Index Compression for Audio Fingerprinting Systems Based on Compressed Suffix Array

Qingmei Xiao, Narumi Saito, Kazuyuki Matsumoto, Xin Luo, Yasushi Yokota, and Kenji Kita

Abstract—**As one of most popular technologies, audio fingerprinting has recently attracted much attention in music retrieval systems. In music retrieval methods based on audio fingerprints, a large database is required in order to compare the fingerprints extracted from the query. In other words, the efficient search method has to be developed. In this paper, we propose a method for index compression using a compressed suffix array. Taking advantage of the fact that the repetitive characters occur frequently in higher bits of the sorted audio fingerprint data, the proposed method compresses the index by encoding the 8-bit data sequences by Run Length Encoding. Vertical Code is also used to compress the array, wherein the positions of the sorted data are stored. Four sets of music databases are used in experiments to evaluate the effectiveness of the proposed method. The experimental results show that the proposed method, compared with the conventional method, only needs 30% of the space of an audio fingerprints database for a music database consisting of 8000 songs, and around 80% of the index space for a database of 1000 songs. Moreover, the entire space cost is reduced to around 60%, compared with the method based on the suffix array.**

Index Terms—**Audio fingerprint, compressed suffix array, index compression, run length encoding, vertical code.**

I. Introduction

A music retrieval system enables users to easily obtain information about an unknown song such as song name, artist and album [1], [2]. Most music retrieval systems adopt audio fingerprinting proposed by Haitsma and Kalker [3]. The audio fingerprint is a feature used to identify the song. Information for an unknown music clip can be derived by using audio fingerprints together with a music information database. Moreover, audio fingerprints are not only used to retrieve music, but also for copyright protection of music, such as detecting the distribution of copyright-infringing songs on the Internet.

A music query can start at any time from the original song, which requires a large search space. Thus a fast and efficient retrieval method is demanded. A few works have been developed on the audio fingerprinting searching method, including a method using a hash table [3], [4] and a tree-structured representation of fingerprints [5]. The method using a suffix array [6] has also been proposed. In the method based on the suffix array, the space cost increases in proportion to the growing music database. In this paper, we proposed a method to reduce the space cost by compressing the index of the database.

The paper is organized as follows: Section II outlines music retrieval based on audio fingerprints. We review a fast Hamming space search method [7] based on a suffix array in Section III, and propose a space-saving method based on a compressed suffix array in Section IV. We evaluate the proposed method in Section V. Finally, the conclusions and future work are given in Section VI.

Manuscript received March 9, 2013; revised June 24, 2013.

Qingmei Xiao, Kazuyuki Matsumoto and Kenji Kita are with the Department of Information Science and Intelligent Systems, the University of Tokushima, Tokushima, Japan (e-mail: hanmay510122@gmail.com, matumoto@is.tokushima-u.ac.jp, kita@is.tokushima-u.ac.jp).

Narumi Saito is with OPTPIA Co., Ltd., Tokushima, Japan (e-mail: saito-narumi@iss.tokushima-u.ac.jp).

Xin Luo is with the School of Computer Science and Technology, Donghua Universtiy, Shanghai, (e-mail: China rashin.lx@gmail.com).

Yasushi Yokota is with Doi Hospital, Emihe, Japan (e-mail: yokota@is.tokushima-u.ac.jp).

II. Music Retrieval Based on Audio Fingerprints

As a kind of message digest (one-way hash function), audio fingerprinting converts an audio signal into a relatively compact representation by using acoustical and perceptual characteristics of the audio signals. For the message digesting methods mainly used for authentication and digital signatures (e.g., MD5), slight difference in the original objects will turn to completely different hash values. In other words, two hash values mapped from an original audio signal and a corrupted one respectively are totally different, which is the reason why the retrieval performance for "corrupted" queries decreases drastically. 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.

A. Audio Fingerprints Extraction

Audio fingerprint extraction employs the signal characteristics of the music data. One of the most popular audio fingerprint extraction algorithms proposed by Haitsma and Kalker [3] uses the sign of energy difference between frequency bands as a feature.

