, we illustrate the features and advantages of audio fingerprinting, including a relatively compact representation and that similar inputs are hashed to similar hash values. Then to review the most popular audio fingerprint extraction algorithm which is proposed by Haitsma and Kalker. The sign of power differences between successive frequecy bands is used in this algorithm. For the distance between audio fingerprints, we use the bit error tate to calculate. Finally, we give the main idea and processing of the audio fingerprint search method.

Chapter 3 proposes a fast Hamming space search method based on query multiplexing. For the proposed method is inspired by the Locality-Sensitive Hashing (LSH), the LSH is introduced first. As a typical LSH method, the retrieval method based on random permutations requires for a huge amount of memory in order to perform many random permutations on the original database in advance. We give our mehod based on query multiplexing which costs much less space.

Chapter 4 gives a performance evaluation of audio fingerprinting systems in real-noise conditions. We first review the main idea of the Hamming space search based on query multiplexing and the related work on the audio fingerprinting algorithm under additive noise. Then we evaluate the performance of audio fingerprinting systems in real-noise conditions by adding noise data (ten types of noise data) to music data in accordance with signalto-noise ratios (SNR).

Chapter 5 presents an index compression method based on a compressed suffix array in order to reduce the space. We first give the reason why a compressed suffix array for index compression is used. Subsequently, the method how to compress the index by using a compressed suffix array is elaborated. Run Length Encoding and Vertical Code used for index compression are stated in this chapter. Finally, the music retrieval method based on index compression using a compressed suffix array is evaluated through experiments.

In Chapter 6, we conclude this thesis and disscuss the future work.

Chapter 2

Music Retrieval based on Audio Fingerprinting

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 developed based on different acoustic features, such as Fourier coefficients [6], Mel frequency cepstral coefficients [7], spectral flatness [8] and so on. In particular, the fingerprint extraction algorithm by Haitsma and Kalker [1] uses a feature of the energy difference between frequency bands as follows: (1) frame segmentation, (2) frequency-domain representation, (3) frequency band division, and (4) sub-fingerprint extraction.

(1) Frame segmentation

An input audio signal is segmented into frames, and then 32-bit sub-fingerprints are extracted from each overlapping frame. Haitsma and Kalker 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 process of frame segmentation is shown in Figure 2.1.

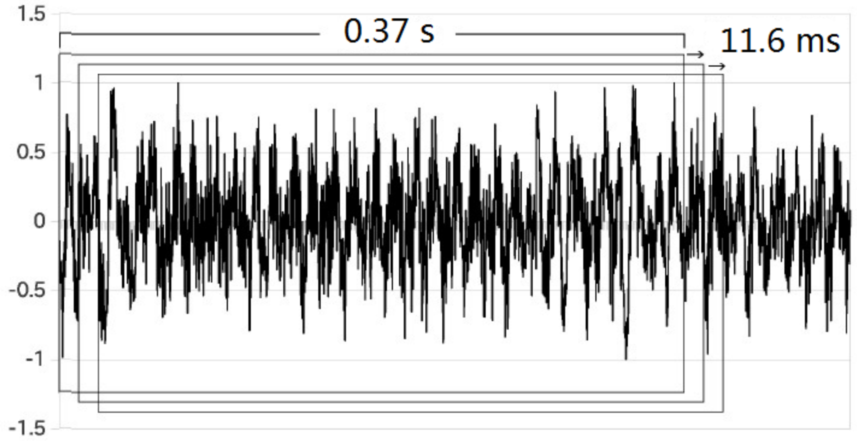

Figure 2.1: Frame segmentation

(2) Frequency-domain representation

Sub-fingerprints are actually calculated in the frequency domain. Each frame is first converted into a frequency domain by using FFT. The frames are weighed by the Hanning window in order to reduce the influence of the boundary of the frames. Window functions such as Hanning window and Fourier transform are to be explained in the next section.

(3) Frequency band division

For each frame, segment all the frequency bands lying in the range from 300Hz to 2000Hz into 33 non-overlapping frequency bands which have a logarithmical spacing. These bands are set by the Bark scale based on the hunan auditory characteristics.

(4) Sub-fingerprint extraction

Next, a sub-fingerprint is calculated by checking the sign (plus or minus) of the energy difference between two successive frequency bands. 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 by Equation (2.1):

$$F(n,m) = \begin{cases} 1 & \text{if } ED(n,m) > 0 \\ 0 & \text{if } ED(n,m) \leq 0, \end{cases} \tag{2.1}$$

where

$$ED(n,m) = E(n,m) - E(n,m+1) - \big(E(n-1,m) - E(n-1,m+1)\big). \tag{2.2}$$

By the above method, a 32-bit sub-fingerprint can be derived from one frame. One sub-fingerprint is extracted per 11.6 milliseconds. That is, about 25,000 sub-fingerprints can be generated from a 5-minute song.

Haitsma and Kalker [1] 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.

The overview of Haitsma-Kalker algorithm is shown in 2.2.

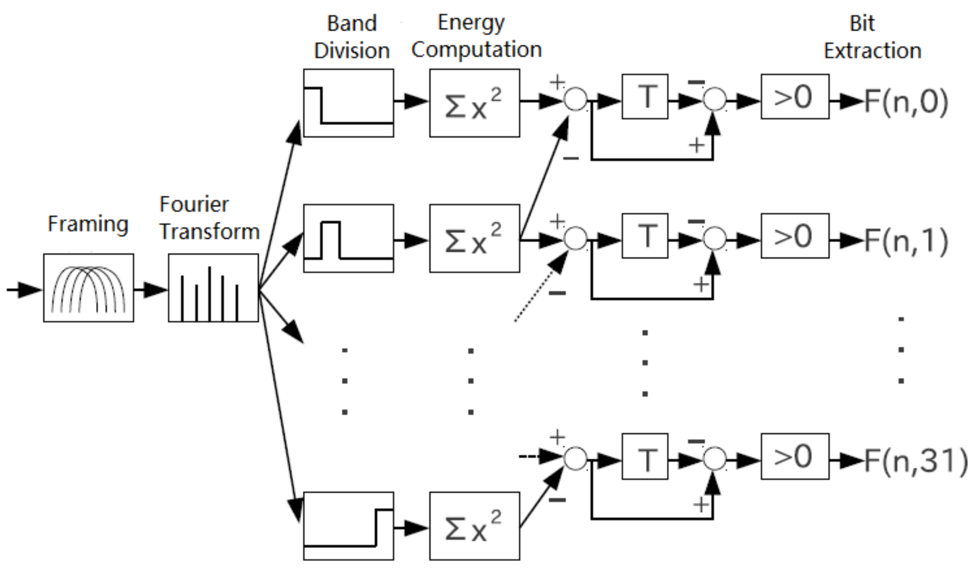

Figure 2.2: The overview of Haitsma-Kalker fingerprint extraction scheme [1]

2.1.1 Fourier Transform

In this subsection, we describe the Fourier transform algorithm [9]. Fourier transform(FT) can transform a mathematical function of time (or space) into a frequency function. Fourier transform is defined by

$$H(f) = \int_{-\infty}^{\infty} h(t)\exp(-i2\pi f t)dt. \tag{2.3}$$

Here, t is the time, f represents the frequency and $\exp(x) = e^x$. If Equation (2.3) is expressed as $\omega = 2\pi f$, it becomes

$$H(\omega) = \int_{-\infty}^{\infty} h(t)\exp(-i\omega t)dt. \tag{2.4}$$

The digital signal used here is discrete Fourier transform (DFT), which is defined by

$$X_k = \sum_{n=0}^{N-1} x_n \exp(-i2\pi kn/N). \tag{2.5}$$

In (2.5), the sequence of N complex numbers $x_0,\ x_1,\ \cdots,\ x_{N-1}$ is transformed into an N-periodic sequence of complex numbers $X_0,\ X_1,\ \cdots,\ X_{N-1}$.

2.1.2 Fast Fourier Transform

Fourier transform is implemented by fast Fourier transform (FFT) on a computer. According to the definition, the computation of Fourier transform is proportional to N^2, given N samples. FFT is an algorithm that reduces the computational complexity. The principles of Fourier transform are to be explained in the following.

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) and its inverse. A Fourier transform converts time (or space) to frequency and vice versa. An FFT rapidly computes such transformations. As a result, fast Fourier transforms are widely used for many applications in engineering, science, and mathematics.

For the discrete Fourier transform formula (2.5), suppose

$$W_N = \exp(-i2\pi/N), \tag{2.6}$$

Equation (2.5) can be expressed as

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn}. \tag{2.7}$$

Given $N = 2^Q$, the calculation can be expressed by a matrix of a product of two matrices, that is $2^Q \times 2^Q$.

If $Q = 2$, that is, $N = 2^2 = 4$, Equation (2.8) becomes

$$X_k = \sum_{n=0}^{3} x_n W_N^{kn}, \tag{2.8}$$

where $k = 0,\ 1,\ 2,\ 3$. It also can be expressed by a matrix as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{2.9}$$

where

$$W_4 = W_4^9 = \exp(-i2\pi/4) = -i \tag{2.10}$$

$$W_4^2 = W_4^6 = \exp(-i2\pi 2/4) = -1 \tag{2.11}$$

$$W_4^3 = \exp(-i2\pi 3/4) = i \tag{2.12}$$

$$W_4^4 = \exp(-i2\pi) = 1. \tag{2.13}$$

Interchanging the second and third rows, we can get

$$\begin{bmatrix} X_0 \\ X_2 \\ X_1 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \tag{2.14}$$

which can be replaced by

$$\begin{bmatrix} X_0 \\ X_2 \\ X_1 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -i \\ 0 & 0 & 1 & i \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{2.15}$$

Substitute the following

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_{0,0} \\ x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix} \tag{2.16}$$

and change the second part of Equation (2.15) to

$$\begin{bmatrix} x_{0,1} \\ x_{1,1} \\ x_{2,1} \\ x_{3,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_{0,0} \\ x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix} = \begin{bmatrix} x_{0,0} + x_{2,0} \\ x_{1,0} + x_{3,0} \\ x_{0,0} - x_{2,0} \\ x_{1,0} - x_{3,0}, \end{bmatrix} \tag{2.17}$$

in the first stage of processing. Replace the first part of Equation (2.15) with

$$\begin{bmatrix} x_{0,2} \\ x_{1,2} \\ x_{2,2} \\ x_{3,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -i \\ 0 & 0 & 1 & i \end{bmatrix} \begin{bmatrix} x_{0,1} \\ x_{1,1} \\ x_{2,1} \\ x_{3,1} \end{bmatrix} = \begin{bmatrix} x_{0,1} + x_{1,1} \\ x_{0,1} - x_{1,1} \\ x_{2,1} - ix_{3,1} \\ x_{2,1} + ix_{3,1} \end{bmatrix} \tag{2.18}$$

for the second stage of processing. Equation (2.14) can be expressed as

$$\begin{bmatrix} X_0 \\ X_2 \\ X_1 \\ X_3 \end{bmatrix} = \begin{bmatrix} x_{0,2} \\ x_{1,2} \\ x_{2,2} \\ x_{3,2} \end{bmatrix} = \begin{bmatrix} x_{0,1} + x_{1,1} \\ x_{0,1} - x_{1,1} \\ x_{2,1} - ix_{3,1} \\ x_{2,1} + ix_{3,1}. \end{bmatrix} \tag{2.19}$$

Similarly, the matrix $2^Q \times 2^Q$ can be decomposed into Q matrixes of $2^Q \times 2^Q$ which contains many elements of 0. The matrix operation, 2^Q times in each stage, is so-called butterfly operation. That is, such operation as

$$\begin{cases} x_{k,l} & = x_{k,l-1} + W^p x_{k+N/2^l,l-1} \\ x_{k+N/2^l,l} & = x_{k,l-1} - W^p x_{k+N/2^l,l-1} \end{cases} \tag{2.20}$$

has 2^{Q-1} sets.

In this calculation, the multiplication process decreases since $W^p x_{k+N/2^l,l-1}$ can be shared. In other words, FFT reduces the computation.

2.1.3 Window Function

It is necessary for the discrete Fourier transform to reduce the influence of the ends of the chosen frame. The reason is that the distortion of the harmonic component occurs when the values of the leftmost and rightmost differ greatly. A window function is applied to the signal in order to improve this situation, and the following two conditions are required for the window function:

(a) Width of main-lobe is small,

(b) Amplitude of side-lobe is small.

(a) shows that the frequency resolution is high. However, there is a trade-off between (a) and (b). That is, the larger the attenuation of the side lobe, the width of the main lobe would increase.

