Journal of Information Science OnlineFirst, published on May 8, 2008 as doi:10.1177/0165551507087712

JIS

An efficient frequent melody indexing method to improve the performance of query-by-humming systems

Jinhee You and Sanghyun Park

Department of Computer Science, Yonsei University, Korea

Inbum Kim

Division of Computor, Kimpo College, Korea

Abstract.

In recent years, the need to efficiently store and retrieve large amounts of musical information has increased. In this paper, we design and implement a Query-By-Humming (QBH) system, which can retrieve melodies similar to users' humming. To make this QBH system efficient, the following three methods were proposed. First, we convert the melodies to be indexed into the corresponding strings, in order to increase search speed. The conversion method is designed to tolerate the errors involved in humming. Second, we extract significant melodies from music and then build a couple of indexes from them. For this task, we propose reliable methods for extracting melodies that occur frequently and for melodies that begin after a long rest. Third, we propose a three-step index searching method for minimizing database access. Through the experiments with a real-world data set, it was verified that this system has noticeable improvements over the N-gram approach.

Keywords: content-based indexing method; multimedia database; music information retrieval; query-by-humming system

1. Introduction

As the use of multimedia information is becoming more common, users require new methods to retrieve multimedia information more easily and more quickly than the current methods. In order to satisfy these demands, many studies have been conducted. Currently, the most common approach for finding multimedia content is text-based retrieval. The text-based retrieval method finds multimedia content using keywords that are previously annotated. However, users cannot use the text-based method when they do not remember the keywords associated with desired multimedia content. Also, since the text-based method only searches for the multimedia content that exactly matches a user's query, it is not appropriate for similarity searching. In particular, the

Correspondence to: Sanghyun Park, Department of Computer Science, Yonsei University, Korea. Email: sanghyun@cs.yonsei.ac.kr

text-based retrieval method is most common in the field of music information retrieval. Using the text-based method, users find the desired music by submitting keywords such as the music title, the album title, and the name of the singer. However, the text-based method is not appropriate for music information retrieval when users do not remember relevant keywords or they want to find melodies that are similar to the desired music.

There have been many studies [1–3] which employ the content-based retrieval method in order to solve the problems aforementioned. Unlike the text-based method, the content-based method finds the desired information by directly comparing multimedia content. In addition, since the content-based method computes the similarity between multimedia content during the retrieval process, it is capable of performing similarity searching.

Query-By-Humming (QBH) [4] is a study of the content-based method in the area of music information retrieval. The approach of QBH is to find the music that contains melodies that are similar to the queries hummed via microphone. Since QBH systems are capable of retrieving the desired music even when users' humming is inaccurate, they are useful for people who are not talented singers or who have a speech impediment. However, QBH systems also have drawbacks. That is, if a database contains a large amount of music, then the operation to find the music similar to a user's humming requires complicated computation and long search times. This is because QBH systems have to sequentially read music from a database, and then measure the similarity between each one of its melodies and the user's humming by calculating the distance with representative numeric data.

In this paper, we propose an efficient QBH system. In order to reduce the excessive cost for calculating the distance between numeric melodies, we convert the numeric melodies into the corresponding strings and then compare the strings using some existing string matching algorithms. Also, to guarantee the accuracy of the search results, we design the conversion method to tolerate the errors involved in humming. In addition, to reduce the search time, we extract significant melodies, which have a high chance of being queried, from music and then build a couple of indexes from them. The definition of significant melodies is divided into two types in this paper:

1. a melody that appears frequently in a single piece of music; and
2. a rest-unit melody that begins after a long rest within the music.

Furthermore, we propose reliable methods for extracting melodies that occur frequently and for melodies that begin after a long rest.

One of the representative indexing methods for QBH systems is N-gram [5–7], where *N* consecutive notes are represented as a single symbol. The N-gram method has the advantage of simple indexing. However, since the N-gram method extracts N-grams from every possible position in the music, the number of N-grams to be stored is directly proportional to the length of the piece of music. Therefore, indexes are likely to become large when a music database is big. In addition, since the errors contained in humming are not considered when converting N-grams into corresponding symbols, the search result becomes less accurate as a user's humming contains more errors. In this paper, we try to overcome a number of the limitations of the N-gram method. Through experimental comparison with the N-gram method, we verify that our system has noticeable improvements in terms of the speed and accuracy of the search operations.

As shown in Figure 1, the proposed QBH systems have two major steps including system construction and query processing. In system construction,

1. feature data is extracted from the MIDI music;
2. feature data is stored in a feature database;
3. feature data of numeric type is converted to corresponding symbols of character type; and
4. significant string melodies are extracted and indexed.

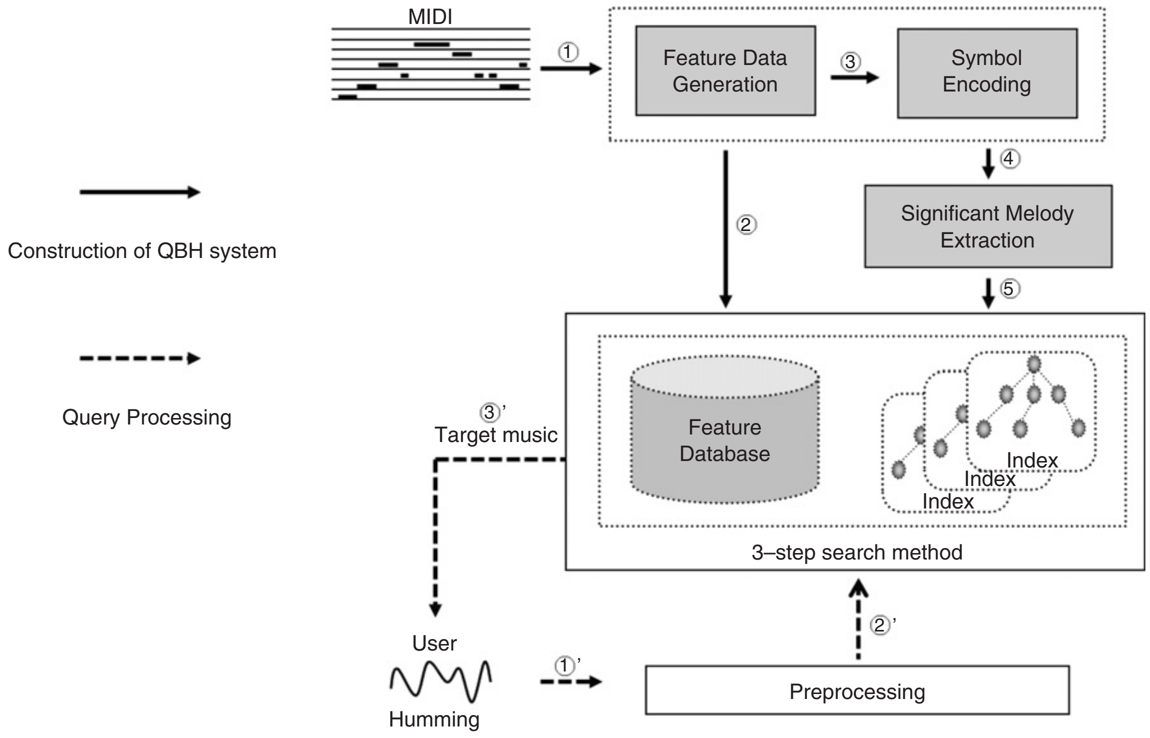

Fig. 1. Construction and query processing of the proposed QBH system.

In query processing,

1. the user's humming is preprocessed and, as a result, the corresponding query string is obtained; and
2. three-step index searches are performed in order to minimize database access and obtain query answers as early as possible.