The audio fingerprint extraction algorithm in [3] performs as follows. First, segment the input music into overlapping frames for each 0.37 seconds, and then divide each frame into 33 non-overlapping frequency bands. Next, extract 32-bit sub-fingerprint by determining the 32 signs of energy difference between two successive frequency bands of each frame.

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:

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

DOI: 10.7763/IJIET.2013.V3.317

where

$$ED(n,m) = E(n,m) - E(n,m+1) - \big(E(n-1,m) - En{-}1,\, m{+}1 \qquad (2)$$

Haitsma and Kalker has demonstrated that the sign of energy difference between 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.

B. Distance between Audio Fingerprints

The sub-fingerprint is a 32-bit feature extracted from a frame of an input music, and the information contained in one sub-fingerprint is not enough to identify the audio. Thus, a fingerprint block, which is a sequence of sub-fingerprints and contains sufficient information, 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. For the sub-fingerprints extracted from audio clips A and B, let $FA(n, m)$, $FB(n, m)$ be the m-th bit of frame n respectively. The bit error rate of fingerprint block $BER(A, B)$ of length N is formally defined as:

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

$BER(A, B)$ is the error rate per bit. The operator ^ denotes bitwise operation XOR (exclusive or). The numerator in (3) calculates the Hamming distance between two fingerprint blocks.

C. Audio Fingerprints Searching

Audio music retrieval methods based on audio fingerprinting usually follows this process below. First, extract fingerprint blocks from query music and each song in the database respectively. For each song in database and query music, quite a number of fingerprint blocks with different starting positions of frame can be extracted. And then music retrieval becomes a problem to find the fingerprint block in the database whose error bit rate with the fingerprint block derived from the query is the smallest.

One problem of the method based on audio fingerprinting is the quite huge searching space [8]-[11]. For example, distance calculations for a fingerprint database containing 10,000 songs would be several to several dozen times as large as 250 million by brute-force search taking account of matching the fingerprint blocks. Many methods have been developed over the past few years to reduce the number of calculations [12]-[15]. However, these methods encounter a drawback of rapid growing size of hash table as the bit error rates between the query and songs in the database increases.

III. Fast Humming Space Search Based on Suffix Array

Reference [7] has given a fast Hamming space search method for audio fingerprinting systems. The fast Hamming space searching holds the index to the sub-fingerprints sequence that is similar to a suffix array in order to keep high-speed searching in sub-fingerprints database.

The Hamming space search method reduces the searching space greatly by multiplexing the queries of the sub-fingerprint sequence instead of expanding the database. Sub-fingerprints are extracted by shifting the query into frames. Multiplexed sub-fingerprints with slight differences used for query multiplexing can be obtained as the starting position of a frame varies with time due to the great similarity between the overlapping sub-fingerprints in the sequence of sub-fingerprints.

Usually, one sub-fingerprint does not contain sufficient information for music identification, so a sequence of sub-fingerprints (SSF) is employed for matching. Suppose $FP = (FP_1, FP_2, \ldots, FP_n)$ be the sub-fingerprints obtained from all the songs in a database, many m-length SSFs (m is set to be 3 in [5]) can be derived by changing the starting position of the fingerprint, and the i-th sub-fingerprint sequence is defined as $SSF_i = (FP_i, FP_{i+1}, \ldots, FP_{i+m-1})$. Then, all the SSFs are sorted by value and their positions are stored in a suffix array $SA = SA_1, SA_2, \ldots, SA_{n-m+1}$. Array SA contains the indexes to FP, and satisfies the following:

$$SA_j = \text{i iff } SSF_i = (FP_i, FP_{i+1}, \ldots, FP_{i+m-1}) \qquad (4)$$

is the j-th SSF in sorted order.

The search process, as shown in Fig. 1, 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 locate 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.