Hanning window, one of the window functions has been used in Haitsma-Kalker algorithm. Equation (2.21) shows the hanning window for $k = 0,\ 1,\ \cdots,\ N-1$. The Hanning window when $N = 256$ is shown in Figure 2.3.

$$w(k) = \begin{cases} 0.5 - 0.5\cos(2\pi k/N-1) & \text{if } 0 \leq k \leq N-1 \\ 0 & \text{if } k < 0,\ N-1 < k. \end{cases} \tag{2.21}$$

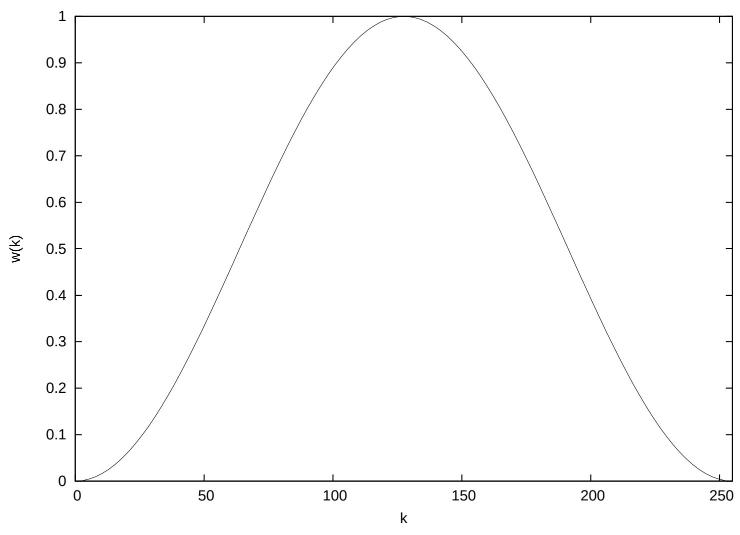

Figure 2.3: Hanning window

In Hanning window, both ends of the chosen signal to be cut out are 0. Both ends outside the interval are weighted by the cosine function. Determining the frequency response by a Fourier transform, the maximum value of the side lobe is found to be -32 dB. The signal analysis is improved by the Hanning window since side lobe is -13 dB when applying a Rectangular window (if a window function is not multiplied).

The Hamming window used in this study is optimized from the Hannning window [10]. Equation (2.22) shows the Hamming window function for $k = 0,\ 1,\ \cdots,\ N-1$. The Hamming window for $N = 256$ is shown in Figure 2.4, where the windows for $k < 0$, and $k > N-1$ are omitted.

$$w(k) = \begin{cases} 0.54 - 0.46\cos\left(2\pi k/N-1\right) & \text{if } 0 \le k \le N-1 \\ 0 & \text{if } k < 0,\ N-1 < k. \end{cases} \tag{2.22}$$

Maximum value of side lobe of Hamming window is -42 dB. The result shows an improvement of about -30 dB, compared with that of a Rectangular window.

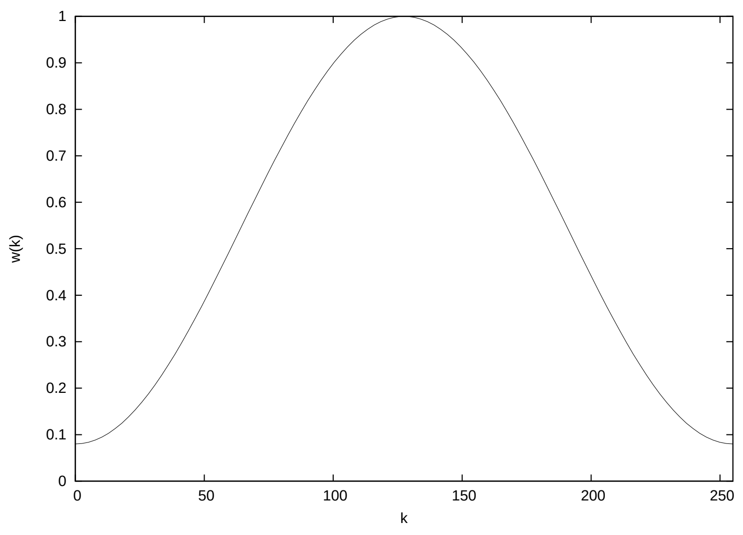

Figure 2.4: Hamming window

2.1.4 Bark Scale

The human auditory system consists of a series of bandpass filters. For example, if there are two tones in the same time, human auditory system always hears one tone while the other becomes inaudible. This phenomenon is caused by increase of the minimum audible range, which is the so-called masking.

The auditory filter is a bandpass filter whose center frequency varies continuously and has the following characteristics:

(a) The frequency analysis of acoustic signal is performed by a bandpass filter which has a closest frequency to the signal.

(b) Those noise components affecting the masking are limited to the frequency components of bandpass filter.

In addition, the frequency bandwidth of the auditory filter is called critical band (CB). Critical band is a function of the center frequency. The critical bandwidth decreases with decreasing center frequency, and increases in size with increasing center frequency. It is called Bark scale when the width of the critical band equals 1.

The following conversion should be applied to transform frequency values f [Hz] into Bark values:

$$\Omega(f) = 13\tan^{-1}(0.76f/1000) + 3.5\tan^{-1}((f/7500)^2). \tag{2.23}$$

As a typical auditory filter, filter with Bark scale can be calculated by the following formula [11]:

$$\Psi(\Omega) = \begin{cases} 0 & \Omega < -1.3 \\ 10^{2.5(\Omega+0.5)} & -1.3 \leq \Omega \leq -0.5 \\ 1 & -0.5 < \Omega < 0.5 \\ 10^{-1.0(\Omega-0.5)} & 0.5 \leq \Omega \leq 2.5 \\ 0 & \Omega > 2.5 \end{cases} \tag{2.24}$$

The frequency band division for audio fingerprint extraction is completed by using the Bark scale as stated above.

2.2 Distance between Audio Fingerprints

The sub-fingerprint is a 32-bit condensed summary 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 [1].

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:

$$BER(A, B) = \frac{1}{32N} \sum_{n=0}^{N-1} \sum_{m=0}^{31} [F_A(n, m) \wedge F_B(n, m)]. \tag{2.25}$$

The operator "$\wedge$" denotes bitwise operation XOR (exclusive or). The numerator of Equation (2.25) calculates the Hamming distance between two fingerprint blocks, which is divided by the bit length of fingerprint blocks ($32N$). That is, the number of different bits between the sub-fingerprints sequences of $F_A(n, m)$, and $F_B(n, m)$. $BER(A, B)$ is the error rate per bit.

Specifically, the length of sub-fingerprint block is $N = 3$ in our study. Let $F_A[n]$ and $F_B[n]$ be the sub-fingerprints of n-th frame extracted from audio clips A and B, the sub-fingerprint blocks can be expressed by

$$F_A[n],\ F_A[n+1],\ F_A[n+2], \tag{2.26}$$

and

$$F_B[n],\ F_B[n+1],\ F_B[n+2] \tag{2.27}$$

respectively.

Values are given in hexadecimal format as follows:

$$\begin{aligned} F_A[n] &= 07\ E4\ FF\ F8, & (2.28)\\ F_A[n+1] &= 07\ E4\ FE\ F8, & (2.29)\\ F_A[n+2] &= 07\ F4\ 7E\ F8. & (2.30) \end{aligned}$$

$$\begin{aligned} F_B[n] &= 1F\ E0\ E7\ FC, & (2.31)\\ F_B[n+1] &= 07\ E0\ FE\ FC, & (2.32)\\ F_B[n+2] &= 07\ F0\ 7E\ 7C. & (2.33) \end{aligned}$$

$F_A[n]$ and $F_B[n]$ can be expressed in binary format as

$$F_A[n] = 0000\ 0111\ 1110\ 0100\ 1111\ 1111\ 1111\ 1000 \tag{2.34}$$

and

$$F_B[n] = 0001\ 1111\ 1110\ 0000\ 1110\ 0111\ 1111\ 1100. \tag{2.35}$$

Hence,

$$F_A[n] \wedge F_B[n] = 0001\ 1000\ 0000\ 0100\ 0001\ 1000\ 0000\ 0100, \tag{2.36}$$

and the Hamming distance between $F_A[n]$ and $F_B[n]$ is 6. Similarly, the Hamming distance between $F_A[n+1]$ and $F_B[n+1]$ is 2, and 3 between $F_A[n+2]$ and $F_B[n+2]$

The Hamming distance between the sub-fingerprint block starting from $F_A[n]$ and $F_B[n]$ respectively becomes 11. According to (2.25), BER can be determined by

$$BER(A, B) = \frac{11}{32 \times 3} = 0.115. \tag{2.37}$$

That is, the bit error rate is 11.5 %

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 scheme of audio fingerprint search can be dipicted by Figure 2.5. The left part of the figure denotes the sub-fingerprint sequece obtained from the query, and the right part shows of that obtained from the music in the database, which contains N songs and i takes values from 0 to $N-1$. As shown in Figure 2.5, audio fingerprint search is actually to calculate the Hamming distance between the sub-fingerprint block Q_x extracted from query and D_y from each song in the database.

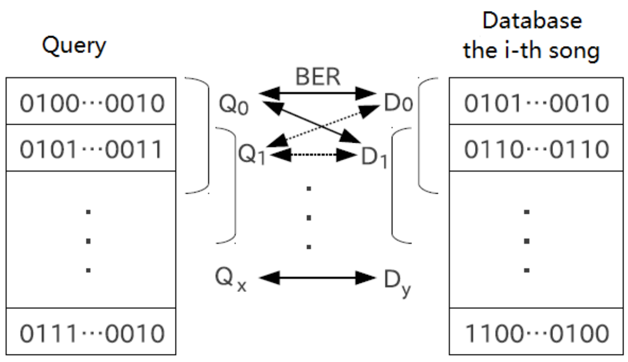

Figure 2.5: Audio fingerprint search

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 [1]. 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 [1], 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. Therefore, a fast Hamming space searching method is urgently required.

Chapter 3

Fast Hamming Space Search based on Query Multiplexing

In this chapter, we propose a fast retrieval method based on query multiplexing using a suffix array for audio fingerprinting systems [12]. 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 Locality-Sensitive Hashing, and then review the suffix array that is employed in our method. Finally, we give a new retrieval method based on query multiplexing.

3.1 Retrieval based on 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 [14], the method for Jaccard distance using min-wise independent permutation [15], the method based on random projection for cosine distance [16], and the method using p-stable distribution for Lp distance [17]. 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.

The audio fingerprint search is actually a Hamming space search. 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 Ham-ming space based on the Point Location in Equal Balls (PLEB) problem [19], also Charikar [16] and Ravichandran et al. [20] improved the algorithm by using random permutations of binary vectors.

The concept of random permutations is as follows: given a set of n vectors $D = d_1, d_2, \cdots, d_n$, where each vector consists of k binary bits, permutation σ is defined as a bijection on $\{1, 2, \cdots, k\}$, 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 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, corresponding data set D_σ and search for all data sets.

This is an overview of the LSH for Hamming space proposed by Charikar [16] and Ravichandran [20]. The details of the theories and experimental analysis of this method are discussed in [16] and [21].

3.2 Suffix Array

The Hamming space search based on query multiplexing uses a suffix array *(SA)* to search audio fingerprint efficiently. We are to elaborate the suffix array in this section.

3.2.1 Definition of Suffix Array

Suffix array is defined to be an array of integers providing the starting positions of suffixes in lexicographical order. It is quite commonly applied in string search and full text indices. We use a suffix array for string search as an example [22].

As the same as the original string, all the suffixes end at the same place with the special sentinel letter that is unique, but start at different positions. Given a string of length n

$$T = t_0\ t_1\ \cdots\ t_{n-1}, \tag{3.1}$$

the suffixes of T can be expressed as

$$T_i = t_i\ t_{i+1}\ \cdots\ t_{n-1}, \tag{3.2}$$

in which $i = 0,\ 1,\ \cdots,\ n-1$.

For example, there are 11 suffixes given a string of "mississippi", including "mississippi", "ississippi", "ssissippi", "sissippi", "issippi", "ssippi", "sippi", "ippi", "ppi", "pi", and "i", as shown as in Table 3.1

The array keeping the starting positions of suffixes in lexicographical order is so-called suffix array. That is, if

$$T_{SA[0]} < T_{SA[1]} < \cdots < T_{SA[n-1]}, \tag{3.3}$$

Table 3.1: Suffixes of “mississippi”

Starting position	Suffix
0	mississippi
1	ississippi
2	ssissippi
3	sissippi
4	issippi
5	ssippi
6	sippi
7	ippi
8	ppi
9	pi
10	i

the suffix array should be

$$SA = SA[0],\ SA[1],\ \cdots,\ SA[n-1]. \tag{3.4}$$

In Equation (3.3), “<” refers to an lexicographical order. Suffixes of “mississippi” can be sorted in lexicographical order as shown in Table 3.2, where

$$SA = 10,\ 7,\ 4,\ 1,\ 0,\ 9,\ 8,\ 6,\ 3,\ 5,\ 2. \tag{3.5}$$

Table 3.2: Sorted suffixes of “mississippi”

Starting position	Sorted suffix
10	i
7	ippi
4	issippi
1	ississippi
0	mississippi
9	pi
8	ppi
6	sippi
3	sissippi
5	ssippi
2	ssissippi

3.2.2 Binary Search on Suffix Array

The string search based on suffix array is to be desribed in this subsection. The original string and suffix array are used in the string search based on suffix array. The same string of “mississippi” is used for example. Let T denote the original string and SA represent the suffix array.

