JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 471-484 (2019)
DOI: 10.6688/JISE.201903_35(2).0013

Short Paper

An Efficient Search Algorithm for Fingerprint Databases*

GUANG-HO CHA
Department of Computer Engineering
Seoul National University of Science and Technology
Seoul, 139-743 Korea

In this paper, we present an efficient search algorithm for fingerprint databases that store songs or data with a similar structure. A song is represented by a high dimensional binary vector using the audio fingerprinting technique. Audio fingerprinting extracts from a song a fingerprint which is a content-based compact signature that summarizes an audio recording. A song can be recognized by matching an extracted fingerprint to a database of known audio fingerprints. In this paper, we are given a binary fingerprint database of songs and focus our attention on the problem of effective and efficient database search. However, the nature of high dimensionality and binary space makes many modern search algorithms inapplicable. The high dimensionality of fingerprints suffers from the curse of dimensionality, *i.e.*, as the dimension increases, the search performance decreases exponentially. In order to tackle this problem, we propose a new search algorithm based on inverted indexing, the multiple sub-fingerprint match principle, the offset match principle, and the early termination strategy. We evaluate our technique using a database of 2,000 songs containing approximately 4,000,000 sub-fingerprints and the experimental result shows encouraging performance.

Keywords: fingerprint database, binary database, audio fingerprint, similarity search, audio identification

1. INTRODUCTION

Large digital music libraries are becoming popular on the Internet and computer systems, and with their growth our ability to automatically analyze and interpret their content has become increasingly important. The ability to find acoustically similar, or even duplicate, songs within a large music database is a particularly important task with numerous potential applications. For example, the artist and title of a song could be retrieved given a short clip recorded from a radio broadcast or perhaps even sung into a microphone. Broadcast monitoring is also the most well known application for audio fingerprinting. It refers to the automatic playlist generation of radio, TV or Web broadcasts for, among others, purposes of royalty collection, program verification and advertisement verification.

Due to the rich feature set of digital audio, a central task in this process is that of extracting a representative audio fingerprint that describes the acoustic content of each song. Fingerprints are short summaries of multimedia content. Similar to a human fin-

Received April 6, 2012; revised June 19, & August 29, 2012; accepted October 23, 2012.
Communicated by Hsin-Min Wang.
* This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03036561).

gerprint that has been used for identifying an individual, an audio fingerprint is used for recognizing an audio clip. An ideal fingerprinting system should fulfill several requirements. It should be able to accurately identify an item, regardless of the level of compression and distortion or interference in the transmission channel. We hope to extract from each song a feature vector that is both highly discriminative between different songs and robust to common distortions that may be present in different copies of the same song.

Haitsma and Kalker's fingerprinting [1-3] is the seminal work on fingerprinting. It extracts 32-bit sub-fingerprints for every interval of 11.6 milliseconds from a song and the concatenation of 32-bit sub-fingerprints extracted constitutes the fingerprint of the song. Each sub-fingerprint is derived by taking the Fourier transform of 5/256 second overlapping intervals from a song, thresholding these values, and subsequently computing a 32-bit hash value. This technology was acquired by Gracenote, a music database company in Emeryville, California [4]. Wang's fingerprinting [5, 6] is also the seminal work on fingerprinting and known to be very robust against noise and distortion caused by using a mobile phone connection and added background noise. His algorithm was employed in the Shazam music recognition service [7]. Shazam's audio fingerprinting first generates a spectrgram for a song. The spectrogram is a 3 dimensional graph with time, frequency, and intensity. Each point on the graph represents the intensity of a certain frequency at a specific point in time. The Shazam algorithm identifies peak intensities (called anchor points) and zones in the vicinity of them (called target zones). For each point in the target zone, it creates a hash value that is the aggregation of the following: the frequency at which the anchor point is located (f1) + the frequency at which the point in the target zone is located (f2) + the time difference between the time when the point in the target zone is located in the song (t2) and the time when the anchor point is located in the song (t1) + t1. In this paper, we adopt an audio fingerprinting technique of Seo *et al.* [8] based on the normalized spectral subband moments in which the fingerprint is extracted from the 16 critical bands between 300 and 5300 Hz. Therefore, fingerprint matching in out work is performed using the fingerprint from 20-second music clips that are represented by about 200 subsequent 16-bit sub-fingerprints.

Given this fingerprint representation, the focus of our work has been to develop an efficient search method for song retrieval. This problem can be characterized as a nearest neighbor search in a very high dimensional (*i.e.*, 3200 (= 200 × 16)) data space.