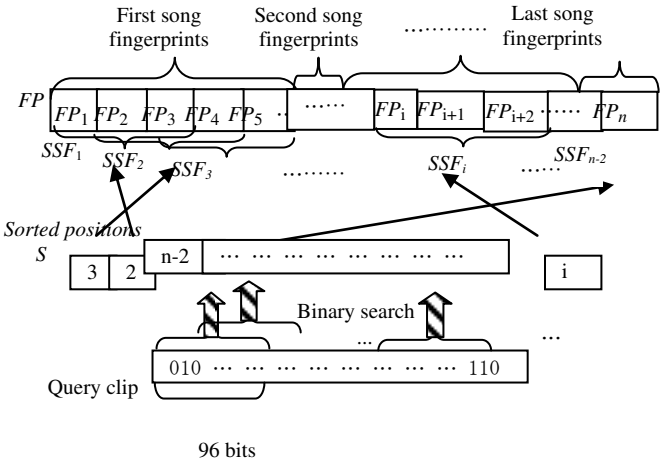

Fig. 1. Overview of scheme of SSF search.

In the binary search on array S, most similar SSFs can be found by checking the neighborhood 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 that required by conventional methods such as the method based on random permutations [16].

IV. Music Retrieval Based on Compressed Suffix Array

A. Index Compression

We use a compressed suffix array for index compression because the index has the same structure as the suffix array [14]. The proposed method first compresses sub-fingerprints extracted from the database in sorted order, and keeps the compressed data. That is, use $FP'= FP'[0],\ldots, FP'[n]$ ($FP'[i] = FP'[SA[i]]$) to replace FP. And then segment FP' into pieces in an interval of M_1. In other words, $FP'= FP'_0,\ldots, FP'_{n/M1}$ can be derived from $FP'_k= FP'[M_1\times k]; FP'[M_1\times k+1],\ldots, FP'[M_1\times(k+1)-1]$—the block of FP. This is because the required restoration is performed instantaneously during searching.

In addition, for each segmented block FP'_k, suppose each sub-fingerprint (32 bits) is in unit of 8 bits, and we perform Run-Length Encoding (RLE) for 8-bit bytes data sequences [17]-[19]. RLE is an encoding technique wherein a series of repetitive data symbols are compressed into a shorter code, which indicates the length of a code and the data being repeated. The 8-bit sequence is arranged because the values of bits in the upper side are likely to be the same since FP' is the sorted.

For example, given that the segmentation interval M_1 ($M_1 = 8$) of sub-fingerprint FP having been sorted, the FP'_0 can be expressed as shown in (5).

$$FP'_0= \begin{matrix} 00 & 00 & 00 & 00, \\ 00 & 00 & 00 & 00, \\ 00 & 00 & 00 & 01, \\ 00 & 00 & 00 & 01, \\ 00 & 00 & 01 & 01, \\ 00 & 00 & 01 & 01, \\ 00 & 00 & 11 & 01, \\ 00 & 00 & 11 & 01. \end{matrix} \quad (5)$$

Fetch only values of the highest bits from (5) and we can get

$$FP'_0(0, 3), FP'_0(1, 3),\ldots, FP'_0(7, 3) =00, 00, 00, 00, 00, 00, 00, 00. \quad (6)$$

which becomes (7) after performing Run Length Encoding since the repetitive characters of "00" occurs 8 times.

$$00, 00, 08. \quad (7)$$

Similarly, the next bit is shown as in (8) and the same encoded data as in (7) by Run Length Encoding.

$$FP'_0(0, 2), FP'_0(1, 2),\ldots, FP'_0(7, 2) =00, 00, 00, 00, 00, 00, 00, 00. \quad (8)$$

The other two bits are shown in (9) and (12), and their encoded data as in (10) and (12) respectively by Run Length Encoding.