Table 3.3: Data T and sorted positions SA

i	T	SA	Sorted suffix
0	m	10	i
1	i	7	ippi
2	s	4	issippi
3	s	1	ississippi
4	i	0	mississippi
5	s	9	pi
6	s	8	ppi
7	i	6	sippi
8	p	3	sissippi
9	p	5	ssippi
10	i	2	ssissippi

The binary search is available since SA has stored the positions of sorted suffixes. Binary search is an algorithm to find the position of a target value among the sorted data. In each step, the alogorithm compares the target value with the value of the middle element of the array to halve the searching data. If the values match, then a matching element has been found and its position is retured as the result. Otherwise, if the target value is smaller than the middle element, then the algorithm repeats on the sub-array to the left of the middle element. If the target value is bigger, it repeats to the right of the middle element.

The process of binary search algorithm on suffix array is demonstrated by a pseudocode bellow. Let Q be the string to be searched on and T be the target string, SA presents the suffix array of T and n is the length of T and SA. T_x refers to T_x, and it is assumed that Q is included in T in this algorithm.

```
function binary_search(char *Q, char *T, char *SA, int n)
```

```
{
    int left = 0;
    int right = n-1;
    int middle = ( left + right ) / 2;

    while ( 1 )
       { middle = ( left + right ) / 2;

         if ( T_SA[mid] > Q )
           { right = middle;}
         else if ( T_SA[middle] < Q )
           { left = middle }
         else {
                return ( SA[middle] );
                }
        }
}
```

For example, searching a substring Q = "sip" from string T = "mississippi", the process is as follows:

1. $left = 0$, $right = 10$, thus $mid = 10/2 = 5$,
   $T_{SA[5]} = T_9 =$ "pi".
   Since $T_9 < Q$, $left = 5$, the algorithm repeats to the right.
2. $left = 5$, $right = 10$, therefore, $mid = 15/2 = 7$,
   $T_{SA[7]} = T_6 =$ "sippi".
   $T_6 == Q$, the algorithm ends since Q has been matched.

Through the above process, "sip" is found to be included in "mississippi".

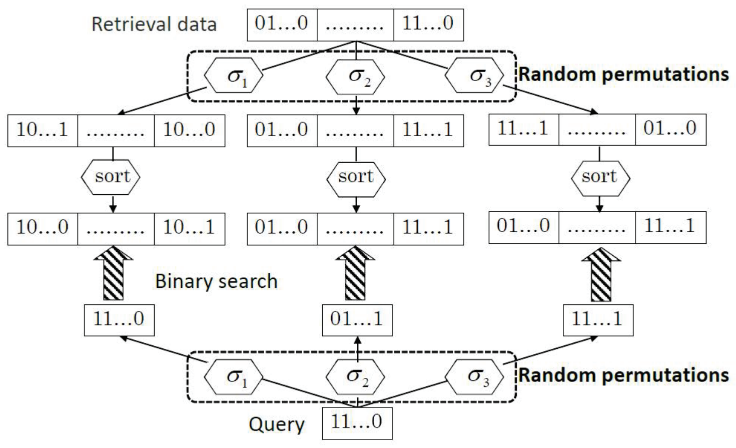

Figure 3.1: Schematic diagram of search based on random permutation

3.3 Hamming Space Search based on Query Multiplexing

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. The scheme of the search method based on random permutations is shown in Figure 3.1.

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. However, 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.

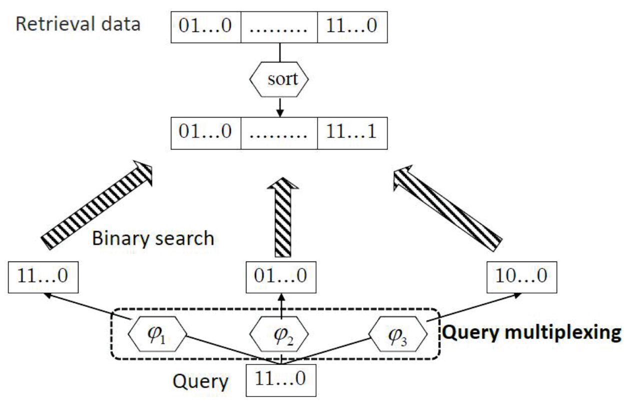

Figure 3.2: Schematic diagram of search based on query multiplexing

Based on the assumption that if one might multiplex the query vectors without expanding the database, then Hamming space searching would require little memory, the fast Hamming space search method based on query multiplexing for audio fingerprinting [12] is proposed. The scheme of the proposed method is shown in Figure 3.2. In the random permutation method, multiple random permutations (σ_1, σ_2 and σ_3 in Figure 3.1) 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 3.2). The definition of functions φ_i is necessarily application-dependent.

3.3.1 Suffix Array Method

Suppose $FP_i[j]$ be the sub-fingerprint extracted from the j-th frame of song i, n_i be the total number of sub-fingerprint of song i, the sub-fingerprint of song i should be

$$FP_i = FP_i[0],\ FP_i[1],\ \cdots,\ FP_i[n_i - 1]. \tag{3.6}$$

Given a database consisting of N songs, i ranges from 0 to $N-1$. Let FP denote the sub-fingerprints obtained from all the songs in a database, it can be expressed as

$$\begin{aligned} FP &= FP_0,\ FP_1,\ \cdots,\ FP_{N-1} \\ &= FP_0[0],\ FP_0[1],\ \cdots,\ FP_{N-1}[n_{N-1}-1] \\ &= FP[0],\ FP[1],\ \cdots,\ FP[n-1]. \end{aligned} \tag{3.7}$$

Moreover, the suffix array should be

$$SA = SA[0],\ SA[1],\ \cdots,\ SA[n-1]. \tag{3.8}$$

$SA[i]$ in Equation (3.8) keeps the sorted positions of the sub-fingerprint of length 3 as follows:

$$FP[i],\ FP[i+1],\ FP[i+2]. \tag{3.9}$$

FP and the sorted positions SA stated above are kept as the index of database. The length of SA for an n-length FP is also n. In other words, SA and FP has the same data size, and thus the total data size is twice the size of sub-fingerprint. That is, the total size of n-length FP is $2n$. However, the binary search can be used for search since SA has kept the positions of sorted data.

Figure 3.3 shows the scheme of the method based on suffix array.

3.3.2 Music Retrieval based on Suffix Array

This method is based on the sub-fingerprint matching scheme. The functions φ_i stated above create the multiplexed search queries of sub-fingerprint sequences from the query audio clip [13]. By multiplexing queries of the sub-fingerprint sequence instead of expanding the database vectors, the method greatly reduces the searching space. Many sub-fingerprints are extracted by shifting the query into frames.

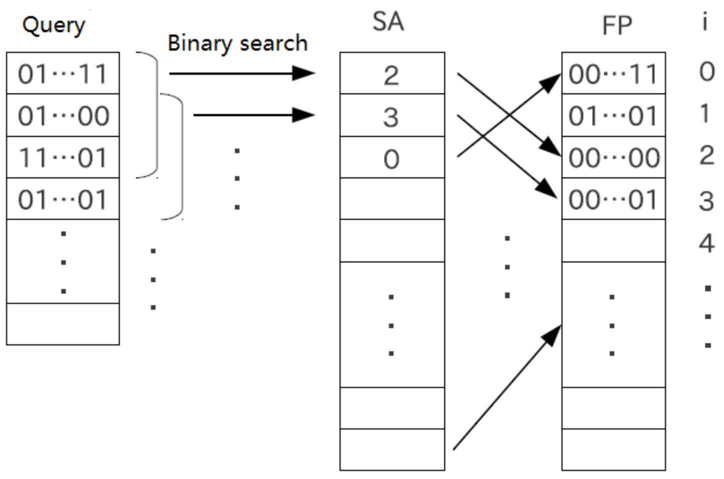

Figure 3.3: The scheme of the method based on suffix array

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 makes 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: firstly, estimate several candidates of starting position in the database those using sub-fingerprints obtained from the query. Secondly, 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 sub-fingerprints (SSF) is employed for matching. Let $FP = FP_1, FP_2, \cdots, FP_N$ denote the sub-fingerprints obtained from all the songs in a database, many m-length ($SSFs$) (m is set to be 3 in [12]) can be derived by changing the starting position of the fingerprint, and the i-th sub-fingerprint sequence is defined as $SSF_i = (FP_i, FP_i{+}1, \cdots, FP_i{+}m{-}1)$. Then, all $SSFs$ are sorted by value and their positions are stored in a one-dimensional array $S = S_1, S_2, \cdots, S_N - m + 1$. Array S, the same as a suffix array [22], contains the indexes to FP, and satisfies

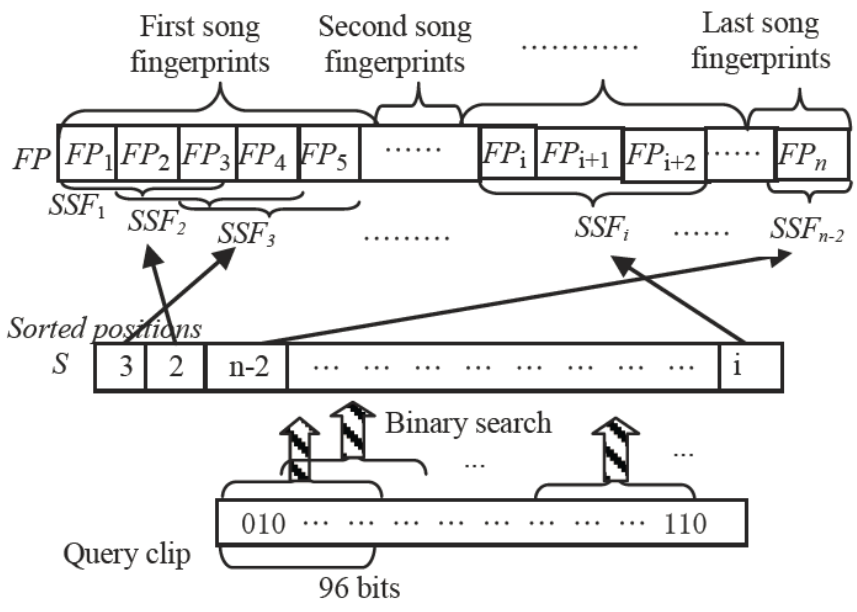

Figure 3.4: Schematic diagram of sub-fingerprint sequence search

the following:

$$S_j = i \qquad \text{iff} \qquad SSF_i = (FP_i, FP_i + 1, \cdots, FP_i + m - 1)$$
$$\text{is the } j\text{-th SSF in sorted oder} \tag{3.10}$$

The search process, as shown in Figure 3.4, can be summarized in the following phases:

(1) Extract the sub-fingerprint sequence *FP* from query music.

(2) For all *SSFs* of query music, find candidate positions where the sub-fingerprints obtained from the query locating in the database by performing a binary search on array S.

(3) Assign to candidate position the start position of the *FP*, and calculate the Hamming distance (bit error rate) between *FP* and the fingerprint block (128 sub-fingerprints) corresponding to the *SSF* .

(4) Finally, output the top n songs as final results.

In the binary search on array S, most similar *SSFs* can be found by checking the neighbourhood positions of the searched block in array S. An index size that is proportional to the length of the sub-fingerprint sequence in the database enables the memory/storage required to be much less than required by conventional methods such as the method based on random permutations.

Specifically, the process of music retrieval based on suffix array is desmonstrated below:

1. Let $Q_{i,\ i+2}$ be the 3-th sub-fingerprint derived from query Q,

$$Q_{i,\ i+2} = Q[i],\ Q[i+1],\ Q[i+2]. \tag{3.11}$$

   i ranges from 0 to $n_q - 128$ if the length of Q is n_q. $FP_{i,\ i+2}$ refers to the sub-fingerprint sequence derived from the database, and

$$\begin{aligned} & FP_{SA[j],\ SA[j]+2} \\ = \ & FP[SA[j]],\ FP[SA[j]+1],\ FP[SA[j]+2]. \end{aligned} \tag{3.12}$$

   For each $Q_{i,\ i+2}$ $(i = 0,\ 1,\ \cdots, n_q - 128)$, find the $FP_{SA[j],\ SA[j]+2}$ that makes bit error rate 0. The matching of sub-fingerprint block is to be performed for the $FP_{SA[j],\ SA[j]+2}$ derived above. The similarity is also determined by the error rate even for the vicinity of the position where it has been matched in binary search, so the matching with each sub-fingerprint block starting with $Q[i]$ becomes multiple.