High dimensional nearest neighbor search (NNS) is a very well studied problem: given a set P of points in a high dimensional space, construct a data structure which, given any query point q, finds the point p closest to q under a defined distance metric. The NNS problem has been extensively studied for the past two decades. The results, however, are far from satisfactory, especially in high dimensional spaces [9]. Most NNS techniques generally create a tree-style index structure, the leaf nodes represent the known data, and searching becomes a traversal of the tree. Specific algorithms differ in how this index tree is constructed and traversed. However, most tree structures succumb to the curse of dimensionality, that is, while they work reasonably well in a 2 or 3 dimensional space, as the dimensionality of the data increases, the query time and data storage would exhibit an exponential increase, thereby doing no better than the brute-force sequential search [9]. Recent work [10-12] appears to acknowledge the fact that a perfect search that guarantees to find exact nearest neighbors in a high dimensional space

is not feasible. However, the aforemention methods cannot be extended to the fingerprinting system which deals with the binary vectors and the Hamming distance metric. That is, in the fingerprinting system, the bit error rate between two binary vectors is used as a distance metric rather than Euclidean distance. In addition, the objective of the fingerprinting system is not to search the object most similar to the query object but to identify the query object with errors. Therefore, the above approaches cannot be applied to the fingerprinting system. Moreover, the big limitation of their work [10-12] is that it often needs to search a significant percentage of the database.

In this paper, we adopt the inverted file [13] as the underlying index structure and develop not only the technique to apply the inverted file indexing to high dimensional binary fingerprint databases but also the efficient search algorithm for fast song retrieval. Though our work focuses on searching songs based on audio fingerprints, the devised technique is generally applicable to other high dimensional binary vector search domains.

The organization of the paper is as follows. We begin by discussing the related work in Section 2. Section 3 is the heart of the paper where we prersent our indexing and search algorithm. We present a performance evaluation in Section 4 and conclusion is made in Section 5.

2. RELATED WORKS

The problem we investigate is: given a 3200-bit vector (200 16-bit sub-fingerprints) with errors, how to quickly find its nearest neighbor in the 3200-dimensional Hamming space.

Haitsma and Kalker [1-3] proposed an indexing scheme that constructs a lookup table (LUT) for all possible 32-bit sub-fingerprints and lets the entries in the LUT point to the song(s) and the positions within that song where the respective sub-fingerprint value occurs. Since a sub-fingerprint value can occur at multiple positions in multiple songs, the song pointers are stored in a linked list. Thus one sub-fingerprint value can generate multiple pointers to songs and positions within the songs. By inspecting the LUT for each of the 256 extracted sub-fingerprints, a list of candidate songs and positions is generated. Then the query fingerprint block (*i.e.*, 256 sub-fingerprints) is compared to the positions in the database where the query sub-fingerprint is located.

The first problem of the Haitsma-Kalker method is that the 32-bit LUT containing 232 entries is too large to be resident in memory. Furthermore, LUT is very sparsely filled because only a limited number of songs reside in comparison with the size of LUT. By adopting inverted file indexing we resolve this problem.

In an inverted file index, a list of all indexing terms is maintained in the search structure called a vocabulary, and the vocabulary is usually implemented by the B-trees. However, in our approach, we employ a hash table instead of the B-trees as the vocabulary in order to accomplish the lookup time of $O(1)$ rather than $O(\log n)$. Contrary to large text databases that widely use the inverted file index where the number of query terms is a few, in fingerprint querying, the number of query terms (*i.e.*, query sub-fingerprints) is several hundred when we assume the duration of the query song clip is several seconds. Therefore, the lookup time of $O(1)$ is crucial.

The second problem of the Haitsma-Kalker method is that their search algorithm is

built on the "single match principle". That is, their method assumes that if two fingerprints are similar, they would have a relatively high chance of having at least one "matching" identical sub-fingerprint, and therefore their method fetches the full fingerprint of the song matched from a database and compares it with the query fingerprint as soon as it finds a single sub-fingerprint matched to a certain query sub-fingerprint. They ignore the multiple occurrences of matching. However, in fact, multiple occurrences of sub-fingerprint matched are common and many candidates with multiple matches are found during the search. Therefore, if the multiple occurrences of matches are not considered in the query evaluation, the search wastes much time to inspect the candidate songs which are eventually judged to be incorrect even though they have several matches to the query sub-fingerprints. We tackle this spurious match problem and improve the search performance by introducing the "multiple sub-fingerprints match principle". It means that the full fingerprint of the candidate song should be retrieved only when the number of accumulated matches of it meets a specified threshold.