The rest of the paper is organized as follows. Section 2 describes typical feature data and indexing methods commonly used in the area of content-based music retrieval. Sections 3 and 4 explain in detail the system construction and query processing steps of the proposed QBH system, respectively. Section 5 shows and analyzes the results of experimental comparison with the N-gram method. Finally, concluding remarks are given in Section 6.

2. Related work

Recently, studies on finding feature data suitable for content-based music retrieval have been actively conducted. Feature data that is most appropriate for a specific content-based music retrieval system depends on the type of query, which is either an accurate query such as an original song or an inaccurate query such as humming. In particular, the interest in finding feature data that is suited to QBH systems has been growing [4, 8]. UDS strings [9] are one of the typical feature data sets for QBH systems. UDS strings express the relationship between pitches as one of three letters: 'U', 'D', and 'S'. If the pitch of a note is higher than that of its previous note, it is denoted as 'U'. If the pitch of a note is lower than that of its previous note, it is denoted as 'D'. And, if the pitch of a note is the same as its previous note, it is denoted as 'S'. UDS strings have a limitation in accuracy because they represent only pitch contours while ignoring the durations of notes. Therefore, the accuracy of this type of search result is very low when the user's humming contains many errors. There is a QBH system that uses tempo as its feature data [10]. However, since tempo data ignores the pitch and duration information, this type of system becomes inaccurate when the user's humming is not accurate.

Kline and Glinert [7] proved that QBH systems that use feature data that combine the pitch interval and the duration ratio of adjacent notes produce the most accurate search results. This is because both the pitch and note duration is taken into account, and it expresses them using accurate numeric values rather than rough melody contours. Based on this study, it was decided to use pitch interval and the duration ratio of adjacent notes as the feature data in the proposed QBH system.

Another study on content-based music retrieval is to devise an efficient indexing method. The motivation for this study is that, when there are no appropriate index structures in a large music database, the operation to retrieve target music requires sequential access to the database and a consequently enormous number of comparisons and long search times. One of the most typical indexing methods for content-based music retrieval is N-gram [11, 12], where *N* adjacent notes are extracted at every possible position, which is just like sliding windows with *N* notes, and represented as a single symbol. Since the number of N-grams to be stored is proportional to the length of the music, indexes are likely to become large when a music database is big. In addition, since the errors contained in humming are not considered when converting N-grams into corresponding symbols, the search result becomes less accurate as the users' humming contains more errors. There are other studies that are based on spatial access methods. These studies treat the pitch and duration of a note as a point in two-dimensional space. One of the representative methods in this category uses R-tree [13]. Its operation to traverse down an R-tree is efficient. However, since there can be many notes that are mapped to the closely located points, its selectivity may not be good.

Most indexing methods for content-based music retrieval build the index for a specific part of a melody [14–16]. Assuming that users easily remember the beginning part of the music, Andreas [14] proposed to index only the initial part of the music. However, the proposed method does not guarantee good search performance when users query with a part of the music other than the beginning. Liu et al. [15] proposed to extract and index what is known as 'frequent melody'. However, since their method extracts frequent melodies without regular units, the extraction process is very complicated. In addition, since their method creates many candidates for frequent melody, their index becomes too large when the database is big. In contrast to the methods mentioned above, there is a different type of method [16], which does not index the music in databases, but rather indexes frequent queries using UDS strings. This has the advantage of obtaining short search times when the users query the melody that many other users have queried before. However, this has the drawback of accessing the entire database when some melodies are first queried. Moreover, UDS strings lead to low accuracy.

3. Construction of the QBH system

3.1. Feature extraction and symbol encoding

The first task in constructing the QBH system is to extract the melodies from the music. The melody is composed of a series of notes, which have the two basic components of pitch and duration. Therefore, when MIDI music is given, these two components are extracted from each note of the music. The pitch of a note is obtained directly from its octave that is previously defined as the value of pitch in MIDI format. The duration of the notes and rests are determined using the length of a quarter note (i.e. time base). Note that, since music may have a different time base, we must scale the time bases of the music to an identical value. After extracting these two components from each note, we must calculate the boundary of each bar using the time signature, the time base, and the duration of all the notes and rests.

Next, we calculate the feature data (i.e. pitch interval and duration ratio), in order to express the melody precisely. Given two adjacent notes (P_i, D_i) and (P_{i+1}, D_{i+1}), their feature data are calculated using the expressions in Figure 2. The feature data between two adjacent notes takes the form of two-dimensional vectors and it is stored in the feature database as the information of the former note.

A rest between adjacent notes is treated as shown in Figure 3. Most QBH system studies have ignored the treatment of rests, but users are likely to hum passages that include rests. Unlike notes, rests have only duration values. Therefore, we represent the pitch of the notes as '-'.

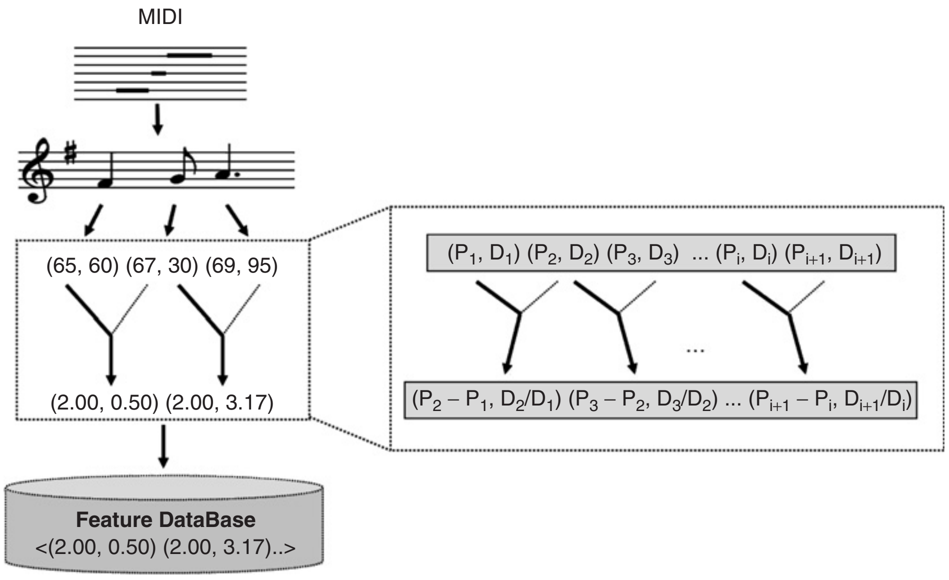
MIDI
(65, 60) (67, 30) (69, 95)
(2.00, 0.50) (2.00, 3.17)
Feature DataBase
<(2.00, 0.50) (2.00, 3.17)..>
(P1, D1) (P2, D2) (P3, D3) ... (Pi, Di) (Pi+1, Di+1)
...
(P2 – P1, D2/D1) (P3 – P2, D3/D2) ... (Pi+1 – Pi, Di+1/Di)

Fig. 2. Extraction of feature data.

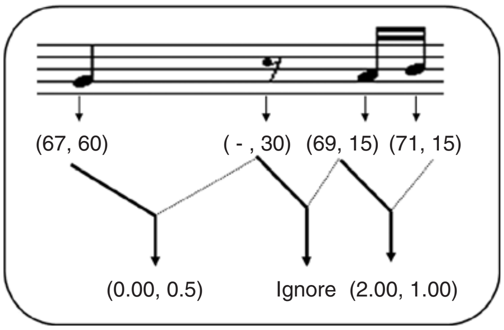

Fig. 3. Treatment of rests.