$$FP'_0(0, 1), FP'_0(1, 1),\ldots, FP'_0(7, 1) =00, 00, 00, 00, 01, 01, 11, 11. \quad (9)$$

$$00, 00, 04, 01, 01, 02, 11, 11, 02. \quad (10)$$

$$FP'_0(0, 0), FP'_0(1, 0),\ldots, FP'_0(7, 0) =00, 00, 01, 01, 01, 01, 01, 11. \quad (11)$$

$$00, 00, 02, 01, 01, 05, 11. \quad (12)$$

Subsequently, consider (7), (10), and (12) as 8-bit data sequences, and they can be expressed as (13) after being packed to 32-bit data. Additionally, FP'_0 becomes (13) by Run Length Encoding. The bigger the size of database, the longer the Run (a series of repetitive characters) Length will be and subsequently the compression efficiency increases. $FP'= FP'_0, \ldots, FP'_{n/M1}$ is retained as the sub-fingerprints of the database.

$$FP''_0= \begin{matrix} 00, & 00, & 08, & 00, \\ 00, & 08, & 00, & 00, \\ 04, & 01, & 01, & 02, \\ 11, & 11, & 02, & 00, \\ 00, & 02, & 01, & 01, \\ 05, & 11. & & \end{matrix} \quad (13)$$

Since the order of original database has been lost, we use $\Psi[i]$, as shown in (14) to represent the order.

$$\Psi[i] = \begin{cases} SA^{-1}[SA[i]+1] & \text{if } SA[i] \neq n \\ 0 & \text{if } SA[i] = n \end{cases} \quad (14)$$

Using $\Psi[i]$, the sequence the same as the database represented by $FP[SA[i]], FP[SA[i]+1], \ldots, FP[SA[i]+j]$ is replaced by $FP'[i], FP'[\Psi[i]], \ldots, FP'[\Psi^j[i]]$ wherein $[\Psi^j[i]]$ indicates the j-th repetitive characters of $i= \Psi[i]$. $\Psi[i]$ is possible to be compressed since it is a partially monotonous increase. The reason for a partial monotonous increase is that, if there are repetitive values, the sorted order is 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 of Ψ during compression of $\Psi[i]$. As the difference of $\Psi[i]$, $d[i]$ is shown as in (15). If $d[i] < 0$, it is always monotonically increasing with the growth of n.

$$d[i] = \begin{cases} \Psi[i]-\Psi[i-1]-1 & \text{if } i \neq 0 \\ 0 & \text{if } i = 0 \end{cases} \quad (15)$$

Divide $\Psi[i]$ into blocks in an interval of M_2. In other words, use block $\Psi_0, \ldots, \Psi_{n/M2}$ for $\Psi_k = \Psi[M_2\times k], \ldots, \Psi[M_2\times(k+1)-1]$. The first block data $\Psi[M_2\times k]$ is stored in Ψ'_k as a sampling. This is because the restore is required, as well as FP', to be performed instantly when searching.

Similarly, divide $d[i]$ into blocks in an interval of M_2. That is, use block $d_0, \ldots, d_{n/M2}$ for $d_k = d[M_2\times k], \ldots, d[M_2\times(k+1)-1]$. First, obtain the bit mask $MSB[k]$ which is required in representation of data in the block from the maximum value of $d[i]$. Moreover, store the value of q-th bit of the binary representation of $d[M_2\times k+p]$ 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 a multiple of 8 M_2, $V_k[q]$ can be processed in bytes.

For example, suppose $M_2 = 8$ and d_0 in decimal is $d_0 = 1, 0, 1, 2, 3, 2, 1, 0$. All the values can be expressed in 2-bit data since the maximum value of d_0 is 3. Convert d_0 to a 2-bit

binary number d_0 = 01, 00, 01, 10, 11, 10, 01, 00, and then we can derive $V_0[0]$ = 0101 0101, $V_0[1]$= 0011 1000 for V_0= $V_0[0]$, $V_0[1]$.

Instead of $\Psi[i]$, indexes can be compressed by maintaining the $V = V_1, \ldots, V_{n/M2}$ and $MSB = MSB[1], \ldots, MSB[n/M_2]$, which are obtained through the above process.