We also introduce the "offset match principle" in the search. It means that if two fingerprints are similar and there are multiple occurrences of sub-fingerprint matches between them, they might share the same relative offsets among the occurrence positions of matches. This offset match principle improves the search performance greatly by excluding the candidates that do not share the same relative offsets of match occurrence positions with the query fingerprint. This reduces the number of random database accesses remarkably.

In searching the database, the Shazam algorithm [5, 6] is the extreme opposite of the Haitsma-Kalker algorithm because it selects the candidates to fetch their full fingerprints from the database according to the order of scores based on the number of matching and time-aligned sub-fingerprints. But it is similar to our method that employs the multiple match principle and the offset match principle. However, Shazam fails to improve the search performance by not considering the fact that the search can be terminated earlier without scanning the full query sub-fingerprints. Although this approach actually reduces the number of full fingerprint fetches due to spurious matches, it misses the chance of early termination because it has to compute the similarity scores of every candidate and determine the order of fetching their full fingerprints from a database. We improve the search performance by applying the "early termination strategy" to the search algorithm.

Miller *et al.* [14] assumed the fingerprint representation of 256 32-bit sub-fingerprints of Haitsma and Kalker [1-3] and proposed the 256-ary tree to guide the fingerprint search. Each 8192(= 32×256)-bit fingerprint is represented as 1024 8-bit bytes. The value of each consecutive byte in the fingerprint determines which of the 256 possible children to descend. A path from the root node to a leaf defines a fingerprint. The search begins by comparing the first byte of the query with the children of the root node. For each child node, it calculates the cumulative number of bit errors seen so far. This is simply the sum of the parent errors and the Hamming distance between the 8-bit value represented by the child and the corresponding 8-bit value in the query. Then a test is applied to each child, in order of increasing error, to determine whether to search that child. If the best error rate seen so far is greater than the threshold, then the child is searched. The search continues recursively and when a leaf node is reached, the error rate associated with the retrieved fingerprint is compared to the best error rate seen so far.

If it is less, then it updates the best error rate to this new value and assigns this fingerprint as the best candidate nearest neighbor so far.

The first problem of Miller *et al.*'s method is that the size of the 256-ary tree is too large and the depth of the tree is also too deep to be practical in the disk-based database search. According to their experimental results [14], they search an average of 419,380 nodes, which is 2.53% of the nodes in the index tree that stores about 12,000,000 sub-fingerprints.

Moreover, they assume that each song is also represented by a fingerprint with 8192 bits, *i.e.*, the same number of bits as the query fingerprint. It means that the length of each song in a database is assumed to be the same as that of the query song. It makes the indexing and search problem simpler. But actually an individual song with an average length of 4 minutes has approximately 10,000 sub-fingerprints in Haitsma-Kalker's method. Therefore, it is not practical to model a song with only an 8192-bit fingerprint and thus this mechanism is not feasible to apply to real applications.

The third problem is that Miller *et al.*'s 256-ary tree uses a probabilistic method based on a binomial distribution to estimate the bit error rate (BER) in each tree node. This BER is used to determine whether to search that node. However, it is difficult to predict the exact BER in advance, and therefore, the correct rate to find the most similar fingerprint is at most 85% in Miller *et al.*'s method [14]. In order to increase the correct rate, the expected BER in each node should be determined more conservatively, and in that case, the search performance may degenerate to be worse than that of the brute-force sequential search. Therefore, it is difficult to reduce the search space to find the nearest neighbor in a high-dimensional space using the *k*-ary tree. Furthermore, if the *k*-ary tree tries to reduce the search space more, the error rate increases inevitably. It means that the *k*-ary tree approach cannot overcome the curse of dimensionality problem.

Keeping those limitations in mind, in this paper, we propose a new indexing and search algorithm that resolves the limitations explained above.

3. INDEXING AND SEARCH ALGORITHM

We now describe a new indexing scheme and a new search algorithm for song databases implemented with audio fingerprints. The underlying structure of our indexing is based on the inverted file that has been widely used for text query evaluation such as Google [15]. Searching an audio fingerprint database to identify a song is similar to searching a text database for documents because a song is represented by a sequence of 16-bit sub-fingerprints and it can be found using some of the sub-fingerprints, and similarly, documents in a text database are represented by multiple keywords and they can be searched by keyword matching. This is the reason why we adopt the inverted file as the underlying index structure in our work. However, there are also many differences between them and they make the fingerprint search problem more difficult.