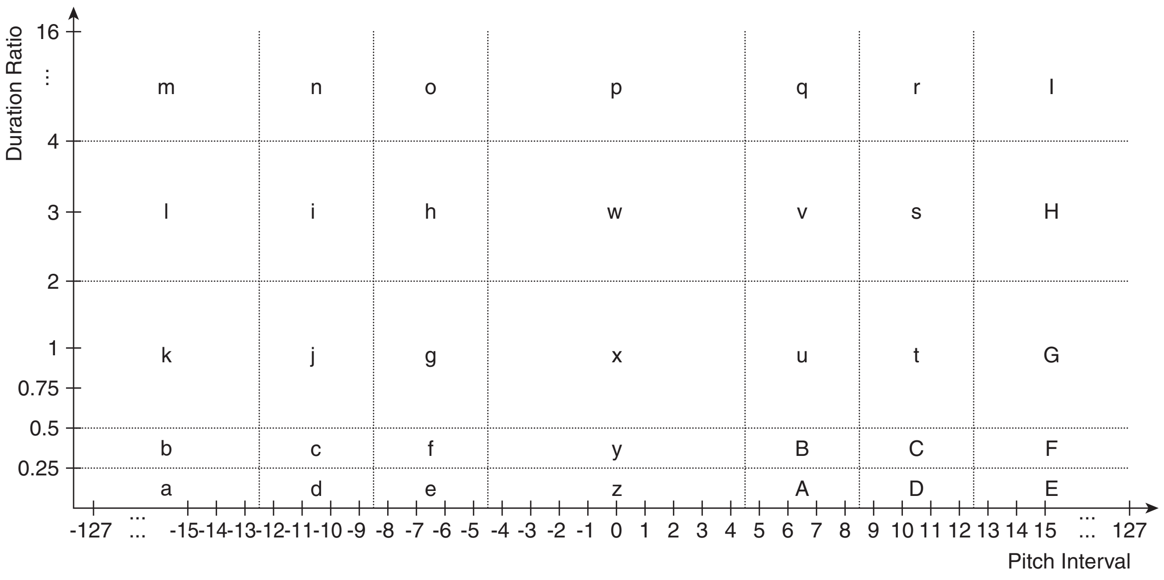

Fig. 4. Example of CES for changing two-dimensional vectors of pitch interval and duration ratio to corresponding characters.

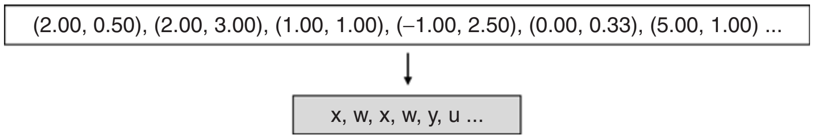

Fig. 5. Result of encoding a series of two-dimensional vectors into a corresponding string using the example of CES in Figure 4.

Feature data of a rest is calculated using either one of two ways according to the position. Consider the case where a rest follows a note. Since it is assumed that the pitch of the note does not change during the rest, the pitch interval is 0.00. The duration ratio is calculated using the expression shown in Figure 3 with the duration values of the former note and the latter rest. The calculated feature data is stored as the information of the former note. Unfortunately, the same feature data can be generated if two notes of the same pitch are adjacent to each other. However, because it is not easy for users to distinguish these two situations and hum them differently, they are not handled separately. Next, consider the case where a note follows a rest. In this case, the rest cannot be expressed as a melody (i.e. dumb melody). Therefore, the feature data of the rest is not calculated. As mentioned earlier, all of the extracted feature data are stored in the feature database.

To calculate the similarity between the query and each melody in the feature database, many methods use the Euclidean distance. However, the operation to calculate the Euclidean distance has a high CPU cost. Therefore, the performance of the methods based on the Euclidean distance deteriorates as the size of the database increases. To resolve this problem, we take the approach of converting the melodies, which are a series of two-dimensional numeric vectors, into corresponding strings of characters, and then employ efficient string matching algorithms to calculate the similarity of the melodies. Figure 4 shows the proposed character encoding standard (CES) with which two-dimensional numeric vectors of pitch interval and duration ratio are changed into corresponding characters. More precisely, the two-dimensional space that has the pitch interval on the *X*-axis and the duration ratio on the *Y*-axis is divided into multiple cells of possibly different sizes and then each cell is assigned a unique character. Under this conversion scheme, it is possible for more than one vector located closely in the two-dimensional space to be changed to the same character. This has the effect of tolerating the errors included in humming. Figure 5 shows the result of encoding a series of two-dimensional vectors into a corresponding string using the example of the CES in Figure 4. To achieve high accuracy, the CES must be chosen optimally for the underlying music database. As explained in Section 5.3, we take an experimental approach where we generate a set of CES suitable for the underlying music database and choose the one that has the highest accuracy.

3.2. Significant melody extraction

Among the many melodies that comprise a single piece of music, the significant melody is the one that is able to represent the music. In this paper, significant melodies, which have a high probability of being queried by users, are divided into two types: frequent melodies that occur repetitively within the music and rest-unit melodies that begin after a long rest. Users typically remember frequent melodies and rest-unit melodies more easily than the other melodies. Therefore, we extract both types of melodies from the music and indexes are built from them. Since the process of extracting rest-unit melodies is straightforward, the focus here will be on the process of extracting frequent melodies.

Given a piece of music, we first carry out a string pattern analysis, treating each bar as a single processing unit. That is, initially a set of strings is extracted, each of which represents a bar, and then a string matching algorithm is executed such as the KMP (Knuth–Morris–Pratt) algorithm [17] to count the number of occurrences of each string. Thus, the following data is obtained for each string:

Table 1
Data obtained from the song titled 'After for a long time', a popular Korean ballad, after executing the proposed string pattern analysis

Bar string	**List of bar positions**	**Count *(C)***	**Rank *(R)***
<j, x, k, j, i, x>	18, 22, 35, 39	4	1
<k, k, j, x, g>	19, 36	2	2
<j, x, j, k, k, k>	20, 37	2	2
<k, j, j, w, g>	21, 38	2	2
…	…	…	…
<j, x, g, k, E, x, i, x>	17	1	3

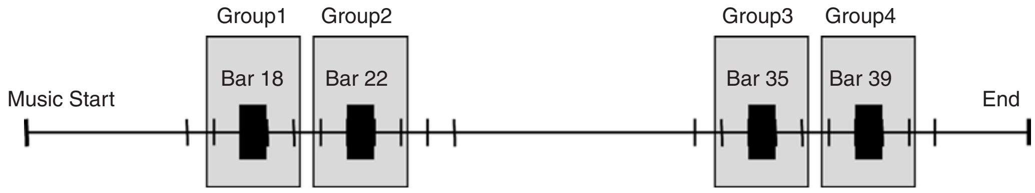

Fig. 6. A set of initial groups constructed from the bars of Table 1 whose rank values are 1.