2. For all $FP_{SA[j],\ SA[j]+2}$ those match $Q_{i,\ i+2}$, calculate the similarity of the two sub-fingerprint blocks. For the length of sub-fingerprint block equals 128,

calculate the error bit rate between $Q_{i,\ i+127}$ and $FP_{SA[j],\ SA[j]+127}$ as follows.

$$Q_{i,\ i+127} = Q[i],\ Q[i+1],\ \cdots,\ Q[i+127], \tag{3.13}$$

$$\begin{aligned} & FP_{SA[j],\ SA[j]+127} \\ = \ & FP[SA[j]],\ FP[SA[j]+1],\ \cdots,\ FP[SA[j]+127]. \end{aligned} \tag{3.14}$$

Figure 3.5 shows the process. Those songs containing a sub-fingerprint block of $FP_{SA[j],\ SA[j]+127}$ who has a bit error rate bellow the threshold are selected as the candidates.

In Figure 3.5, there is only one sub-fingerprint block of $FP_{SA[j],\ SA[j]+127}$ in the database matched $Q_{i,\ i+127}$. However, $FP_{SA[j],\ SA[j]+127}$ should be multiple actually, and multiple candidates of sub-fingerprint block for each $Q_{i,\ i+127}$ can be derived.

3. Oder all the candidates in ascending oder of bit error rate with $Q_{i,\ i+127}$. Output the songs containing the top sub-fingerprint block of $FP_{SA[j],\ SA[j]+127}$ as final results.

3.3.3 Acquisition of Music Information

Let j indicate the j-th song in database, and $POS[j]$ be the starting position of sub-fingerprint of song j in the sub-fingerprint sequence FP of all the songs in database. All positions are stored as

$$POS = POS[0],\ POS,\ \cdots,\ POS[N-1], \tag{3.15}$$

as shown in Figure 3.6. N is the number of songs in database. $n_0 = 0$ and $n_1 = 100$, that is, sub-fingerprint of song 0 covers from $FP[0]$ to $FP[99]$, and song 1 from

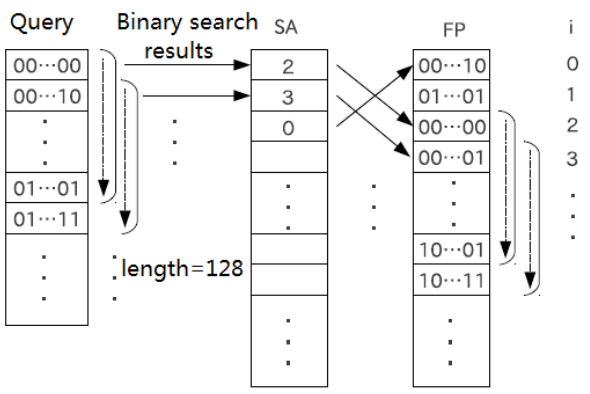

Figure 3.5: Sub-fingerprint block matching based on SA

$FP[100]$ to $FP[n_2]$.

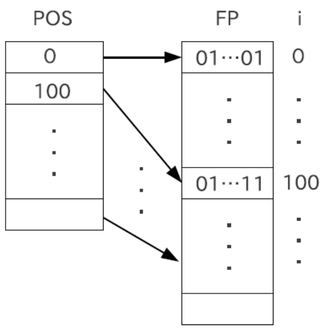

Figure 3.6: Acquisition of music information

For instance, suppose the information of $FP[i]$ to be searched, given

$$POS[j] \leq i < POS[j+1], \tag{3.16}$$

$FP[50]$ is found to be the sub-fingerprint of song 0.

3.4 Experiments and Results

We carried the experiments on a database of 8,740 original songs and 800 artificially corrupted and 268 original queries, in order to evaluate the fast Hamming space search based on query multiplexing.

3.4.1 Music Data

The database contained 8, 740 songs in mp3 format from CDs or the Internet. There were many genres in the database such as pop, classical, and folk music. An index of the number of each song was created. 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.

3.4.2 Acoustical Analysis Settings

The Haitsma-Kalker algorithm was used for fingerprints extraction in our experiments. However, there were some different points: 1) the length of each frame is 1.024 seconds, 2) 32 milliseconds for frame shift, 3) an improved Hamming window; and 4) the length of sub-fingerprints block is 128 instead of 256.

Although these settings seem rough compared with those given by Haitsma and Kalker [1], these parameters were determined by many preliminary experiments and the resulting proposed algorithm gave a high accuracy.

3.4.3 Experimental Results

The query multiplexing and binary search reduced search space and time greatly. Retrieval times varied as the query music, and each song was retrieved in approximately 0.4 to 0.6 seconds. Moreover, the retrieval time per sub-fingerprint block did not exceed 0.1 milliseconds. The algorithm is considered competent and fast.

The retrieval accuracy on real data is shown in Table 3.4. The retrieval rate for

Table 3.4: Results on real music data

Category		Notes	Audio Music	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 3.5: Results on corrupted music data

Corruption		Accuracy
superimposition of white noise	-5dB	98.5%
	0dB	100%
	10dB	100%
Low-pass filter	1kHz	100%

"*original music*" was 96.2%, and that 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 were divided into two classes with regard to noise, but the results did not show any influence of noise. Experimental results showed that the proposed method delivered accurate, fast retrieval.

Results on artificially corrupted music data shown in Table 3.5 indicated that the method was highly robust to superimposition of white noise, and it was sufficient to retrieve the music if there was low bass left.

Chapter 4

Audio Fingerprinting Systems in Real-Noise Conditions

4.1 Related Work

With the development of music compression technology and widespread availability of recorded music, content-based music information retrieval has become one of the most attractive application services to be pursued by many companies in recent years [23][24]. But a smart phone can be used anywhere, including some fairly noisy places such as shopping malls or playgrounds (i.e., varieties of external noise are unavoidable and should be taken into account) [25], that is, the query music could be distorted by external noise. For this reason, it is necessary to examine the robustness of audio fingerprinting systems against real noise in practical applications [26]. In this chapter, we evaluate the performance of audio fingerprinting systems in real-noise conditions by adding noise data (ten types of noise data) to music data in accordance with signal-to-noise ratios (SNR) [27].

There are several studies having reported on the audio fingerprinting algorithm under additive noise. Balado et al. [28] and Doets et al. [29] focused on the bit error rate (BER) under additive white-noise conditions, but both failed to consider robustness against varieties of noise in a real environment. Furthermore, although Park et al. [30][31] discussed the performance of music retrieval in five types of noise conditions, their study lacked sufficient discussion on the relationship between the retrieval accuracy and frequency-temporal characteristics under each type of noise.

Moreover, we focus on the relationship between retrieval accuracy and noise type, as well as figuring out the type of noise against which robustness achieves high [32]. For the search algorithm, we employ a fast Hamming space search method proposed in [12].

4.2 Experiments and Results

First, we carried out experiments on the original query data before superimposing the noise data. Second, we used the noisy query data in experiments to ascertain how much the noise in a real environment affects the performance of audio fingerprinting systems.

4.2.1 Music and Noise Data

In the experiments, we used the database of 8,740 songs in mp3 format that was used in [12]. First, 104 songs were selected randomly from the database as the original query data, and then another 96 songs were added to the query data to cover as many genres as possible (e.g., pop performances, hip-hop, and classical). These original query data were downsampled from 44.1 kHz to 16 kHz and quantized to a 2-channel 16-bit linear PCM in wav format.

We selected ten types of noise data from the JEIDANOISE database, corresponding to environments in which a smart phone is most likely to be used for music retrieval. The details of the noise data are described in Table 4.1, according to the locations and main noise sources [33]:

We generated the noisy query data by adding the noise data to the original query

Table 4.1: Noise data classification

Noise types		Main noise sources
Noise canrs(2000cc) (N1)		Aerodynamic noise and engine noise
Trains (N2)		Gap between rail cars, pantograph and lower part of cars (aerodynamic noise from bogies, etc.)
Crossroad (N3)		Cars, bikes, and engine noise when waiting
Trunk road (N4)		Cars and bikes
Elevator hall of the hotel (N5)		Footsteps, human speech and the bell of the elevator
Crowds (N6)		Human speech
Vicinity of ticket vending machines of station	In concourse (N7-a)	Human speech, falling coins and footsteps
	Beside the road (N7-b)	Human speech, falling coins, footsteps, payphones, and running trains
Exhibition hall	In booth (N8-a)	Human speech, music and sound effects of the events through speakers
	Passage (N8-b)	Human speech and door opening/closing of exit/entrance

data in accordance with the SNR, and the noisy query data were then compressed to mp3 format.

4.2.2 Experimental Results

Experiments carried out on the original query data achieved an accuracy of 100%. By contrast, Table 4.2 and Table 4.3 show the results of our experiments carried out on the noisy query data. Table 4.2 provides the results from mechanical noises that contained little human speech. These noises consisted mainly of noise from running cars or trains. Table 4.3 shows the results from human noise, mainly from human speech.

Table 4.2: Accuracy under mechanical noise

SNR	Accuracy(%)			
	N1	N2	N3	N4
20dB	99.5	98.5	93.5	92.5
10dB	96.0	93.5	75.5	75.0
0dB	78.0	77.0	68.0	67.5
-5dB	74.5	73.0	61.5	58.5
20dB	72.0	68.5	51.5	48.0

Table 4.3: Accuracy under human noise

SNR	Accuracy(%)					
	N5	N6	N7-a	N7-b	N8-a	N8-b
20dB	88.5	76.0	74.5	77.0	72.5	73.5
10dB	73.5	71.5	66.0	67.5	64.0	67.5
0dB	65.5	57.5	47.0	51.5	27.5	49.5
-5dB	56.5	36.0	7.0	24.0	0.5	20.0
20dB	34.5	2.0	1.5	0	0	0

4.2.3 Impact of High Frequency Bands

As stated in Section 2, the audio fingerprinting extraction is based on the calculation of energy differences between the successive frequency bands. In addition, the impact

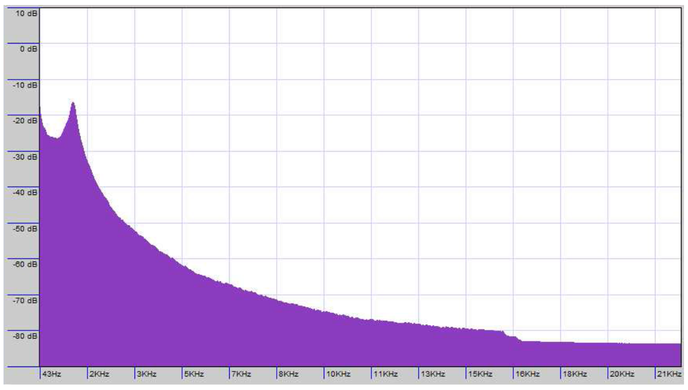

Figure 4.1: Noise of frequency band near 1khHz

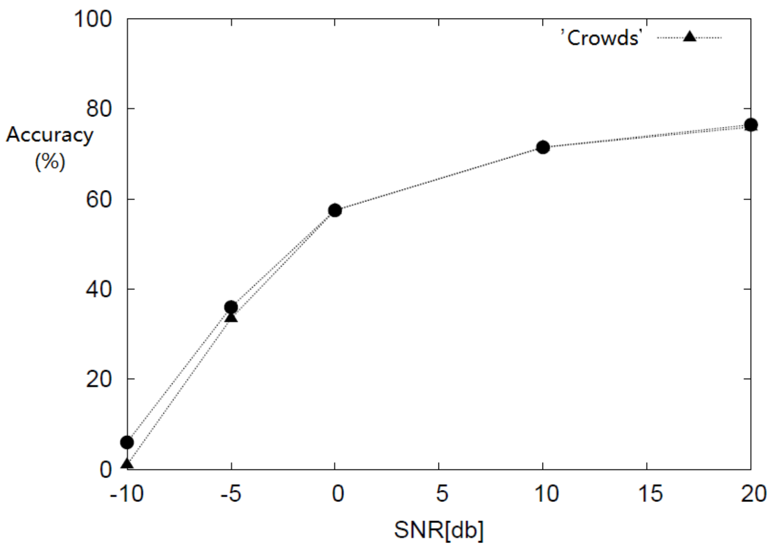

Figure 4.2: Accuracy under the noise of frequency band near 1khHz

of noise on accuracy depends on how much noise that affects the characteristics included. In other words, the more the high frequency bands of noise are within the range of the frequency bands used in audio fingerprinting (from 300Hz to 2 kHz), the more severe their influence on accuracy.