In the fingerprint search, the query fingerprint may not match any fingerprint in a database because the fingerprint extracted from a query song may have bit errors due to distortions to the query song. In other words, contrary to the text database search where only exact matching is supported, the fingerprint search should identify the correct song in a database even though there is a severe signal degradation of the query song. This means that the fingerprint search must support imperfect matching or similarity search.

Individual bits in a fingerprint have their own meaning, and therefore, the Hamming distance between two fingerprints is used as a dissimilarity metric. This means that the search problem is: given a 3200 (= 16×200)-bit vector with errors, how to find the vector most similar to that in the 3200-dimensional binary space. However, the high-dimensional similarity search is known to suffer from the dimensionality curse. As aforementioned, the indexing method such as the *k*-ary tree approach cannot avoid the dimensionality curse problem.

The query fingerprint is assumed to be 200 16-bit sub-fingerprints in our work. However, assuming that the duration of the average song is 4 minutes, then the number of sub-fingerprints in a song is approximately 2,000 in our system. This difference of lengths between the query song and songs in a database makes the search problem more difficult.

We resolve the problems explained above by adapting the inverted file index suitably to our high dimensional binary database and creating the search algorithm with several sophisticated strategies.

3.1 Index Structure

An inverted file index works by maintaining a list of all sub-fingerprints in a collection, called a *vocabulary*. For each sub-fingerprint in the vocabulary, the index contains an *inverted list*, which records an identifier for all songs in which that sub-fingerprint exists. Additionally, the index contains further information about the existence of the sub-fingerprint in a song, such as the number of occurrences and the positions of those occurrences within the song.

Specifically, the vocabulary stores the following for each distinct 16-bit sub-fingerprint t,

- a count f_t of the songs containing t,
- the identifiers s of songs containing t, and
- the pointers to the starts of the corresponding inverted lists.

Each inverted list stores the following for the corresponding sub-fingerprint t,

- the frequency $f_{s,t}$ of sub-fingerprint t in song s, and
- the positions p_s within song s, where sub-fingerprint t is located.

Then the inverted lists are represented as sequences of $<s, f_{s,t}, p_s>$ triplets. These components provide all information required for query evaluation. A complete inverted file index is shown in Fig. 1.

t	f_t	s	Inverted list for t
1100010010001010	1	<5>	5(2: 24, 45)
0100000111001011	2	<4, 34>	4(1: 8), 34(3:78, 90, 234)
1100110001100001	1	<77>	77(1: 18)
⋮	⋮	⋮	⋮
1010111010101001	3	<102, 981, 1201>	102(1: 62), 981(2: 12, 90), 201(2: 99, 187)

Fig. 1. Inverted file index. The entry for each sub-fingerprint t is composed of the frequency f_t, song identifiers s, and a list of triplets, each consisting of a song identifier s, a song frequency $f_{s,t}$, and the positions p_s within song s, where sub-fingerprint t is located.

The vocabulary of our indexing scheme is maintained as a hash table instead of the B-trees in order to achieve the lookup time approaching $O(1)$ rather than $O(\log n)$. Typically, the vocabulary is a fast and compact structure that can be stored entirely in main memory.

For each sub-fingerprint in the vocabulary, the index contains an inverted list. The inverted lists are usually too large to be stored in memory, so a vocabulary lookup returns a pointer to the location of the inverted list on disk. We store each inverted list contiguously in a disk rather than construct it as a sequence of disk blocks that are linked. This contiguity has a range of implications. First, it means that a list can be read or written in a single operation. Accessing a sequence of blocks scattered across a disk would impose significant costs on query evaluation. Second, no additional space is required for next-block pointers. Third, index update procedures must manage variable-length fragments that vary in size, however, the benefits of contiguity greatly outweighs these costs.

3.2 Indexed Search

The query evaluation process is completed in two stages. The first stage is a *coarse index search* that uses the inverted file index to identify candidates that are likely to contain song matches. The second stage is a *fine database search* that fetches each candidate's full fingerprint from a song database and then computes the similarity between the query fingerprint and the song's fingerprint fetched. The fine search is more computationally expensive because it requires the random disk access. Therefore, our strategy is to avoid the expensive random disk accesses as possible as we can.

To conduct the coarse index search, we use a ranking structure called *accumulator*. The accumulator records the following:

- the accumulated occurrences of the song identifiers (IDs) matched,
- the matching positions both within the query and the matched song IDs, and
- the accumulated number of matches that have the same relative offsets between the matching positions within the query and the candidate songs when there are multiple matches. We call this *offset-match-count*.