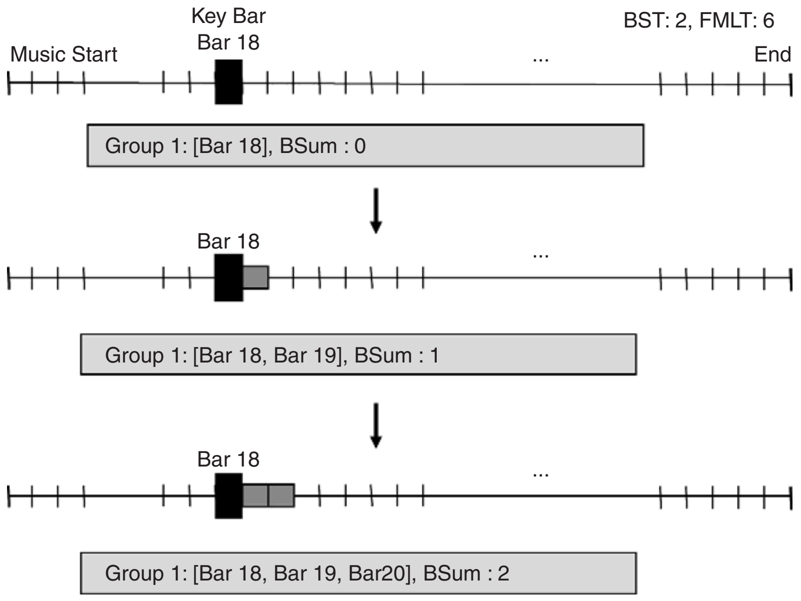
Key Bar
Bar 18
BST: 2, FMLT: 6
Music Start
...
End
Group 1: [Bar 18], BSum : 0
Bar 18
Group 1: [Bar 18, Bar 19], BSum : 1
Bar 18
Group 1: [Bar 18, Bar 19, Bar20], BSum : 2

Fig. 7. First few stages of creating the frequent melody around bar 18.

1. a list of bar positions (*BarId_list*);
2. the number of occurrences of the string (*C*), and the descending order of value C (*R*).

As an example, Table 1 is from the song titled 'After for a long time', which is a popular Korean ballad, after executing the proposed string pattern analysis.

After executing the string pattern analysis, we create the groups of bars. As Figure 6 indicates, each bar whose rank value (i.e. *R*) is 1 produces a separate group and becomes the key bar (KB) of its own group. Therefore, the initial number of groups is the same as the number of bars whose rank values are 1. Next, to construct frequent melodies, each group expands in both forward and backward directions by joining the bars that are adjacent to the group and that have a high *C* value. This joining step requires two parameters, bar selection threshold (BST) and frequent melody length threshold (FMLT). The BST is for allowing only frequent bars to be joined into the groups. That is, in the case that a bar has a *C* value less than the bar selection threshold, it cannot be joined with the group even if it is adjacent to the group. The FMLT is for controlling the length of frequent melodies. Whenever a bar is joined with the group being expanded, the difference from the *R* value of the bar to the *R* value of the key bar of the group is calculated and then added to the variable BSum (i.e. bar sum), whose initial value is 0. If the joining of a bar to a group makes the value of BSum exceed the frequent melody length threshold, the bar is not allowed to join the group.

Based on Table 1, Figure 7 shows the first few stages of creating the frequent melody around bar 18. This example assumes the values of BST and FMLT are 2 and 6, respectively. The first bar with the greatest value of *C*, which is bar 18, becomes the key bar of the first group. Next, both sides of the key bar are considered, which are bar 17 and bar 19, one by one. Since the *C* value of bar 17 is less than the value of BST (i.e. $1 < 2$), bar 17 cannot be joined with the first group and the extension to the left direction cannot be executed further. Consider the joining of bar 19 to the first group. The *C* value of bar 19 is not smaller than the value of BST (i.e. $2 \geq 2$). The difference from the *R* value of bar 19 to the *R* value of the key bar (i.e. bar 18) of the first group is 1 (i.e. $2 - 1 = 1$) and the current value of BSum is 0. If this difference is added to the current value of BSum, 1 will be obtained, which does not exceed the value of FMLT. Since bar 19 satisfies both parameters, BST and FMLT, bar 19 is joined to the first group, the value of BSum is updated to 1, and the extension to the right direction (i.e. bar 20) is continued. Generally, such an extension process ends when progress cannot proceed in either the left or right direction. At this moment, the extent of the group becomes a frequent melody.

As shown in Figure 8, it is possible for a frequent melody to overlap with other frequent melodies. In this case, rather than treating each one of the overlapping frequent melodies separately, they are merged into a single one in order to reduce the storage space. Eventually, as shown in Figure 9, the two frequent melodies from the example of Table 1 are obtained.

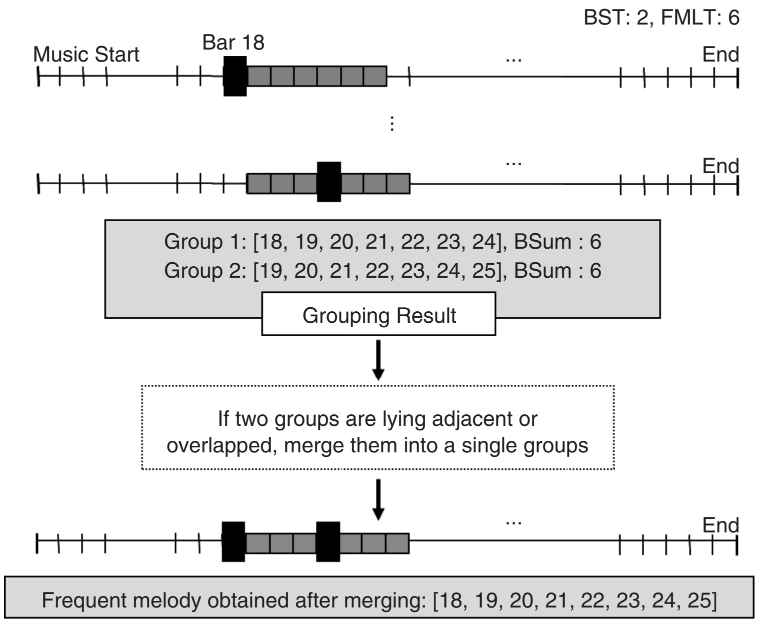
BST: 2, FMLT: 6
Music Start
Bar 18
...
End
Group 1: [18, 19, 20, 21, 22, 23, 24], BSum : 6
Group 2: [19, 20, 21, 22, 23, 24, 25], BSum : 6
Grouping Result
If two groups are lying adjacent or overlapped, merge them into a single groups
Frequent melody obtained after merging: [18, 19, 20, 21, 22, 23, 24, 25]

Fig. 8. Handling of frequent melodies overlapping each other.

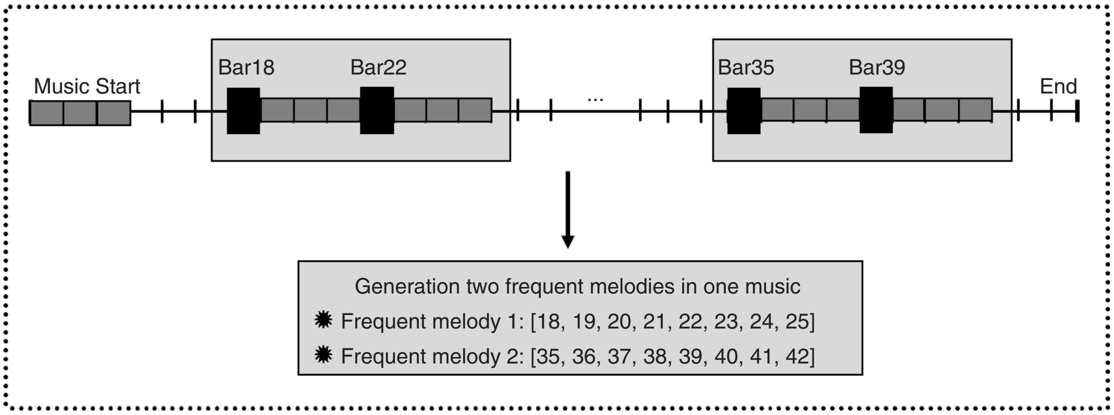
Music Start
Bar18
Bar22
...
Bar35
Bar39
End
Generation two frequent melodies in one music
Frequent melody 1: [18, 19, 20, 21, 22, 23, 24, 25]
Frequent melody 2: [35, 36, 37, 38, 39, 40, 41, 42]

Fig. 9. Frequent melodies obtained finally from the example of Table 1.