Figure 4.1 dipicts the noise produced by computer. Audio fingerprint uses low frequency bands, and the noise lies in high frequency bands near 1khHz, therefore, the energy difference would be affected by the decrease of SNR. Figure 4.2 shows the accuracy under the noise of frequency band near 1khHz.

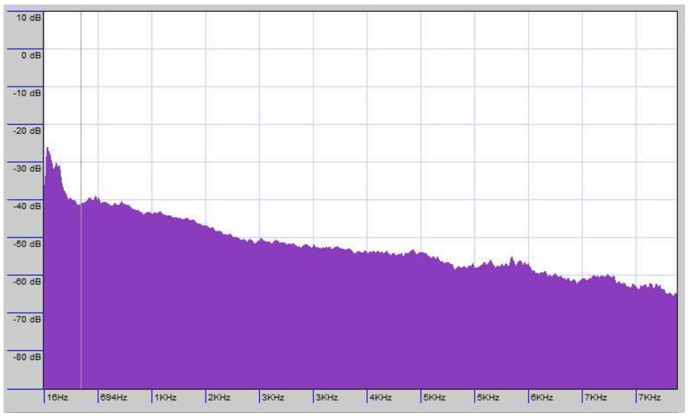

Figure 4.3: Noise from crossing

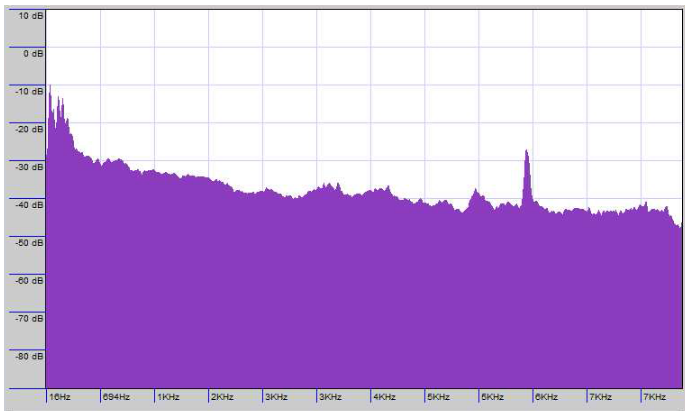

Figure 4.4: Noise from trunk road

As shown in Table 4.2, the experiments perform well under each type of mechanical noise. The accuracy maintains above 48%, even with an SNR of -10dB. This indicates that there is still music information left for the energy difference in the frequency bands which have been used in audio fingerprinting, even if noise data are superimposed on it.

For noise from N1 (car interior, shown in Figure 4.5), as well as N3 in Figure 4.3 and N4 in Figure 4.4, the higher frequency bands are lower than 300Hz and thus have little impact on accuracy. For N2, the noise from trains produced by the wheels above 1600Hz and bogies in a range from 500Hz to 800 Hz (see Figure 4.6) falls to a greater extent in the frequency bands used in audio fingerprinting, and the accuracy remains high. This is attributable to multiple standstills during which the

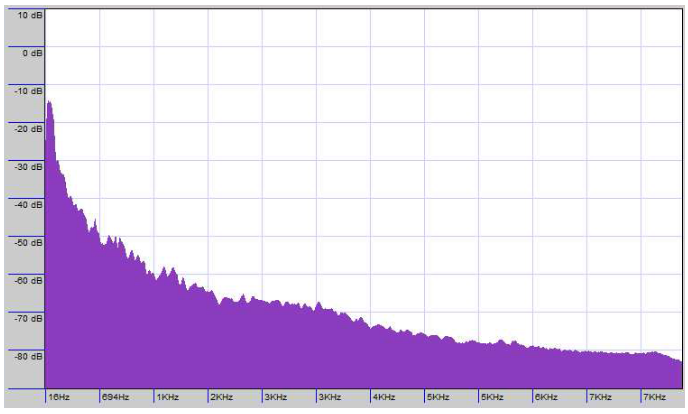

Figure 4.5: Noise from running cars (inner car)

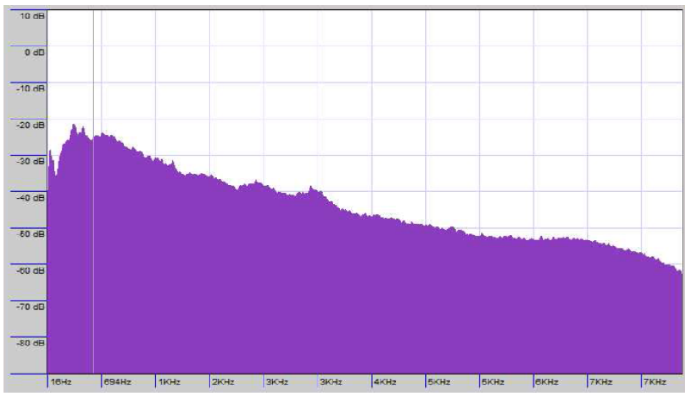

Figure 4.6: Noise from trains in operation

noise declines to a low level (see Figure 4.7).

With respect to noise that is mainly from human beings (shown in Table 4.3), except for N5 (elevator hall), the accuracy falls swiftly with an SNR of -10dB (e.g., noise from crowds as shown in Figure 4.8). The reason is that the energy of the human voice concentrates in relatively low frequency bands and the long-term average spectrums are nearly flat up to 800Hz. However, in an elevator hall, the accuracy remains above 34% even with an SNR of below -10dB. This is because human speech is so sparse that only footsteps are heard and high frequency bands such as those of an elevator bell only lie above 3 kHz infrequently (see Figure 4.9). Hence the energy difference of human speech is small and it does not perform poorly. For noise in the vicinity of ticket vending machines, the high frequency bands of noise by the road

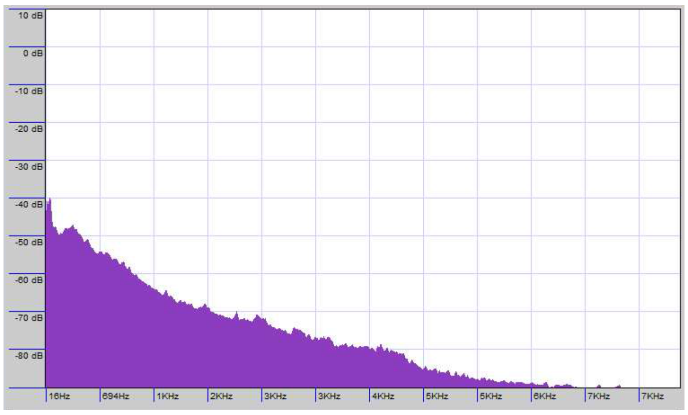

Figure 4.7: Noise from trains during standstills

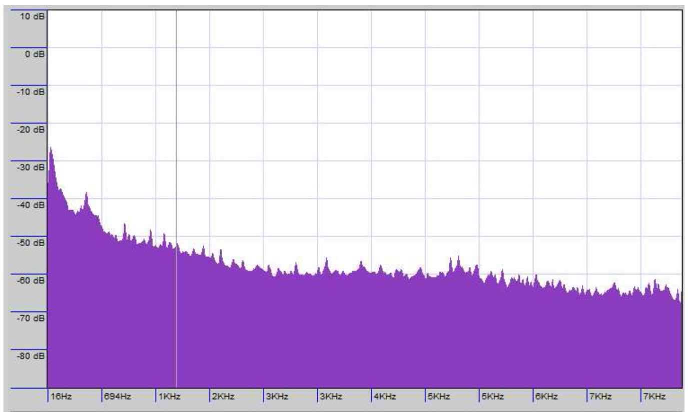

Figure 4.8: Noise from crowds

side (see Figure 4.10) lie above 3kHz, therefore the accuracy is higher than that of noise in the station (see Figure 4.11). For the same reason, the accuracy of noise from the passage of exhibition hall (see Figure 4.12) is higher than that of booth (see Figure 4.13).

4.2.4 Impact of Temporal Changes

Temporal changes in noise can also impact results. On one hand, the noise level is very low during standstills (e.g., a quiet period after one train passes and until the next train comes), which is one of the reasons why results under the noise from trains and the elevator hall are better. On the other hand , the noise level can also drop in a relatively quiet period.

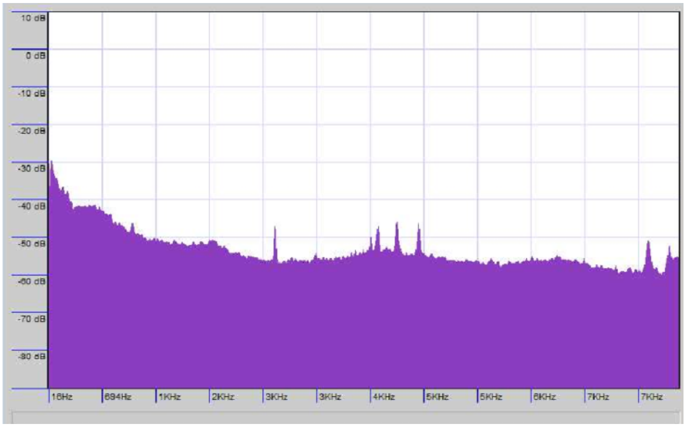

Figure 4.9: Noise from elevator hall

Table 4.4: Numbers of approximate-silent intervals in noise data

Silence length (s)	Numbers of intervals		
	N1	N2	N3
0.37-1	2	8	14
1-3	3	1	9
3-5	1	1	5
>5	3	2	1

To ascertain how much a relatively quiet period affects the results, we investigated the number of approximate-silent intervals during a 2-minute period of the noise data, according to the frame length used in the Haitsma-Kalker algorithm: 0.37s. Here, the approximate-silent interval refers to the sound of noise data below 26dB. The results are shown in Table 4.4.

There are multiple approximate-silent intervals in noise from N1, N2 and N3, which helps to improve the accuracy of the retrieval. For N2, although it is quite noisy when trains travel an intersection, the noise drops quite low during standstills to such an extent that we may not feel it, therefore the characteristics are believed to remain in the standstills and it is still possible to recognize the music with a relatively high accuracy. Although there are no approximate-silent intervals in the noise from N4 (trunk road) and N5 (elevator hall), the accuracy stays high due to the existence of long quiet intervals. Among the four types of noise in Table 4.2, the

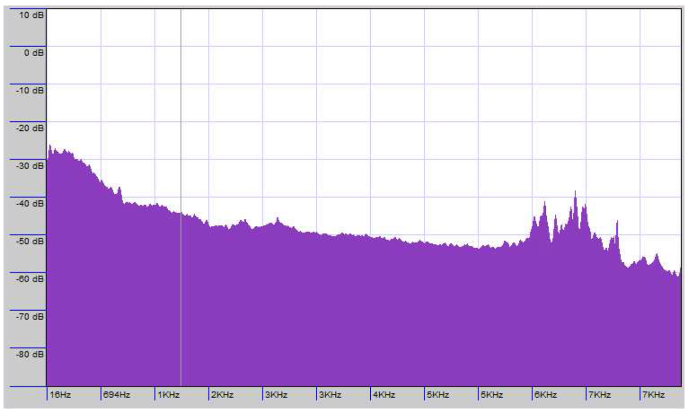

Figure 4.10: Noise from vending machine by the road side

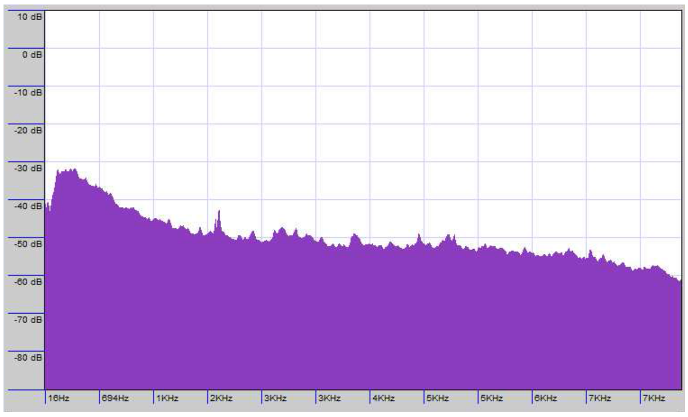

Figure 4.11: Noise from vending machine in the station

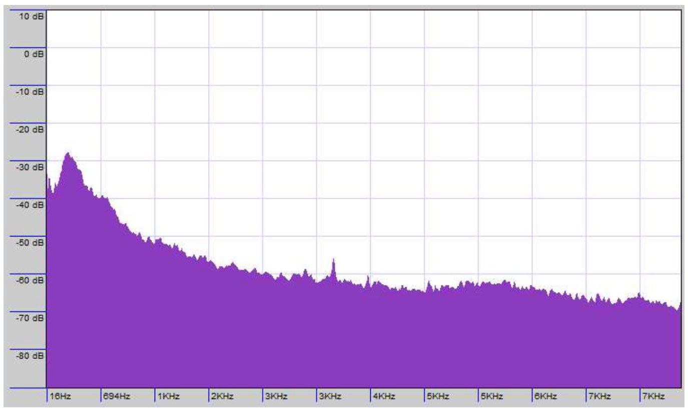

Figure 4.12: Noise from the passage of exhibition hall

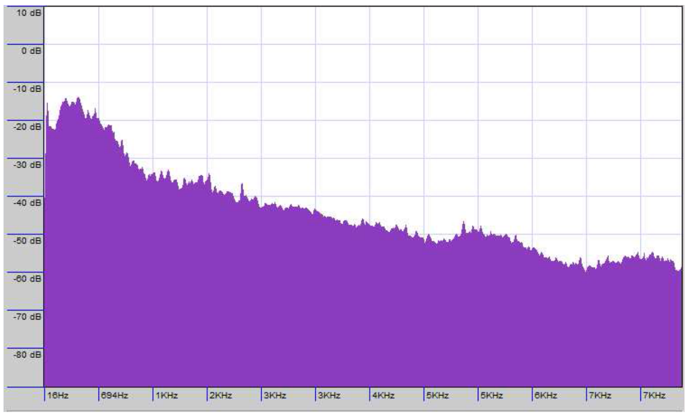

Figure 4.13: Noise from the booth

experiment performs worst for all SNR under noise from N4, probably due to the fact that the cars move too quickly and noisily when travelling the trunk road.