Ultimately, the information that we need is the offset-match-count for every candidate of a search. If a specific song ID has been encountered and its offset-match-count has reached a certain threshold, then we load the full fingerprint from database using the retrieved song ID. The subsequent comparison is based on the Hamming distance between the query fingerprint and the song fingerprint on their matching positions. Computing the Hamming distance involves counting the bits that differ between two binary vectors.

In practical applications, many search candidates that have a single match or even multiple matches with query sub-fingerprints are generated even though they are not the correct object that we are looking for. Therefore, a significant percentage of a database needs to be searched if the search algorithm loads the full fingerprint of a candidate as soon as it encounters the candidate whose sub-fingerprint matches a certain query sub-fingerprint. In other words, a search strategy such as the Haitsma-Kalker method based on the single match principle is inevitably inefficient in disk-based applications although this problem may be less evident if all data are resident in memory. In fact, the candidate

whose offset-match-count has reached a certain threshold (*e.g.*, 3) has a great possibility of being the correct object that we are seeking and there are almost no candidates that have their offset-match-count reach the threshold and are not the correct answer. If two fingerprints are similar and there are multiple occurrences of sub-fingerprint match between them, they may share the same relative offsets among the occurrence positions of matches.

The search algorithm using an inverted file index is illustrated in Fig. 2 and described in Fig. 3. There are six cost components in the fingerprint search, as summarized in Table 1. The first one is to initialize the array *accumulator* that records the accumulated number of matches that have the same relative offset between the matching positions of the query and the retrieved song IDs. The second one is to compute n sub-fingerprints from a query song clip. These two operations are computed very quickly. The third one is to retrieve the index information about songs stored in the inverted file index. This I/O operation is fast because the index information is lightweight and several inverted lists can be read in a single operation since they are contiguously stored in a disk. The fourth one is to count the accumulated occurrences of the IDs and their offset-match-counts. This simply involves read/write accesses to the memory array. The fifth operation is to fetch the full fingerprints of the candidate songs. This is the most expensive I/O operation that includes the random disk accesses to the candidate song's finger print database. The final one is to fully compare the retrieved candidate fingerprint with

Table 1. Cost components in fingerprint search.

No	Components	Computation	Operation	Cost
1	Initialize accumulator	$O(1)$	memory	very fast
2	Compute n sub-fingerprints	$O(1)$	memory	fast
3	Retrieve song IDs	$O(n)$	disk	lightweight
4	Count occurrences	$O(n)$	memory	very fast
5	Load full fingerprint	$O(n)$	disk	heavyweight
6	Compute Hamming Distance	$O(n)$	memory	fast

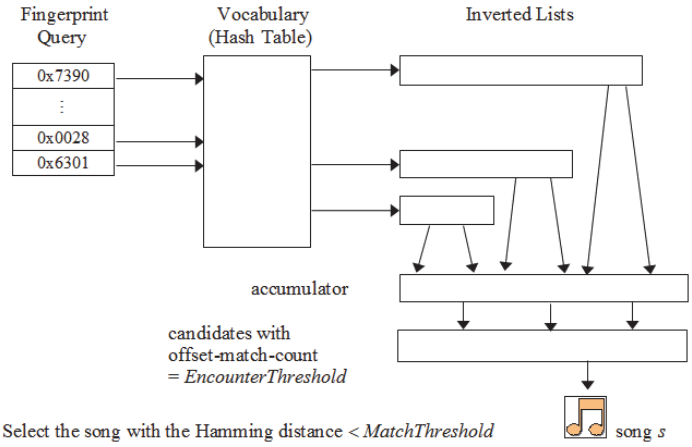


Fig. 2. Using an inverted file index and an accumulator to calculate song's similarity score.

the query fingerprint and this operation is also computed fast. Therefore, our strategy is to make the best use of the fast operations 3 and 4 while avoiding the most expensive operation 5 and the operation 6.

In the search algorithm, the query sub-fingerprints are all generated at a time. Initially each song has a similarity of zero to the query, this is represented by creating the array accumulator A initialized by zero. The count of song occurrences and offset-match-count of $A[j]$ are increased whenever the match of song j and the match of position offset are found, respectively. Contrary to the Haitsma-Kalker method that fetches the full song fingerprint from a database and compares it with query as soon as it finds a candidate matched to a query sub-fingerprint, our search algorithm postpones it until the offset-match-count of the candidate reaches a certain threshold (EncounterThreshold in the algorithm of Fig. 3) in order to avoid the expensive operations 5 and 6 in Table 1 as much as possible. Based on our experimental results, EncounterThreshold = 3 shows a suitable trade-off between speed and accuracy. Without considering the offset match principle and the multiple match principle, the search algorithm would be similar to the Haitsma-Kalker method.