B. Index Restoration

We elaborate the restoration of the index Ψ. The restoration is performed by using Ψ' (a sampling of Ψ). As stated above, Ψ which represents the order of data in the sorted sub-fingerprints *SFP*, can be expressed by Ψ' and the difference *d* as shown in (16).

$$\Psi[i] = \Psi'[i'] + j' + \sum_{k}^{j'} d[M2 \times i' + k] \qquad (16)$$

In (16), $i' = i/M_2$, $j' = i \bmod M_2$. In addition, $\Psi[i] = \Psi[i] \bmod n$ in the case of $\Psi[i] > n$. As the sum of $d[i']$ from 1 to j', $\sum_{k=1}^{j'} d[M2 \times i' + k]$ uses $MASK = \{(1<<j') \mid ((1<<j')-1) -1\}$, and it turns into $\sum_{k=0}^{MSB[i']}\{\text{popcount}(V'[k]\&MASK) << k\}$ wherein *MASK* indicates the MASK from 1 to j' and popcount(x) indicates the number of bit valued 1 in the data x. The "|" denotes OR operation, "&" denotes AND operation and "<<" represents left shift operation. We can get $\Psi[i]$ from Ψ, V and MSB through the above process.

C. Search Based on Compressed Suffix Array

The music retrieval method based on a suffix array performs a binary search by using sub-fingerprints *FP* extracted from all songs in the database and a suffix array *SA* wherein the sorted positions of the sub-fingerprints sequences with a length of 3 are stored. The proposed method is based on a compressed suffix array that performs a binary search directly on *SFP*—the *FP* which have been sorted in advance.

The search method based on compressed suffix array is to find, from the $FP'[i], \ldots, FP'[\Psi^j[i]]$, the sub-fingerprints block that has the smallest error bit rate with the sub-fingerprints block extracted from the query music. The process can be divided into three stages as follows

- For the sub-fingerprints sequence with a length of 3 derived from query Q as in (17), perform a binary search on the sub-fingerprints sequence with a length of 3 derived from the database as in (18). The same as used in the existing methods, the bit-error is used to evaluate the similarity.

$$Q_{i,\ i+2} = Q[i], Q[i+1], Q[i+2] \qquad (17)$$

$$SFP_{j,\ j+2} = SFP[j], SFP[\Psi[j]], SFP[\Psi^2[j]] \qquad (18)$$

- For $SFP_{j,\ j+2}$ explored above, calculate the degree of similarity in sub-blocks fingerprint. That is, calculate 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 that contain $SFP_{j,\ j+127}$ will be selected as candidates if the bit error rate is below the threshold value.
- Arrange the candidates in order of the bit error rate, and then output the music with lower bit error rates as the results.

V. Exprients and Results

We carried the experiments to evaluate both the method based on the suffix array and that on the compressed suffix array. The music database and audio fingerprints used in the experiments are the same for both methods.

Just like the method based on suffix array, 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.

A. Experimental Conditions

Music data The database contained 8, 000 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. In our experiments, we selected three sets of music from the database, corresponding to 1000 songs, 2000 songs and 4000 songs. Then the indexes of each set were created, in order to obtain the rate of change on the size and the time.

Compression setting The segmentation interval M_2 for both the order of data Ψ that have been sorted and the difference *d* was assigned a value of 32, that is M_2 =32. This was because the length of the sub-fingerprints was 32. Assign the segmentation interval M_1 a value of 32 similarly for sorted sub-fingerprints SF, and the same value for the sampling interval M_3 of the SA's sampling SA_S in order to obtain the song number.

B. Data Size

Subsequently, we are to compare the proposed method for index compression based on a compressed suffix array with that on a suffix array. The size of the index data and the size of the sub-fingerprint are to be described separately.

Size of sub-fingerprint The proposed method first sorted the fingerprints *FP* and then encoded the sorted *FP* by RLE. The size of sub-fingerprint is shown in Table I.