3.3. *Indexing*

To speed up the retrieval of the desired music, three suffix trees [18] are built, as shown in Figure 10. The idea of a suffix tree comes from a trie [19]. A trie is an index structure used for indexing sets of keywords of varying sizes. A suffix tree [18] is a trie whose set of keywords comprises the suffixes of a string. Nodes with a single outgoing edge can be collapsed, yielding the structure known as the suffix tree.

First, for each song stored in the database, we retrieve the corresponding encoded string, extract every suffix from the string, and then insert them into the suffix tree named *all music suffix tree index* or *1st index*. The main purpose of this index is to minimize the direct access to the database that may occur when users query melodies other than frequent melodies or melodies

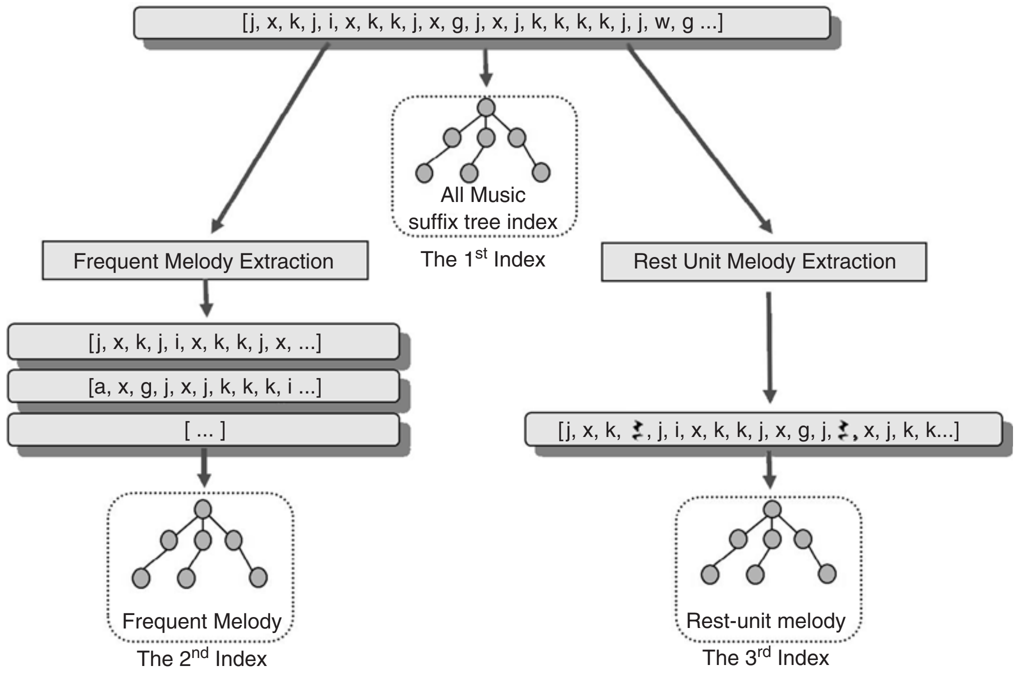
[j, x, k, j, i, x, k, k, j, x, g, j, x, j, k, k, k, k, j, j, w, g ...]
All Music suffix tree index
The 1st Index
Frequent Melody Extraction
Rest Unit Melody Extraction
[j, x, k, j, i, x, k, k, j, x, ...]
[a, x, g, j, x, j, k, k, k, i ...]
[...]
[j, x, k, 𝄽, j, i, x, k, k, j, x, g, j, 𝄽, x, j, k, k...]
Frequent Melody
The 2nd Index
Rest-unit melody
The 3rd Index

Fig. 10. Three suffix trees for speeding up the retrieval of the desired music.

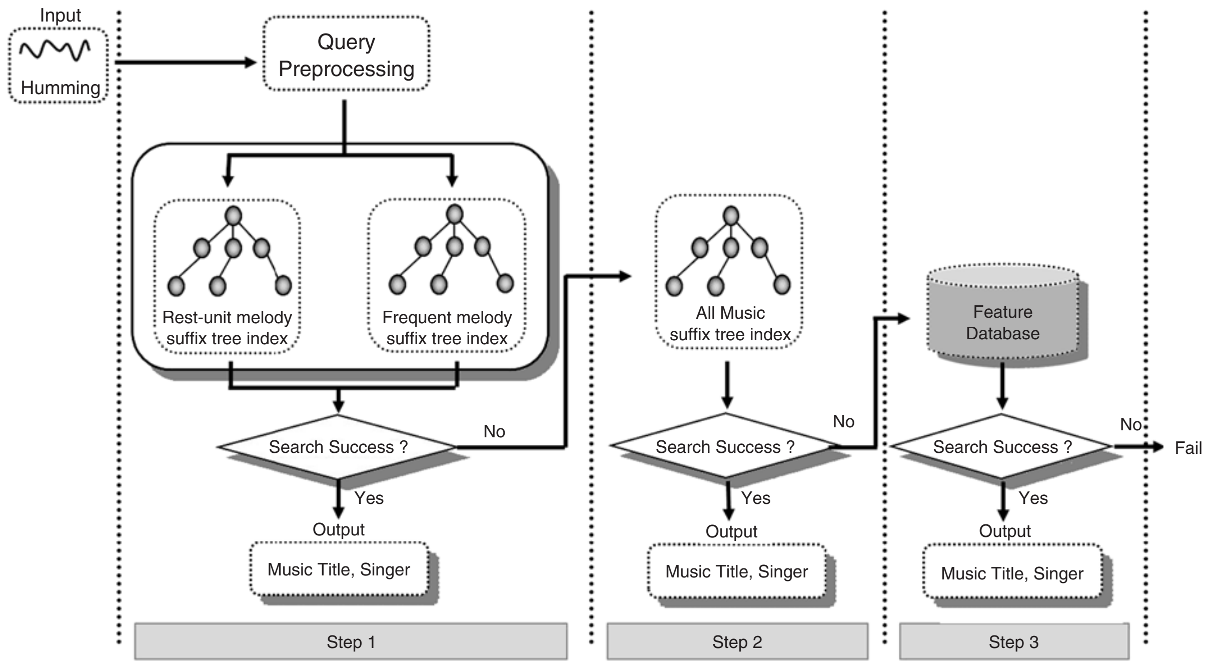

Fig. 11. Query processing with two stages, preprocessing and searching.

starting after a long rest. Next, for each frequent melody discovered in the music database, every suffix is extracted and inserted into the suffix tree named *frequent melody suffix tree index* or *2nd index*. If users query some part of the frequent melodies, the desired music can be quickly retrieved using this index. Finally, for each rest-unit melody that begins after a long rest, every suffix is extracted and inserted into the suffix tree named *rest-unit melody suffix tree index* or *3rd index*. If users query some part of the rest-unit melodies, the desired music can be quickly retrieved using this index.

4. Query processing

Queries hummed by users are processed with two stages, preprocessing and searching. In preprocessing, a query is transformed into a series of two-dimensional feature vectors including the pitch interval and duration ratio, and then into the corresponding string using the same CES employed at the time of index construction.