[Algorithm] Fingerprint Search
Input: Query song clip Q
Output: Match song P or 'No match'

```
1: Initialize accumulator A[j], 1 ≤ j ≤ |DB|, for each song j in a database (DB).
2: Compute n sub-fingerprints t1, t2, ..., tn (e.g. n = 200) from a query Q.
3: for i = 1 to n do {
4:       IDs ← InvertedFileIndex[ti]                    // coarse index search
5:       for j ∈ IDs do {
6:             Increment occurrences of j in A[j] by 1.
7:             Compute the offset-match-count c of j in A[j].
8:             if c = EncounterThreshold then {
9:                   P ← DB[j]                          // fine database search
10:                  if HammingDistance(P, Q) < MatchThreshold then
11:                        Return P;                    // early termination
12:            }
13:      }
14: }
15: Return 'No match'
```

Fig. 3. Indexed computation of similarity between a query Q and a fingerprint database.

Besides the multiple match principle and the offset match principle, another method to hasten our search algorithm is the "early termination strategy" shown in steps 10 and 11 in the algorithm of Fig. 3. The fewer errors in a query, the more likely the match is found at an early stage. Even before the full search of n (*e.g.*, $n = 200$) sub-fingerprints is completed, if a song's offset-match-count meets the condition of EncounterThreshold, then the song is fully compared with query and it can be reported as a match if its Hamming distance to the query is less than a certain threshold (MatchThreshold in the algorithm of Fig. 3). This early search termination also contributes to the speedup. Without considering the early termination strategy and marking the similarity score of all candidate matches, the search algorithm may be similar to the Shazam algorithm.

4. EXPERIMENTAL RESULTS

In order to evaluate our indexing scheme and search algorithm, we digitized 2,000 songs and computed about 4,000,000 sub-fingerprints from the songs. In our experimental database containing 2,000 songs, the size of inverted file index is about 3 GB, which is more than many computers can afford to put it into memory. For the applications we envision, *e.g.*, music search and identification, the database must be very large, perhaps one million songs. If we assume approximately 2,000 unique sub-fingerprints per song, this means we may wish to search through about 2 billion sub-fingerprints. Under this case, all the inverted lists cannot reside in memory.

For experiments, 1,000 queries were generated by randomly selecting 20-second portions from the song database and playing them through inexpensive speakers attached to a PC. These song snippets were then digitized to be used as queries using an inexpensive microphone. For each query, we know the answer, *i.e.* correct song, because we know which song each query is derived from. Therefore, we can compute the error rate that could not identify the correct song. In addition, we generated 100 queries from songs not stored in a database to evaluate the performance when a search returns no match.

We compared our method with the Haitsma-Kalker method [1-3] and the Shazam method [5, 6] to assess the performance of ours. We simulated the Haitsma-Kalker method by using the offset-match-count = 1 based on the fact that it is built on the single match principle. The experiments of our algorithm was conducted using offset-match-count = 3, *i.e.*, *EncounterThreshold* = 3 in the algorithm in Fig. 3. The effect of various values of *EncounterThreshold* is reported in Table 2.

Moreover, in the experiment, the matching threshold (*MatchThreshold*) is set at 0.3. Fig. 4 shows the distribution of the normalized Hamming distances for our experimental database based on $2000 \cdot (2000-1)/2 = 1{,}999{,}000$ cross comparisons among 2,000 different songs in the database. In the experiment, the normalized minimum Hamming distance among 2,000 different songs is 0.31, the maximum distance is 0.72, and the mean is 0.52. Based on this experimental results, we adopt the value of 0.3 as *MatchThreshold*. According to this experimental result, if we use the larger value of *MatchThreshold*, the false positive rate may increase, on the other hand, the smaller value of *MatchThreshold* may increase the false negative rate.

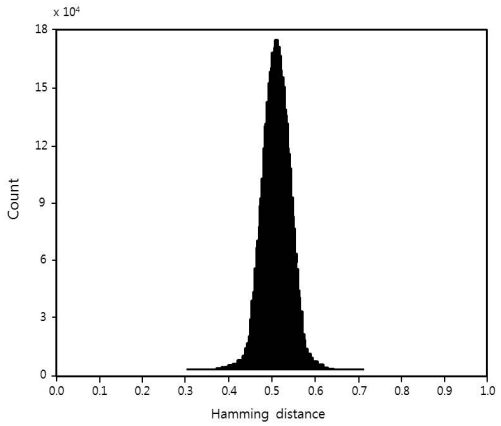


Fig. 4. Histogram of the experimental database (2,000 songs).