The experiments show that the retrieval accuracy under noise from running cars remains higher than other noise when lowering the SNR, even if the original music could not be heard clearly. The reason is that the noise is high in frequency bands lower than 300Hz, which makes the influence on retrieval accuracy small. For noise data containing standstills, such as noise from trains and an elevator hall, it is possible to retrieve a portion of music due to the low noise at the standstills, which results in less impact on the retrieval accuracy. For noise that is mainly from human speech, however, the accuracy drops sharply with an SNR of -10dB. The frequency bands of human speech correspond to a greater degree to the frequency bands used as characteristics in audio fingerprinting; therefore, the more noise in these frequencies, the greater effect on retrieval performance.

Chapter 5

Index Compression based on Compressed Suffix Array

In this chapter, we elaborate the proposed method of index compression for music retrieval by using a compressed suffix array [34][35]. By taking advantage of the fact that the repetitive characters occur frequently in high bits of the sorted audio fingerprint data, the proposed method compresses the index of the sorted sub-fingerprints by encoding the 8-bit data sequences with the Run Length Encoding [36].

5.1 Compressed Suffix Array

5.1.1 Retention of Sorted Data

The proposed method does not directly maintain the sub-fingerpints *FP* those are extracted from all the songs in the database [37]. Instead, the sorted sub-fingerprints are kept. We use *SFP* to denote the sub-fingerprints, which have been sorted. That is, use

$$SFP = SFP[0],\ SFP[1],\ \cdots,\ SFP[n-1] \tag{5.1}$$

where

$$SFP[i] = FP[SA[i]] \tag{5.2}$$

to reduce the space cost by keeping the compressed *SFP*. In our method, *SA* is different from the conventional methods based on suffix array, because we only use the complete suffix.

$SA[i]$ in (5.2) stores the positions of suffixes of the sorted sub-fingerpints sequence, which is expressed by

$$FP[i],\ FP[i+1],\ \cdots,\ FP[n-1]. \tag{5.3}$$

SFP is split into pieces in an interval of M_1. Suppose SFP_k as the block of *SFP*,

$$SFP = SFP_0,\ SFP_1,\ \cdots,\ SFP_{n/M_1} \tag{5.4}$$

can be derived from

$$SFP_k = SFP[kM_1],\ SFP[kM_1+1],\ \cdots,\ SFP[(k+1)M_1-1]. \tag{5.5}$$

The reason of using the block as the process unit of *SFP* is that the required restoration should be performed instantaneously during searching.

5.1.2 Run Length Encoding

In order to compress the sorted sub-fingprints *SFP*, the Run Length Encoding (RLE) is performed for each segmented block SFP_k. RLE is an encoding technique, in which a series of repetitive data symbols are compressed into a shorter code, which indicates the length of a code and the data being repeated. RLE is not suitable for many kinds of data, because the compression efficiency would be deteriorated as the run length gets shorter. In other words, the index cannot be compressed with the RLE directly since there are 2^{32} kinds of data in the sorted sub-fingerprints *SFP*.

The proposed method divides each sub-fingerprint (32 bits, 4 bytes) into units of 8 bits (one byte), and then perform Run-Length Encoding (RLE) over the 8-bit data sequences. The repetitive characters in the same unit are removed from the

8-bit sequence.

Given a sorted sub-fingerprint block SFP_k, the value of the m-th byte can be represented by $SFP_k(n,\ m)$. The m-th byte of the 8-bit data sequence is expressed as

$$SFP_k(0,\ m),\ SFP_k(1,\ m),\ \cdots,\ SFP_k(M_1 - 1,\ m), \tag{5.6}$$

where $m = 0,\ 1,\ 2,\ 3$, since the sub-fingprint is 32-bit.

The 8-bit sequence is arranged because the bit values in the upper side are probably to be the same since *SFP* has been sorted. And thus it is easy to compress. For example, given that the segmentation interval M_1 ($M_1 = 8$) of the sub-fingerprint *SFP*, the SFP_0 can be expressed as

$$\begin{aligned} SFP_0 = \;& 00\ \ 00\ \ 00\ \ 00, \\ & 00\ \ 00\ \ 00\ \ 00, \\ & 00\ \ 00\ \ 00\ \ 01, \\ & 00\ \ 00\ \ 00\ \ 01, \\ & 00\ \ 00\ \ 01\ \ 01, \\ & 00\ \ 00\ \ 01\ \ 01, \\ & 00\ \ 00\ \ 11\ \ 01, \\ & 00\ \ 00\ \ 11\ \ 11 \end{aligned} \tag{5.7}$$

By fetching values of the highest bits in SFP_0 only, we can get

$$\begin{aligned} & SFP_0(0,\ 3),\ SFP_0(1,\ 3),\ \cdots,\ SFP_0(7,\ 3) \\ & = 00,\ 00,\ 00,\ 00,\ 00,\ 00,\ 00,\ 00 \end{aligned} \tag{5.8}$$

By performing RLE, the values in Equation 5.8 becomes values in Equation (5.9) since the repetitive characters of 00 occurs 8 times.

$$00,\ 00,\ 08 \tag{5.9}$$

Similarly, the second highest byte in SFP_0 shown as in (5.10). And by performing RLE, we can get the same encoded data as in (5.9).

$$\begin{aligned}&SFP_0(0,\ 2),\ SFP_0(1,\ 2),\ \cdots,\ SFP_0(7,\ 2)\\&= 00,\ 00,\ 00,\ 00,\ 00,\ 00,\ 00,\ 00.\end{aligned} \tag{5.10}$$

The third highest byte in SFP_0 is

$$\begin{aligned}&SFP_0(0,\ 1),\ SFP_0(1,\ 1),\ \cdots,\ SFP_0(7,\ 1)\\&= 00,\ 00,\ 00,\ 00,\ 01,\ 01,\ 11,\ 11\end{aligned} \tag{5.11}$$

where 00 occurs four times, 01 and 11 occur twice respectively. So we can get

$$00,\ 00,\ 04,\ 01,\ 01,\ 02,\ 11,\ 11,\ 02. \tag{5.12}$$

Finally, the lowest byte is shown as

$$\begin{aligned}&SFP_0(0,\ 0),\ SFP_0(1,\ 0),\ \cdots,\ SFP_0(7,\ 0)\\&= 00,\ 00,\ 01,\ 01,\ 01,\ 01,\ 01,\ 11\end{aligned} \tag{5.13}$$

and its RLE encoded data is

$$00,\ 00,\ 02,\ 01,\ 01,\ 05,\ 11. \tag{5.14}$$

By replacing the 8-bit data sequences in SFP_0 with the derived codes in (5.9), (5.12) and (5.14), we get the RLE encoded data as

$$\begin{aligned} EFP_0 = \; & 00 \quad 00 \quad 08 \quad 00, \\ & 00 \quad 08 \quad 00 \quad 00, \\ & 04 \quad 01 \quad 01 \quad 02, \\ & 11 \quad 11 \quad 02 \quad 00, \\ & 00 \quad 02 \quad 01 \quad 01, \\ & 05 \quad 11. \end{aligned} \tag{5.15}$$

In other words, SFP_0 shown as (5.7) becomes (5.15) by RLE. The bigger the size of database, the longer the Run (a series of repetitive characters) length becomes and subsequently the compression efficiency increases.

The sorted sub-fingerprints SFP is compressed and maintained by EFP as the sub-fingerprints of the database. EFP is the encoded data sequence of SFP_k.

$$EFP = EFP_0,\ EFP_1,\ \cdots,\ EFP_{n/M_1} \tag{5.16}$$

In addition, information at the end of block is required because EFP_k is changeable in length. Let LEN_k be the length of EFP_k, the tail information of EFP_k is retained in

$$LEN = LEN_0,\ LEN_1,\ \cdots,\ LEN_{n/M_1}. \tag{5.17}$$

5.1.3 Compression of the Data Order

Since the order of the original database has been lost, we use Ψ as shown in (5.18) to represent the original order.

$$\Psi[i] = \begin{cases} SA^{-1}[SA[i]+1] & \text{if } SA[i] \neq n-1 \\ SA^{-1}[0] & \text{if } SA[i] = n-1. \end{cases} \tag{5.18}$$

Ψ is possible to be compressed because that it is a partially monotonous increase. The reason for emplying a partial monotonous increase is that, if there are repetitive values, the sorted order can be determined according to the next and subsequent values.

Vertical Code, a code that represents a smaller value in a smaller size, is used for the difference d of Ψ in the compression of Ψ. The difference of Ψ, $d[i]$ is shown as in (5.19). $d[0]$ is undesired, because the sampling Ψ_s of Ψ is stored in order to speed up the retrieval process. If $d[i] \leq 0$, it is always monotonically increasing with the growth of n.

$$d[i] = \begin{cases} \Psi[i] - \Psi[i-1] - 1 & \text{if } i \neq 0 \\ 0 & \text{if } i = 0. \end{cases} \tag{5.19}$$

We divide Ψ into blocks d_k in an interval of M_2, and encode each block d_k by Vertical Code. In other words, we use block $\Psi[0], \cdots, \Psi[n/M_2]$ for

$$\Psi_k = \Psi[M_2 k], \ \cdots, \Psi[M_2(k+1)-1]. \tag{5.20}$$

The head data $\Psi[M_2 k]$ is retained in Ψ_s as a sampling.

Similarly, we divide $d[i]$ into blocks in an interval of M_2. That is, use block d_0, , d_n/M_2 for $d_k = d[M_2 k], \ \cdots, d[M_2(k+1)-1]$. First, we obtain the bit mask $MSB[k]$ which is required in representation of data in the block from the maximum value of $d[i]$. Then, we store the value of the q-th bit of the binary representation of $d[kM_2 + p]$ in the p-th bit of $V_k[q]$. In other words, the size of the V_k (the maximum value of q for each V_k) equals $MSB[k]$. By making M_2 a multiple of 8, $V_k[q]$ can be processed in bytes.

For the block d_k of the difference d, the value of the j-th bit is denoted as $d_k(i,j)$. Similarly, we use $V_k(i,j)$ to represent the j-th bit of V_k. There is a relationship between d_k and V_k shown as follows:

$$V_k(i,j) = d_k(j,i). \tag{5.21}$$

5.1.4 Restoration of the Data Order

In this subsection, we elaborate the restoration of Ψ. The sampling Ψ_s of Ψ is used for the restoration of Ψ. As stated above, Ψ represents the order of data in the sorted sub-fingerprints *SFP*, which can be expressed by Ψ_s and d which is the difference of Ψ.

$$\Psi[i] = \Psi_s[p] + \sum_{k=1}^{q} d_p[k] + q, \tag{5.22}$$

In the above equation,

$$p = i/M, \tag{5.23}$$

$$q = i \bmod M. \tag{5.24}$$

However, in the case of $\Psi[i] > n$, Equation (5.22) becomes

$$\Psi[i] = \Psi[i] \bmod n. \tag{5.25}$$

$\Psi_s[p]$ can be calculated as follows by using d:

$$\Psi_s[p] = \sum_{k=0}^{p-1} SUM[k] + d_p[0] + pM_2. \tag{5.26}$$

The sum of d_p from 1 to q is shown in the second part of (5.22). Since d is represented by the Vertical Code, the sum can be determined by (5.27)

$$\sum_{k=1}^{q} d_p[k] = \sum_{k=0}^{MSB[p]} \{popcount(V_p[k] \& MASK_q) << k\}, \tag{5.27}$$

wherein *popcount*(x) is the number of bits with the value of 1 in the data x [38]. The "|" denotes OR operation, "&" is AND operation and "<<" represents the left shift operation.

$$MASK_q = \{(1 << q)|((1 << q) - 1)\} - 1. \tag{5.28}$$

Through the above process, we can get $\Psi[i]$ from Ψ_s, V and *MSB*V by the following equation.

$$\Psi[i] = \Psi_s[p] + \sum_{k=0}^{MSB[p]} \{popcount(V_p[k] \& MASK_q) << k\} + q. \tag{5.29}$$

The restoration is fast owing to the use of bit operation.