As shown in Figure 11, there are three steps in the searching stage. In the first step, the frequent melody and rest-unit melody suffix tree indexes are searched in parallel for the string that corresponds to the submitted query. If the query looks for some part of the frequent melodies or rest-unit melodies, the target music will be found during the first searching step. Otherwise, the second step involves searching all music suffix tree indexes for the query string. If the target music is not found during the second searching step, then the third searching step is utilized, in which the Euclidean distance is used as a similarity measure, where the feature database is searched for the series of two-dimensional feature vectors that correspond to the submitted query. Remember that the goal of this query processing approach is to minimize direct access to the database. This goal can be attained by processing as many queries as possible within the first or second searching step.

Table 2
Number of queries whose Euclidean distances to the corresponding original melodies in the music database are 2**i* for each *i* from 1 to 8

Euclidean distance between queries and their corresponding original melodies	Number of queries
2	30
4	30
5	30
6	30
8	30
8	30
10	30
12	30
14	30
16	30

5. Performance evaluation

In this section, we evaluate the proposed QBH system through experiments with real data. We collected 3000 Korean songs to construct a music database and, as shown in Tables 2 and 3, let users hum the songs they wanted to retrieve. As shown in Table 2, for each *i* from 1 to 8, 30 queries were collected whose Euclidean distances to the corresponding original melodies in the music database are 2**i*. Also,

Table 3
For each *i* from 1 to 6, the number of queries that were hummed for 5**i* seconds and their average number of notes

Query time (s)	Average number of notes in queries	Number of queries
5	7	30
10	13	30
15	20	30
20	26	30
25	30	30
30	33	30

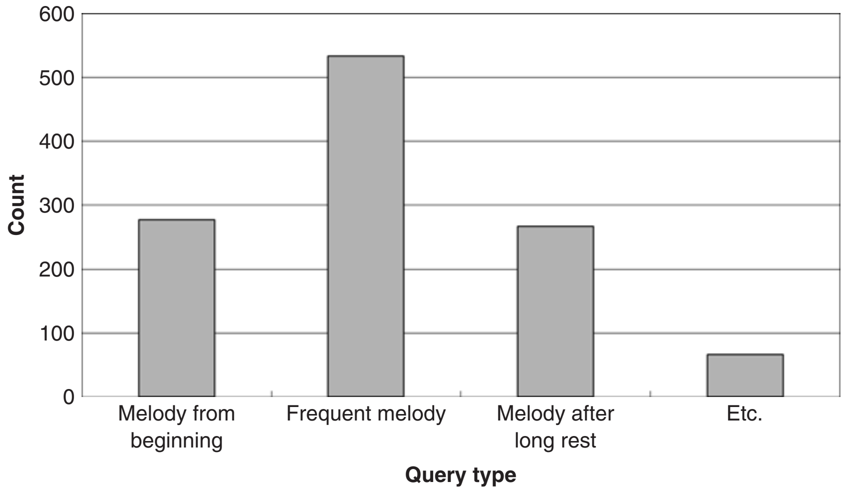

Fig. 12. Number of queries contained in frequent melodies, rest-unit melodies, and melodies starting at the beginning of music, respectively.

as shown in Table 3, for each *i* from 1 to 6, 30 queries were collected that were hummed for 5**i* seconds and the average number of notes contained in the 30 queries was calculated.

When the QBH system was designed, we assumed that the users would query frequent melodies or rest-unit melodies most often. To verify this assumption, we collected 700 queries of real humming from 30 people and then analyzed the queries. Figure 12 indicates that 76% of the queries contained frequent melodies, 38% of the queries contained rest-unit melodies, and 40% of the queries were the beginning parts of music. Note that some melodies can be included in more than one melody type. For example, it is possible for a frequent melody to start at the beginning of music or after a long rest. This analysis gives strong evidence that the assumption is realistic.

5.1. Determination of an optimal value for the bar selection threshold (BST)

Remember that we used two parameters, BST and FMLT, to control the length of the frequent melodies when the QBH system was constructed. In this subsection, we determine the optimal value of the BST by examining the pattern changes in the index size, the search time, and the accuracy of the QBH system according to varying values of BST. The optimal value of FMLT is determined in the next subsection.

To analyze the pattern change in the index size, we observed the total number of characters stored in the frequent melody suffix tree index while changing the BST value from 1 to 4. For this experiment, we set the value of FMLT to 6, generated frequent melodies, and selected from the frequent melodies 30 queries of 15 seconds. The reason for setting the value of FMLT to 6 is that it makes frequent melodies longer than 15 seconds. Figure 13 shows the result of this experiment. The frequent melody suffix tree index is largest when the value of BST is 1, but it becomes smaller as the value of BST increases. The result indicates that frequent melodies that are stored in the index get longer as the value of BST increases.

Next, we examined the pattern change in the amount of search time within the frequent melody suffix tree index, while increasing the value of BST from 1 to 4. Figure 14 shows the result of this experiment. The *X*-axis is the number of songs stored in the database and the *Y*-axis is the search time of the frequent melody suffix tree index. Regardless of the amount of music stored in the database, the search time increases conspicuously as the value of BST gets larger. This result can be interpreted as the growth of the BST value causing the increment of the index size and subsequently the increase in search time.

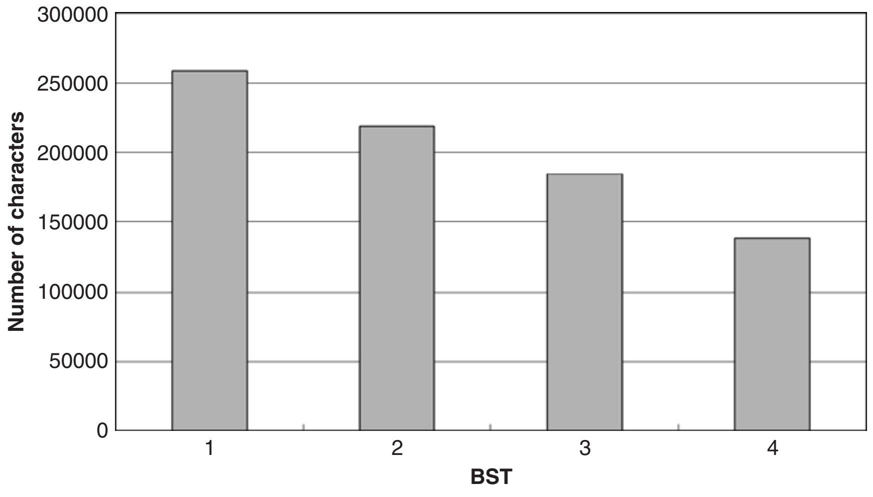

Fig. 13. Total number of characters stored in the frequent melody suffix tree index when the value of BST is 1, 2, 3 and 4, respectively (FMLT = 6, 3000 songs).

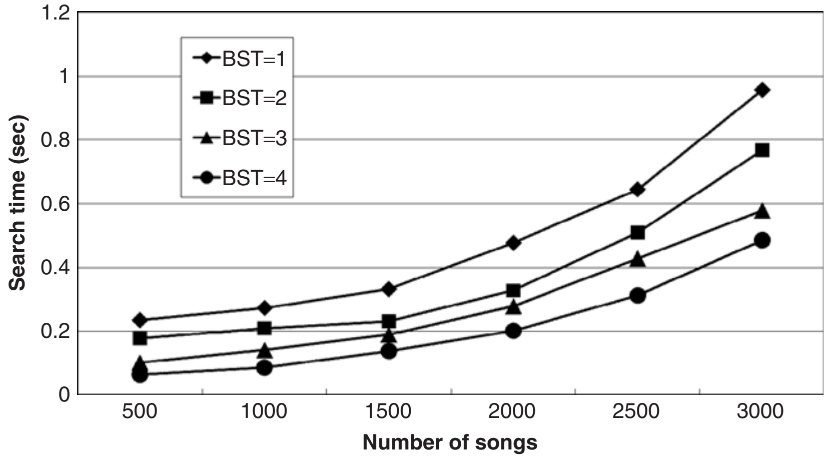

Fig. 14. Search time of the frequent melody suffix tree index when the value of BST is 1, 2, 3 and 4, respectively (FMLT = 6, query length = 15 s).