TABLE I: Size of Sub-Fingerprints

Songs	Conventional method (MB)	Proposed method (MB)	Compression rate (%)
1000	30.0	13.2	44.0
2000	58.1	23.3	40.1
4000	123.4	45.5	36.9
8000	255.5	84.6	33.1

The compression rate in Table I was determined by (19). 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.

$$\text{Compression rate} = \frac{\text{Data size in proposed method}}{\text{Data size in conventional method)}} \times 100\% \qquad (19)$$

Size of indexes The proposed method stored Ψ by Vertical Code in order to obtain the order of data *SFP*. Table II shows the size of *SA* and Ψ. The size of the data Ψ for SA would increase with the increase in the number of songs in the database. That is, the compression efficiency became poor

with the increase in the database. If the types of sub-fingerprint were increased by increasing the number of music, the adjacent data would not necessarily be the same even it existed in the sorted data. In other words, the monotonically increasing portion reduced, which was the cause of the deterioration in compression efficiency.

TABLE II: SIZE OF INDEXES

Songs	Conventional method (MB)	Proposed method (MB)	Compression rate (%)
1000	29.5	23.2	78.6
2000	57.0	46.3	81.2
4000	121.3	101.4	83.6
8000	251.4	214.1	85.2

Total data size The total data size is shown in Table III. The compression rate achieved, in general, 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 is due to the height of the compression rate of the sub-fingerprint.

TABLE III: TOTAL DATA SIZE

Songs	Conventional method (MB)	Proposed method (MB)	Compression rate (%)
1000	59.5	37.3	62.7
2000	115.1	71.4	62.0
4000	244.7	150.8	61.6
8000	506.9	306.7	60.5

C. Search Time

During the search, the songs used as queries are the same as the database, and had the same length as the original songs. In this section, we use the results of a query per 10 seconds. That is, for a query of S_q seconds, the search time per 10 seconds can be denoted by S_s /S_q×10 music for [s] if the search takes S_s seconds.

Table IV shows the average search time for each song in all sets of music data. Slow-down factor (SLF) indicates the multiple of the 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 (20).

$$\text{SLF} = \frac{\text{search time in proposed method}}{\text{search time in conventional method)}} \quad (20)$$

TABLE IV: AVERAGE SEARCH TIME

Songs	Previous method (s)	Proposed method (s)	SLF
1000	0.001	0.012	12.0
2000	0.001	0.012	12.0
4000	0.002	0.016	8.0
8000	0.003	0.017	5.7

Table IV also shows that the proposed method took more time for search, which may be due to the data structure. However, the Slow-down factor tended to decrease with the increase of songs in database.

VI. CONCLUSIONS

In this paper, we have presented a method of index compression using a compressed suffix array. The experimental results show that this method can save much space. Our method took more time for search. 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 application of the music database for music to be searched will be also considered in the future

REFERENCES

[1] Shazam. [Online]. Available: http://www.shazam.com/.

[2] Gracenote. [Online]. Available: http://www.gracenote.com/.

[3] J. Haitsma and T. Kalker, "Highly robust audio fingerprinting system," in *Proc. 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, 2002, pp.107-115.

[4] A. L. Wang, "An Industrial-Strength Audio Search Algorithm," in *Proc. the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, 2003, pp. 7-13.

[5] M. Miller, M. Rodriguez, and I. Cox, "Audio fingerprinting: Nearest neighbour search in high dimensional binary spaces," *Journal of VLSI Signal Processing*, vol. 41, no. 3, pp.285-291, 2005.

[6] U. Manber and G. Myers, "Suffix arrays: a new method for on-line string searches," *1st ACM-SIAM symposium on Discrete algorithms*, 1990.

[7] Q. Xiao, M. Suzuki, and K. Kita, "Fast Hamming space search for audio fingerprinting systems," in *Proc. 12th International Society for Music Information Retrieval Conference (SIMIR 2011)*, 2011, pp.133-138.

[8] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp.604–613.

[9] 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.

[10] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. the 34th Annual ACM Symposium on Theory of Computing*, 2002, pp. 380–388.

[11] A. Wang, "An industrial-strength audio search algorithm," in *Proc. 4th* International *Conference on Music Information Retrieval (ISMIR 2003)*, 2003, pp. 7-13.