Table 2. Search ratios, search time and false negative error rate depending on the offset-match-count in our algorithm (offset-match-count = 3), Haitsma-Kalker's (offset-match-count = 1), and Shazam's algorithm.

offset-match-count (*EncounterThreshold*)		1 (Haitsma-Kalker algorithm)	2	3 (our algorithm)	4	Shazam algorithm
search ratio	when no matches are found	0.58	0.0700	0.0053	0.0007	0.0007
	when matches are found	0.45	0.0017	0.0006	0.0005	0.00049
average search time (msec)		3220	140	30	35	59
false negative error rate		0.0048	0.0063	0.0075	0.0098	0.0049

We define the *search ratio* as the proportion of full fingerprints retrieved for comparison to the whole database size. The search ratio in our algorithm is less than 0.001 in most cases and it means that our algorithm retrieves only a fraction of the whole database. This is due to the multiple match principle and the offset match principle. On the other hand, the Haitsma-Kalker method based on the single match principle could not achieve this performance. Although the Shazam algorithm is comparable to ours with respect to the search ratio, its search speed is not better than that of ours because it does not make advantage of the early termination strategy.

First, we study the performance when a search returns no match, hence has no early termination. We conducted the experiment using the songs not stored in the database. In our algorithm, there is scarcely the case to load the full fingerprints of candidates from the database when a match is not found. This is mainly due to the offset match principle. However, in the Haitsma-Kalker method, a significant percentage of the database needs to be searched even though there is no match, *i.e.* its search size may approach the database size. This inefficiency is caused by the single match principle that retrieves the full fingerprint of the candidate from the database as soon as it finds a single sub-fingerprint match.

Table 2 reports the search ratios of the competitors and the effect of the *EncounterThreshold*. When we use the offset-match-count (*EncounterThreshold* in the algorithm in Fig. 3) = 3, our algorithm reports the average search time of 30 msec. This is achieved because the algorithm checks only 0.06% of the database. When offset-match-count = 1, a significant percentage (from 45% to 59%) of the database need to be searched. It is particularly inefficient to retrieve sporadically dispersered data from disk. Fingerprints in a database are actuallly retrieved randomly and it is very costly. On the other hand, when offset-match-count $\geq$ 2, only a small fraction of the database are read. When offset-match-count = 4, only 0.05% of the database are checked. However, its gain is offset by the search speed. This is caused by the delayed early termination. As a result, the value offset-match-count = 3 becomes a suitable trade-off between speed and accuracy. The search ratio of the Shazam algorithm is almost the same as the case of offset-match-count = 4. It means that the candidate with offset-match-count = 4 is almost the song we want to find.

Second, let us study the effect of early termination on the speedup. In fact, the fewer errors in a query, the more likely the match found at an early stage. Fig. 5 plots the average percentage of having early termination at each sub-fingerprint position in query

fingerprints. It shows that most matches are found by scanning no more than 60 sub-fingerprints (about 30%) in a query fingerprint. This demonstrates that early termination contributes to the speedup.

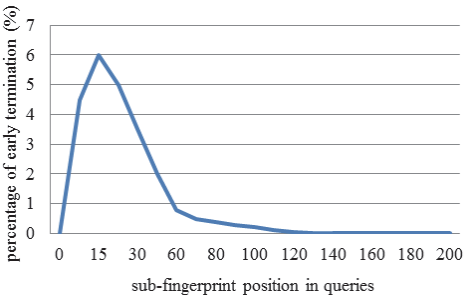


Fig. 5. Percentage of early termination at each sub-fingerprint in a query.

Third, let us consider the false negative error rate in the search. Compared with the case of offset-match-count = 1, the degradation of false negative error rates when when offset-match-count ≥ 2 is negligible, *i.e.*, from 0.0048 to 0.0098 as shown in Table 2. It means that the false negative error rate of our algorithm is less than 1% although we adopt the offset match principle to avoid the expensive operation of retrieving full fingerprints. In effect, the fewer error bits in a query, the more likely the false negative error is reduced.

Fourth, In order to experiment the case of the frame-drop situation in our search algorithm, we intentionally omitted random 5% and 10% of sub-fingerprints in a query fingerprint and evaluated the experimental results. In this case, the number of sub-fingerprints in a query fingerprint decreases by 5% to 10%. The experimental results under this noisy condition show that the false negative rate, *i.e.*, the error rate that incorrectly reports "no match found", increases up to 4.8% (in the case of 10% time scaling) compared to the cases using original query fingerprints. Fig. 6 shows the false negative error rate in the case of the frame-drop situation. It demonstrates that our approach is still robust under high noisy condition.