Lastly, we examined the pattern change accuracy (i.e. recall and precision) of the frequent melody suffix tree index, while increasing the value of BST from 1 to 4. Figures 15 and 16 show the result of this experiment. Both recall and precision get lower as the BST increases. More specifically, both recall and precision get slightly lower when the value of BST changes from 1 to 2, but they decrease abruptly when the value of BST changes to 3 and then to 4. There are two possible reasons for this phenomenon. The first reason is that a significant number of frequent melodies obtained when the value of BST is 3 or 4 are shorter than the hummed queries. In this case, it may not be possible to obtain the target music by searching only the frequent melody suffix tree index. The second reason is that some of the frequent melodies may be unreliable when the value of BST is too large. For example, when the value of BST is 4, users may not get the target music if they enquire a melody that appears three times in the music. Considering the trade-offs among the index size, search time, and accuracy obtained by the above experiments, we can conclude that the optimal value of BST is 2.

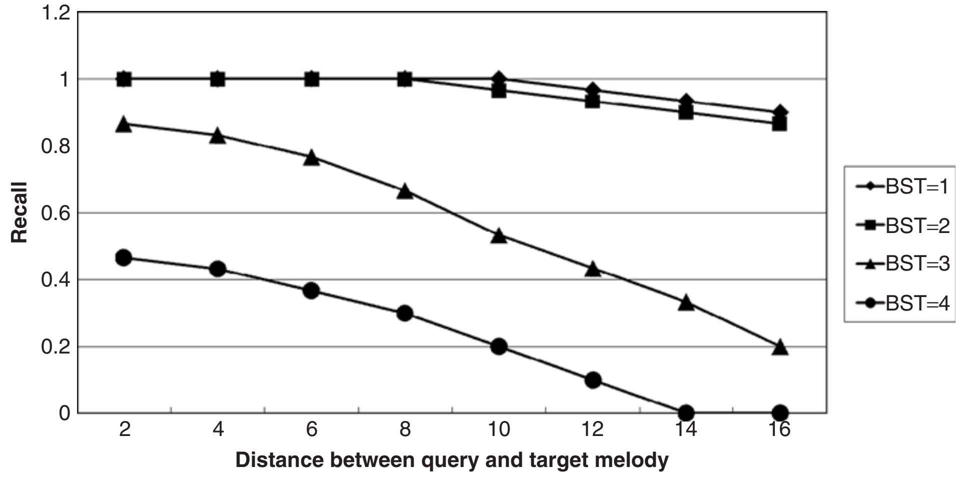

Fig. 15. Recall of the frequent melody suffix tree index when the value of BST is 1, 2, 3 and 4, respectively (FMLT = 6, 3000 songs, query length = 15 s).

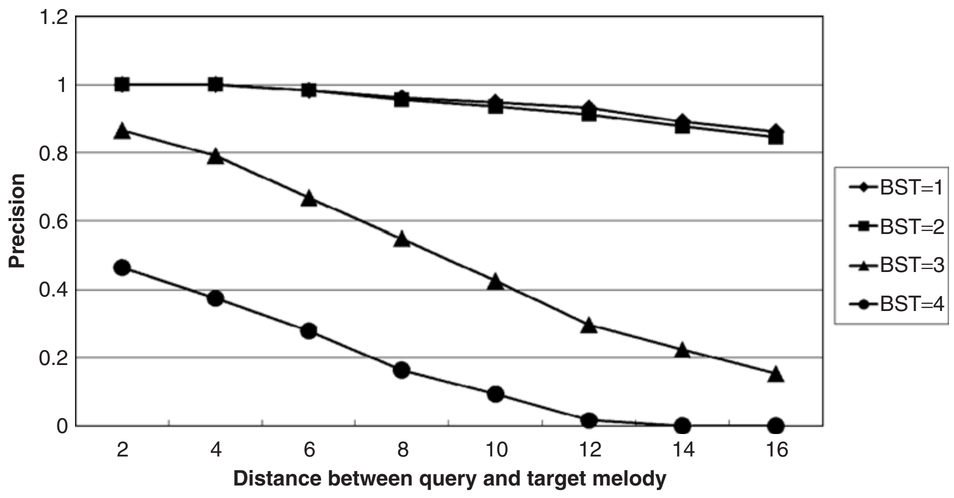

Fig. 16. Precision of the frequent melody suffix tree index when the value of BST is 1, 2, 3 and 4, respectively (FMLT = 6, 3000 songs, query length = 15 s).

5.2. Determination of an optimal value of frequent melody length threshold (FMLT)

In this subsection, we determine the optimal value of FMLT by examining the pattern changes in the index size, search time, and accuracy of the QBH system according to varying values of FMLT. Just like the experiment for determining the optimal value of BST, in this experiment, we constructed a music database with 3000 songs and then extracted queries hummed for 15 s. Using the result of the previous experiment, we set the value of BST to 2 in this experiment. Figure 17 shows the total number of characters stored in the frequent melody suffix tree index when the value of FMLT is 2, 4, 6, 8, and 10, respectively. The figure indicates that the size of the index grows almost linearly as the value of FMLT increases. Figure 18 shows the search time of the frequent melody suffix tree index for each FMLT. Here, the *X*-axis is the number of songs stored in the database and the *Y*-axis is the search time of the frequent melody suffix tree index. The figure indicates that, with any number of songs stored in the database, the search time of the index increases as the value of FMLT grows.

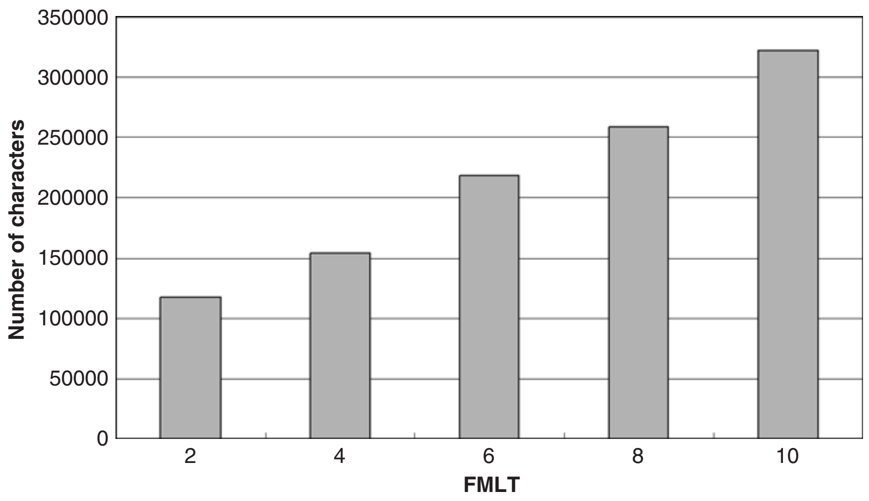

Fig. 17. Total number of characters stored in the frequent melody suffix tree index when the value of FMLT is 2, 4, 6, 8 and 10, respectively (BST = 2, 3000 songs).