5.2 Search Based on Compressed Suffix Array

The proposed method is based on a compressed suffix array that performs a binary search directly on *SFP*, which is the sorted sub-fingerprints. In fact, the search method based on compressed suffix array is to find the sub-fingerprints block that has the smallest error bit rate with the sub-fingerprints block extracted from the query music.

$$SFP[i],\ SFP[\Psi[i]],\ \cdots,\ SFP[\Psi^j[i]]. \tag{5.30}$$

In (5.30), $p = i/M$ and $\Psi^j[i]$ indicates the repetition of $i = \Psi[i]$ for j times. Figure 5.1 shows the overflow of the whole search based on compressed suffix array.

The process of retrieval method based on compress suffix array can be divided into three stages as follows:

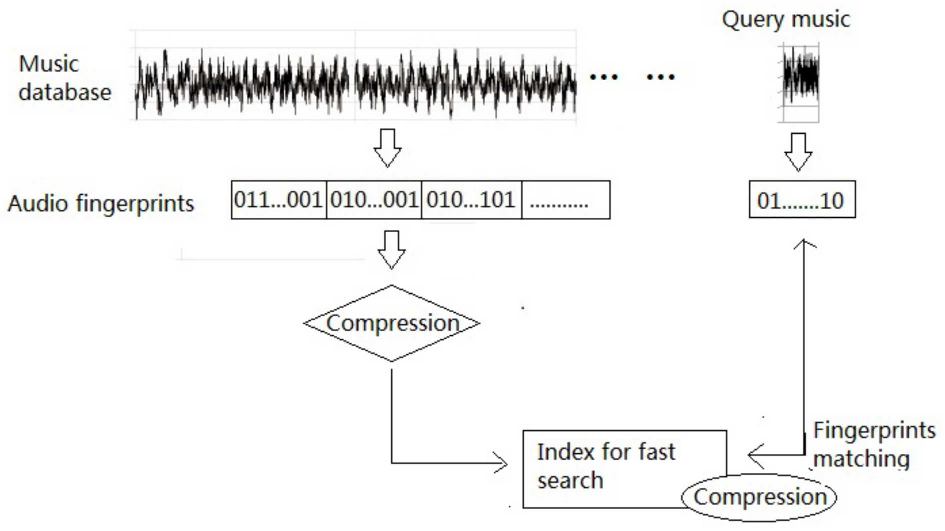

Figure 5.1: Compression of audio fingerprints index

1. For the sub-fingerprints sequence with a length of 3 derived from query Q which is represented by

$$Q_{i,\ i+2} = Q[i],\ Q[i+1],\ Q[i+2], \tag{5.31}$$

we perform a binary search over the sub-fingerprints sequence with a length of 3 derived from the database, which can be expressed as

$$SFP_{j,\ j+2} = SFP[j],\ SFP[\Psi[j]],\ SFP[\Psi^2[j]]. \tag{5.32}$$

As the same, by following the existing methods, we employ the bit error rate to evaluate the similarity.

2. For the $SFP_{j,\ j+2}$ explored in Equation 5.32, we calculate the similarity of the sub-fingerprint blocks. That is, calculate the bit error rate of $Q_{i,\ i+127}$ and $SFP_{j,\ j+127}$, since the length of the block fingerprint is 128. The music data which contains $SFP_{j,\ j+127}$ will be selected as candidates if its bit error rate is below the threshold value.

3. We arrange the candidates in order of the bit error rate, and generate the output music with lower bit error rates as the results.

In contrast, the music retrieval method based on a suffix array performs a binary search, by employing the sub-fingerprints FP extracted from all songs in the database and a suffix array SA of the sorted positions of the sub-fingerprints sequences with a length of 3. Table 5.1 gives the comparison of SA and CSA.

Table 5.1: SA and CSA

	SA			CSA	
i	T	SA	Sorted Suffix	F	Ψ
0	m	10	i	i	4
1	i	7	ippi	i	6
2	s	4	issippi	i	9
3	s	1	ississippi	i	10
4	i	0	mississippi	m	3
5	s	9	pi	p	0
6	s	8	ppi	p	5
7	i	6	sippi	s	1
8	p	3	sissippi	s	2
9	p	5	ssippi	s	7
10	i	2	ssissippi	s	8

As shown as in Figure 5.2, the SA-based method derives the sorted data according to T, and the CSA-based method gets the sorted data by F directly.

In the search, comparisons of the second and subsequent elements are also required. That is, it is necessary to obtain the same order in the results as in the original data. SA-based method derives the sorted data by using T, where

$$T[SA[i]],\ T[SA[i]+1],\ \cdots,\ T[SA[i]+j],\ \cdots., \tag{5.33}$$

while the CSA-based using F and Ψ as shown in (5.32) above.

Figure 5.3 shows the data in the original order. SA-based method directly derives the data in original order by using T.

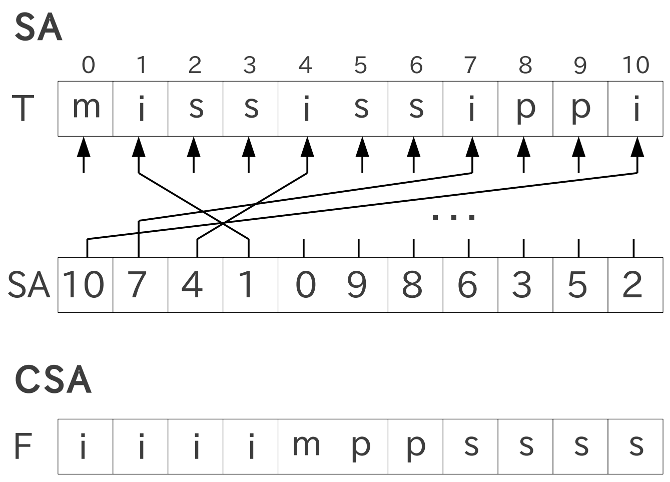

Figure 5.2: The sorted data

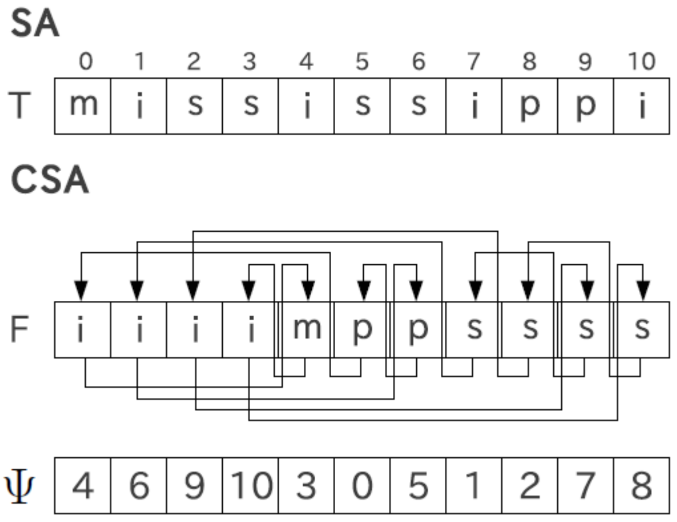

Figure 5.3: Data in original order

5.3 Experiments and Results

We carried the experiments on a database of 8,740 original songs, in order to evaluate the index compression based on the compressed suffix array. Similar to the suffix array-based method, the Haitsma-Kalker algorithm was used for the fingerprint extraction in our experiments in different acoustical analysis settings.

5.3.1 Music Data

In index compression experiments, we selected three sets of music from the database, corresponding to 1,000 songs, 2,000 songs and 4,000 songs respectively. Then the indexes of each set were created, in order to obtain the rate of change on the size and the time.

The fingerprint extraction algorithm in our experiments satisifies: 1) the length of each frame is 1.024 seconds, 2) 32 milliseconds for frame shift, 3) an improved Hamming window; and 4) the length of sub-fingerprints block is 128 instead of 256.

5.3.2 Compression Settings

The segmentation interval M_2 for both the order of data Ψ that have been sorted and the difference d was assigned respectively with the value of 32, that is $M_2 = 32$. This was because the length of the sub-fingerprints was 32. For Vertical Code, the segmentation interval of the block is equivalent to the number of bits of V. That is, according to $M_2 = 32$, V is able to be maintained in 32-bit data structure.

Similarly, we assign the segmentation interval M_1 with the value of 32 for the sorted sub-fingerprints *SFP*, and the same value for the sampling interval M_3 of the *SA*'s sampling SA_s in order to obtain the song number.

5.3.3 Experimental Results

In this subsection, we are to compare the proposed method for index compression based on a compressed suffix array and the suffix array-based method. The experimental results are given in four parts: (1) the size of the sub-fingerprints data, (2) the size of compressed index, (3) the total data size, and (4) the search time.

(1) The size of the sub-fingerprints data

The existing methods always keep the sub-fingerprints *FP* extracted from all the songs of the database directly. The proposed method first sorts the fingerprints *FP* and then encodes the sorted *FP* by RLE. The sizes of sub-fingerprints are shown in Table 5.2 and Figure 5.4.

Table 5.2: Size of sub-fingerprints

songs	*SA*-based method (MB)	Proposed method (MB)	Compression rate (%)
1,000	30.0	13.2	44.0
2,000	58.1	23.3	40.1
4,000	123.4	45.5	36.9
8,000	255.5	84.6	33.1

The compression rate in Table 5.2 was determined by Equation (5.34). The compression rate got higher as the number of songs increased. This is probably because that the Run (a series of repetitive characters) length got longer as the sub-fingerprints increased.

$$\text{Compression rate} = \frac{\text{Data size in proposed method}}{\text{Data size in conventional method}} \times 100\% \qquad (5.34)$$

(2) The size of compressed index

The conventional methods based on suffix array always save the sorted positions *SA* of *FP* directly. The proposed method stores Ψ by Vertical Code in order to obtain the order of data *SFP*. Table 5.3 and Figure 5.5 show the size of *SA* and Ψ.

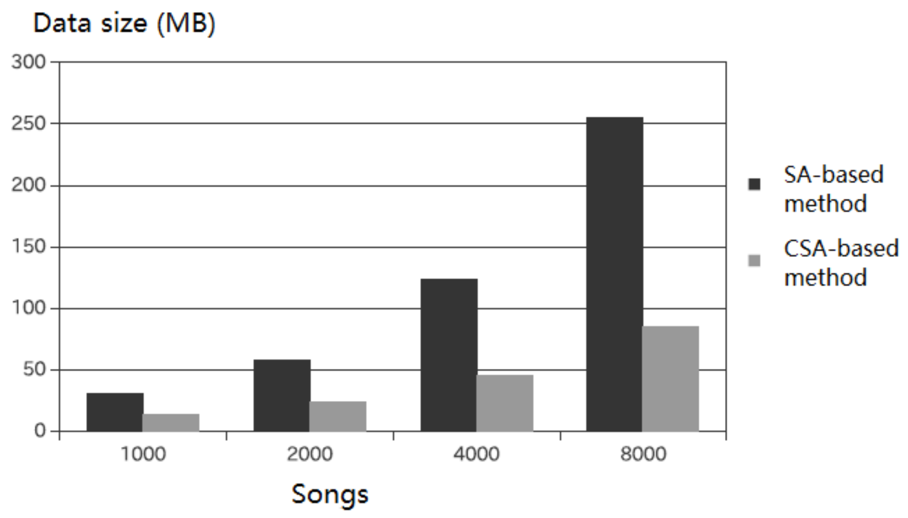

Figure 5.4: Size of sub-fingerprints

Table 5.3: Size of index

songs	*SA*-based method (MB)	Proposed method (MB)	Compression rate (%)
1,000	29.5	23.2	78.6
2,000	57.0	46.3	81.2
4,000	121.3	101.4	83.6
8,000	251.4	214.1	85.2

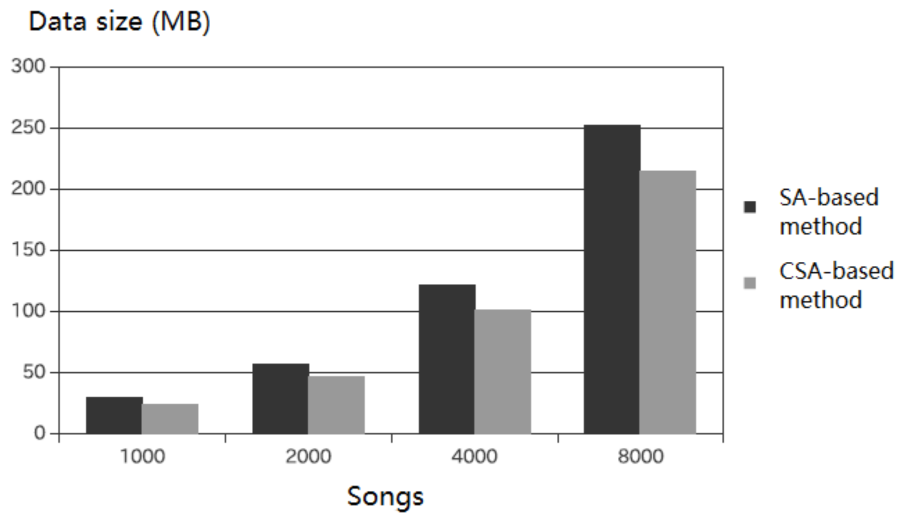

Figure 5.5: Size of index

The size of the data Ψ for SA increased with the number of songs in the database. That is, the compression efficiency became poor with the increase in the size of the database. For $\Psi[i]$, $\Psi[i+1]$ increases when $SFP[i] = SFP[i+1]$. If the types of sub-fingerprint were increased by increasing the number of music, the adjacent data would not necessarily be the same even if the data existed in the sorted data. In other words, the monotonically increasing portion got reduced, which was the cause of the deterioration in compression efficiency.