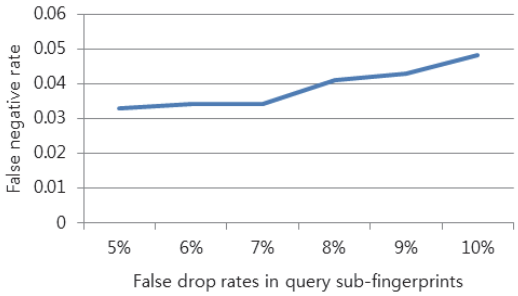


Fig. 6. False negative error rate for the queries intentionally omitted 5% to 10% sub-fingerprints in query fingerprints.

Finally, in the speed test, our method with offset-match-count = 3 is faster than the Haitsma-Kalker method (offset-match-count = 1) and the Shazam method. This speedup is achieved since our search algorithm checks only the most likely song to have a correct match. This is due to employing the strategy of postponing the access of database to fetch the full fingerprint of a song as well as the early termination strategy without considering all sub-fingerprints. The above experimental results show both of the effectiveness and efficiency of our indexing and search strategy.

5. CONCLUSION

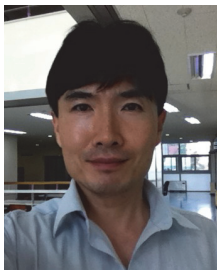
In this paper, we proposed a new search algorithm based on inverted indexing for efficiently searching a large fingerprint database. The indexing method employs the inverted file as the underlying index structure to achieve the efficient song retrieval. The search algorithm adopts the "multiple match principle", the "offset match principle", and the "early termination strategy", so that it postpones the fetch of full fingerprints and therefore reduces the number of expensive random disk accesses dramatically.

The experimental result shows the performance superiority of our method to that of Haitsma-Kalker's and Shazam's. This makes our new search strategy a useful technique for efficient fingerprint database searches including the application of music retrieval.

REFERENCES

1. J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system with an efficient search strategy," *Journal of New Music Research*, Vol. 32, 2003, pp. 211-221.
2. J. Haitsma and T. Kalker, "Highly robust audio fingerprinting system," in *Proceedings of International Symposium on Music Information Retrieval*, 2002, pp. 107-115.
3. J. Oostveen, T. Kalker, and J. Haitsma, "Feature extraction and a database strategy for video fingerprinting," in *Proceedings of International Conference on Visual Information Systems*, Vol. LNCS 2314, 2002, pp. 117-128.
4. http://www.gracenote.com.
5. A. Wang, "The Shazam music recognition service," *Communications of the ACM*, Vol. 49, 2006, pp. 44-48.
6. A. Wang, "An industrial-strength audio search algorithm," in *Proceedings of International Symposium on Music Information Retrieval*, 2003, pp. 713-718.
7. http://www.shazam.com.
8. J. S. Seo, M. Jin, D. Jang, S. Lee, and C. D. Yoo, "Audio fingerprinting based on normalized spectral subband moments," *IEEE Signal Processing Letters*, Vol. 13, 2006, pp. 209-212.
9. G.-H. Cha, X. Zhu, D. Petkovic, and C.-W. Chung, "An efficient indexing method for nearest neighbor searches in high-dimensional image databases," *IEEE Transactions on Multimedia*, Vol. 4, 2002, pp. 76-87.
10. A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of VLDB Conference*, 1999, pp. 518-529.
11. P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*, Springer, 2006, pp. 145-159.

12. C. C. Aggarwal and P. S. Yu, "On indexing high dimensional data with uncertainty," in *Proceedings of SIAM Data Mining Conference*, 2008, pp. 621-631.
13. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison-Wesley, 1999, pp. 192-198.
14. M. L. Miller, M. C. Rodriguez, and I. J. Cox, "Audio fingerprinting: Nearest neighbor search in high dimensional binary spaces," *Journal of VLSI Signal Processing*, Vol. 41, 2005, pp. 285-291.
15. S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, Vol. 30, 1998, pp. 107-117.



Guang-Ho Cha received his Ph.D. degree in Computer Science from Korea Advanced Institute of Science and Technology, Daejon, South Korea in 1997. He is now a Professor at Department of Computer Engineering, Seoul National University of Science and Technology, Seoul, South Korea. His research interests include content-based media retrieval, uncertain databases, and probabilistic queries.