[12] R. Gonzalez, S. Grabowski, V. Makinen, and G. Navarro, "Practical implementation of rank and select queries," in *Proc. 4th International Workshop on Experimental and E_cient Algorithms (WEA 2005)*, 2005, pp.27-38.

[13] C. Bandera, A. M. Barbancho, L. J. Tardón, S. Sammartino, and I. Barbancho, "Humming method for content-based music information retrieval," in *Proc. 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011, pp. 49-54.

[14] P. J. O. Doets and R. L. Lagendijk, "Extracting quality parameters for compressed audio from fingerprints," in *Proc. 6th International Conference on* Music *Information Retrieval (ISMIR 2005)*, 2005, pp.498-503.

[15] V. Chandrasekhar, M. Sharifi, and D. A. Ross, "Survey and evaluation of fingerprinting schemes for mobile query-by-example application," in *Proc. 12th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011, pp.801-806.

[16] 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.

[17] H. Hermansky, "Perceptual linearpredictive (PLP) analysis speech," *Journal of the Acostic Society of America*, vol. 87, no. 4, 1990.

[18] E. L. Hauck, "Data compression using run length encoding and statistical encoding," U.S. Patent 4 626 829, Dec. 2, 1986.

[19] D. Okanohara and K. Sadakane, "Practical Entropy-Compressed Rank Select Dictionary," *Algorithm Engineering and Ex-periments (ALENEX 2007)*, 2007.

Qingmei Xiao was born in Fujian, China, in 1983. She received the B.S. degree in information management & information systems from Dalian Polytechnic University, China, in 2006, and earned a Master degree in Computer Science & Engineering from Dalian University of Technology, China, in 2009. She is currently a Ph. D student studying on music information retrieval system in the Department of Information Science and Intelligent Systems.

Narumi Saito was born in Tokushima, Japan, in 1989. She received the B.S. degree and the Master degree, both in Intelligent Information Engineering, from Faculty of Engineering, the University of Tokushima, Japan, in 2011 and 2013, respectively. She is currently working in OPTPIA Co., Ltd., Japan.

Kazuyuki Matsumoto was born in Tokushima, Japan, in 1980. He received the B.S. degree in Intelligent Information in March 2003 (a Master degree in March 2005 and a Ph. D degree in March 2008) from Faculty of Engineering, the University of Tokushima, Japan. He is currently an assistant professor in Department of Information Science and Intelligent Systems, the University of Tokushima. His research interests include Affective Computing, Emotion Recognition and Natural Language Processing. Mr. Matsumoto is a member of IPSJ, IEICE and NLP.

Xin Luo was born in Hunan, China, in 1972. He received the M.E. and D.E. degrees from the Faculty of Engineering, University of Tokushima, Tokushima, Japan, in 2004 and 2007, respectively. Since 2007, he has been with the School of Computer Science and Technology, Donghua University, Shanghai, China. His current research interests include multimedia information retrieval and pattern recognition.

Yasushi Yokota was born in Tokyo, Japan, in 1956. From 1977 to 1981, he attended Waseda University, Tokyo, Japan, and left this university without a diploma. He graduated from School of Medicine, Akita University, Japan, in 1987. Mr. Yokota is currently an interventional cardiologist superintendent of Doi Hospital, Emihe, Japan. His current research interests include interventional cardiology and medical information retrieval.

Kenji Kita was born in Oita, Japan, in 1957. He received the B.S. degree in mathematics and the Ph. D degree in electrical engineering, both from Waseda University, Tokyo, Japan, in 1981 and 1992, respectively. From 1983 to 1987, he worked for the Oki Electric Industry Co. Ltd., Tokyo, Japan. From 1987 to 1992, he was a researcher at ATR Interpreting Telephony Research Laboratories, Kyoto, Japan. Since 1992, he has been with the University of Tokushima, Tokushima, Japan, where he is currently a Professor in the Department of Information Science and Intelligent Systems. His current research interests include multimedia information retrieval, natural language processing, and music recognition. Prof. Kita is a member of IPSJ, IEICE and NLP.