(3) The total data size

Finally, we illustrate the total data size of sub-fingerprints, SA and Ψ. Table 5.4 and Figure 5.6 show the total data sizes of the conventional method and proposed method. The total data size of the conventional method is the sum of sub-fingerprints and SA. The total data size of the proposed method, besides the sorted FP and Ψ, also took the sampling SA_s^{-1} of SA^{-1} which was necessary for the acquisition of the song number into account. The size of SA_s^{-1} becomes n/M_3+1 when the total number of sub-fingerprints takes n.

Table 5.4: Total data size

songs	*SA*-based method (MB)	Proposed method (MB)	Compression rate (%)
1,000	59.5	37.3	62.7
2,000	115.1	71.4	62.0
4,000	244.7	150.8	61.6
8,000	506.9	306.7	60.5

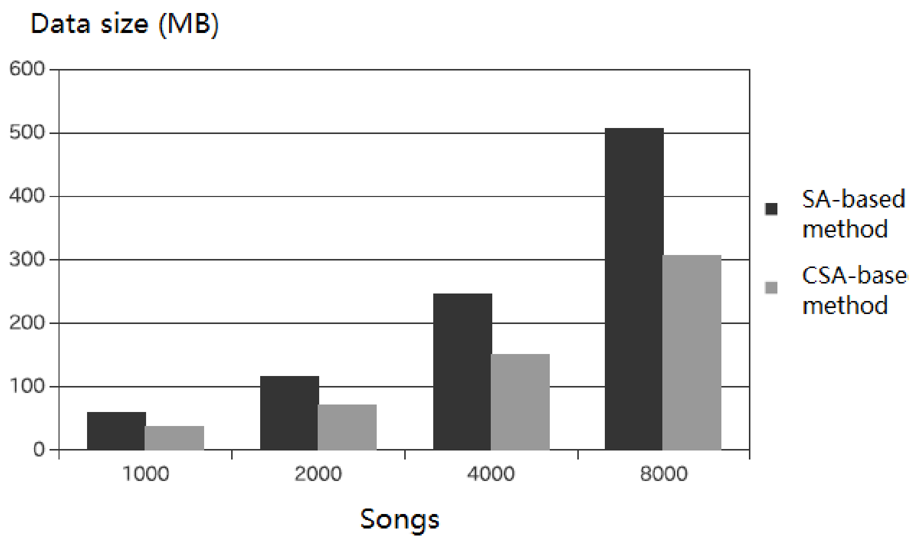

Figure 5.6: Total data size

In general, the compression rate achieved about 60% for each song in the database. In addition, the compression rate got slightly higher with the increase in the number of music. This was due to the height of the compression rate of the sub-fingerprints.

(4) The search time

During the search, the songs used as queries were the same as the database, and had the same length as the original songs. We used the results of a query per 10 seconds, because a query should last seven seconds at least. That is, for a query of S_q seconds, the search time per 10 seconds could be denoted by

$$\frac{S_s}{S_q} \times 10 \text{ [seconds]} \tag{5.35}$$

if the search took S_s seconds.

Table 5.5 shows the average search time for each song in all sets of music data. Slow-down factor (SLF) indicates the multiple slices of time taken by the proposed method, compared with the conventional method. That is, the smaller SLF is, the faster the proposed method. SLF was determined by (5.36).

$$\text{SLF} = \frac{\text{Search time in proposed method}}{\text{Search time in conventional method}} \tag{5.36}$$

Table 5.5: Search time

Songs	SA-based method (s)	Proposed method (s)	SLF
1,000	0.001	0.012	12.0
2,000	0.001	0.012	12.0
4,000	0.002	0.016	8.0
8,000	0.003	0.017	5.7

Table 5.5 indicates that the proposed method took more time for search. However, the slow-down factor tended to decrease with the increase of songs in database.

Chapter 6

Conclusions and Future Work

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

The robustness of the fast Hamming space search method based on query multiplexing against the real noise has been examined through the expriments. The experiments indicated that frequency bands of human speech correspond to a greater degree to the frequency bands used as characteristics in audio fingerprinting. Therefore, more noise in these frequencies has a greater effect on the retrieval performance. As a result, the retrieval accuracy under noise from machanic noise remained higher than other noise, even if the original music could not be heard clearly. By contrast, the accuracy decreased under noise from human beings.

We also presented a method of index compression using a compressed suffix array in order to reduce the space. The experimental results shows that this method can save much space. Our method took more time for search, which maybe due to the data structure. However, the multiples of search time tended to decrease with the increase of songs in database.

Our future work will focus on the use of a large-scale database in order to enhance the retrieval speed. The flexible application of the music database for music to be searched will be also considered in the future.

Acknowledgments

I would like to express my sincere gratitude to all those who helped me during my Ph.D study.

First of all, I would like to express my heartfelt gratitude to Prof. Kenji Kita, my supervisor, for his professional guidance and his patience in supervisions. I am deeply grateful for his consistent encouragement and illuminating instruction on my Ph.D study and research.

Also, I would like to thank Prof. Masami Shishibori, Prof. Fuji Ren, Prof. Motoyuki Suzuki, and Dr. Kazuyuki Matsumoto, who have instucted and helped me a lot in the past three years.

Finally my sincere gratitute would go to my friends and the members of A2 group, especally Narumi Saito, Yusaku Daito and Masashi Onishi, who helped me work out my problems during the difficult course of the thesis.

References

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

[2] Shazam: http://www.shazam.com/.

[3] Gracenote: http://www.gracenote.com/.

[4] A. 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] M. L. Miller, M. Acevedo. Rodriguez, and I. 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] D. Fragoulis, G. Rousopoulos, T. Panagopoulos, C. Alexiou, and C. Papaodysseus. On the Automated Recognition of Seriously Distorted Musical Recordings. *IEEE Transactions on Signal Processing*, Vol.49, No.4, pp.898–908, 2001.

[7] B. Logan. Mel Frequency Cepstral Coefficients for Music Modeling. *Proceedings of the International Symposium on Music Information Retrieval* (ISMIR 2000), 2000.

[8] E. Allamanche. AudioID: Towards Content-based Identification of Audio Material. *110th AES Convention*, pp.5380, 2001. http://www.aes.org/e-lib/browse.cfm?elib=10019.

[9] S. Nakamura. Beginners Digital Fourier Transform. Tokyo Denki University Press, 1989.

[10] L. D. Enochson and R. K. Otnes. *Programming and Analysis for Digital Time Series Data*, pp.142, 1968.

[11] H. Hermansky. Perceptual Linearpredictive (PLP) Analysis Speech. *Journal of the Acostic Society of America*, Vol.87, No.4, 1990.

[12] Q. Xiao, Y. Daito, K. Matsumoto, M. Suzuki, and K. Kita. Fast Search Method for Audio Fingerprinting Systems Based on Query Multiplexing. *IEEJ Transactions on Electronics, Information and Systems*, Vol.132, No.9, pp.1481-1487, 2012.

[13] Q. Xiao, M. Suzuki, and K. Kita. Fast Hamming Space Search for Audio Fingerprinting Systems. *The 12th International Society for Music Information Retrieval Conference*, pp.133–138, 2011.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. *25th International Conference on Very Large Data Bases (VLDB 1999)* , pp.518–529, 1999.

[15] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise Independent Permutations. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp.327–336, 1998.

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

[17] M. Datar, N. Immorlica, P. Indyk, and V. S.Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. *Proceedings of the 20th Annual Symposium on Computational Geometry*, pp.253–262, 2004.

[18] B. Kulis and K. Grauman. Kernelized Locality-Sensitive Hashing for Scalable Image Search. *Proceedings of the 12th IEEE International Conference on Computer Vision (ICCV 2009)*, 2009.

[19] P. Indyk and R. 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.

[20] D. Ravichandran, P. Pantel, and E. Hovy. Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp.622–629, 2005.

[21] G. S. Manku, A. Jain, and A. D. Sarma. Detecting Near-Duplicates for Web Crawling. *Proceedings of the 16th international conference on World Wide Web*, pp.141–149, 2007.

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

[23] T. Yamada, T. Nakajima, N. Kitawaki, and S. Makino. Performance Estimation of Noisy Speech Recognition Considering Recognition Task Complexity. *11th Annual Conference of the International Speech Communication Association*, pp.2042-2045, 2010.

[24] C. Bandera, A. M. Barbancho, L. J. Tardn, S. Sammartino, and I. Barbancho. Humming Method for Content-based Music Information Retrieval. *12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, pp.49-54, 2011.

[25] V. Chandrasekhar, M. Sharifi, and D. A. Ross. Survey and Evaluation of Fingerprinting Schemes for Mobile Query-by-Example Application. *12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, pp.801-806, 2011.

[26] B. Niedermayer, S. Bck, and G. Widmer. On the Importance of Real Audio Data for MIR Algorithm Evaluation at the Note-Level - A Comparative Study. *12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, pp.543-548, 2011.

[27] W. Yoon and K. Park. A Noise Robust Content-based Music Retrieval System. *IEEE international conference on consumer electronics*, pp.1-2, 2009.

[28] F. Balado, N. J. Hurley, E. P. McCarthy, and G. C. M. Silvestre. Performance of Philips Audio Fingerprinting under Additive Noise. *32nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp.209-212, 2007.

[29] P. J. O. Doets and R. L. Lagendijk. Extracting Quality Parameters for Compressed Audio from Fingerprints. *6th International Conference on Music Information Retrieval (ISMIR 2005)*, pp.498-503, 2005.

[30] M. Park, H. Kim, and S. H. Yang. Frequency-Temporal Filtering for a Robust Audio Fingerprinting Scheme in Real-Noise Environments. *ETRI Journal*, Vol.28, No.4, pp.509-512, 2006.

[31] M. Park, H. Kim, Y. M. RO, and M. Kim. Frequency Filtering for a Highly Robust Audio Fingerprinting Scheme in a Real-Noise Environment, *IEICE Trans. Inf. Syst.*, Vol.E89-D, NO.7, pp.2324-2327, 2006.

[32] Q. Xiao, Y. Daito, M. Suzuki, and K. Kita. Performance Evaluation of Audio Fingerprinting Systems in Real-Noise Conditions. *The 2012 IET International*

Conference on Frontier Computing - Theory, Technologies and Applications, pp.61–66, 2012.

[33] T. Kurita. Development of External-Noise Reduction Technologies for Shinkansen High-Speed Trains. *Journal of Environment and Engineering*, Vol.6, No.4, pp.805-819, 2011.

[34] R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *32nd ACM symposium on Theory of computing*, 2000.

[35] D. Okanohara and K. Sadakane. Practical Entropy-Compressed Rank Select Dictionary. *Algorithm Engineering and Experiments (ALENEX 2007)*, 2007.

[36] Q. Xiao, N. Saito, K. Matsumoto, X. Luo, Y. Yokota, and K. Kita. Index Compression for Audio Fingerprinting Systems Based on Compressed Suffix Array. *International Journal of Information and Education Technology*, Vol.3, No.4, pp.455-460, 2013.

[37] N. Saito. Index Compression for Audio Fingerprinting Systems. *Tokushima University Master Thesis*, 2012.

[38] R. Gonzalez, S. Grabowski, V. Makinen, and G. Navarro. Practical Implementation of Rank and Select Queries. *4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, pp.27-38, 2005.