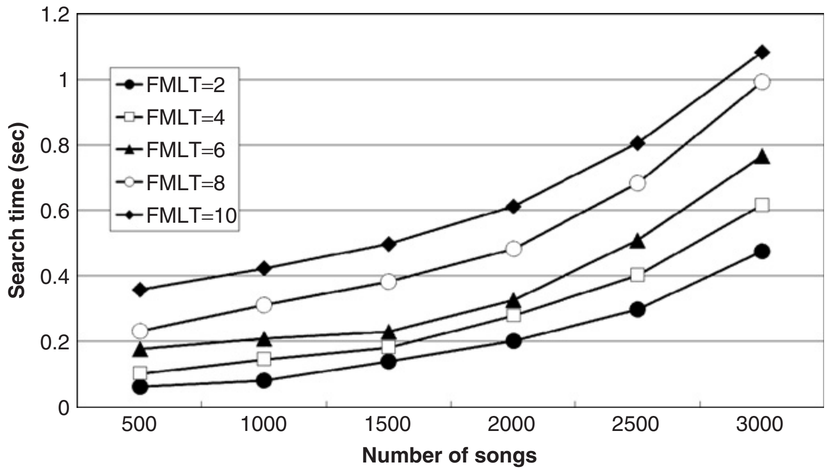

Fig. 18. Search time of the frequent melody suffix tree index with various values of FMLT and various numbers of songs stored in the music database (BST = 2, query length = 15 s).

Next, we examined the pattern change in accuracy (i.e. recall and precision) of the frequent melody suffix tree index, while increasing the value of FMLT from 2 to 10 by increments of 2. Figures 19 and 20 show the result of this experiment. In these two figures, the *X*-axis is the distance between queries and their corresponding target music and the *Y*-axis is the recall and precision, respectively. For any value of FMLT, both recall and precision decrease as the distance between the queries and their corresponding target music increases. Also, for any distances between queries and their corresponding target music, both recall and precision increase as the value of FMLT increases. This result indicates that a significant number of frequent melodies obtained when the value of FMLT is small are shorter than the hummed queries.

Considering the trade-offs among the index size, search time, and accuracy obtained by the above experiments, it is concluded that the optimal FMLT is 6, which leads to the creation of the index that is fairly fast, relatively small, and shows accuracy that is comparable to the indexes with a

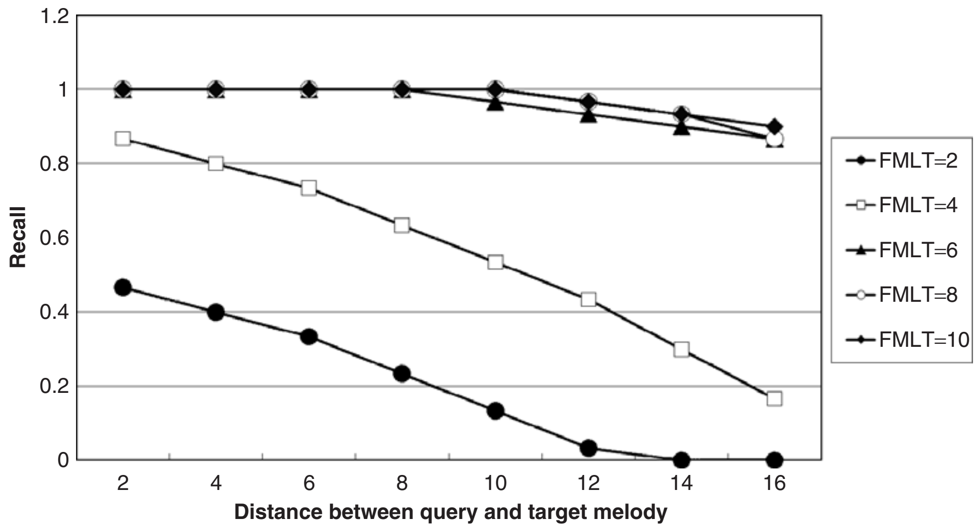

Fig. 19. Recall of the frequent melody suffix tree index with various values of FMLT and various distances between queries and their corresponding target music (BST = 2, 3000 songs, query length = 15 s).

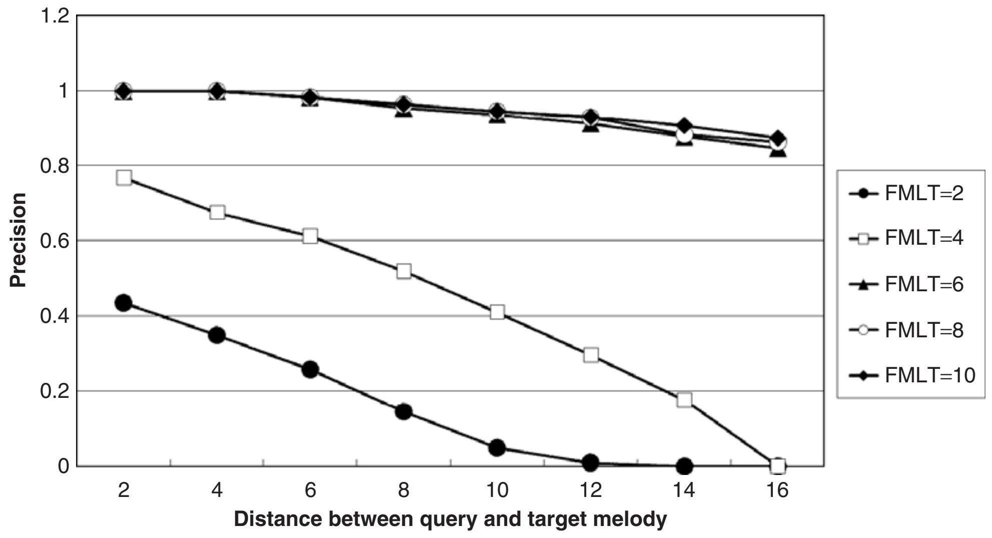

Fig. 20. Precision of the frequent melody suffix tree index with various values of FMLT and various distances between queries and their corresponding target music (BST = 2, 3000 songs, query length = 15 s).

larger value of FMLT (i.e. 8 or 10). Note that the value of FMLT must be determined considering the typical length of queries, in order to avoid the situation where frequent melodies to be stored in the index are shorter than the queries.

5.3. Selection of an optimal character encoding standard

The accuracy of the QBH system is highly affected by the CES employed when converting two-dimensional vectors of feature data to their corresponding characters. Therefore, in this section, the experimental approach to determine the optimal standard for character encoding is explained. For this experiment, as shown in Figure 21, we generated four types of CES by dividing each axis of the two-dimensional space with non-overlapping ranges of possibly different sizes. Then the QBH system was constructed with each one of the four CESs and its accuracy evaluated. According to the result of previous experiments, we set the value of the BST and FMLT to 2 and 6, respectively, in this experiment.

Figure 22 compares the recall of four character encoding standards: ‘Range 1’, ‘Range 2’, ‘Range 3’, and ‘Range 4’, while changing the distances between the queries and their corresponding target music. The recall of all four standards decreases gradually as the distances between the queries and their target music increase. This figure also indicates that the recall of Range 4 is higher than the other three standards. This result can be interpreted as follows:

1 the cells in the middle area, where lots of feature data are positioned, of Range 4 are bigger than those of other standards,

2 more feature data in the middle area are assigned the same character, and therefore,

3 even though errors are contained in humming, it is more likely for users to obtain the target music.

On the contrary, the recall of Range 1 is lower than the other three methods. The reason for this is that the cells in the middle area of Range 1 are smaller than those of other standards and therefore the accuracy of the query result is very sensitive to the errors contained in humming.

Figure 23 compares the precision of four character encoding standards, while changing the distances between the queries and their corresponding target music. The precision of all four standards falls noticeably as the distances between the queries and their target music increase. This is because wrong answers are retrieved more often as the distances between the queries and their target music increase. Note that the precision of Range 4 drops most rapidly because the cells in its middle area

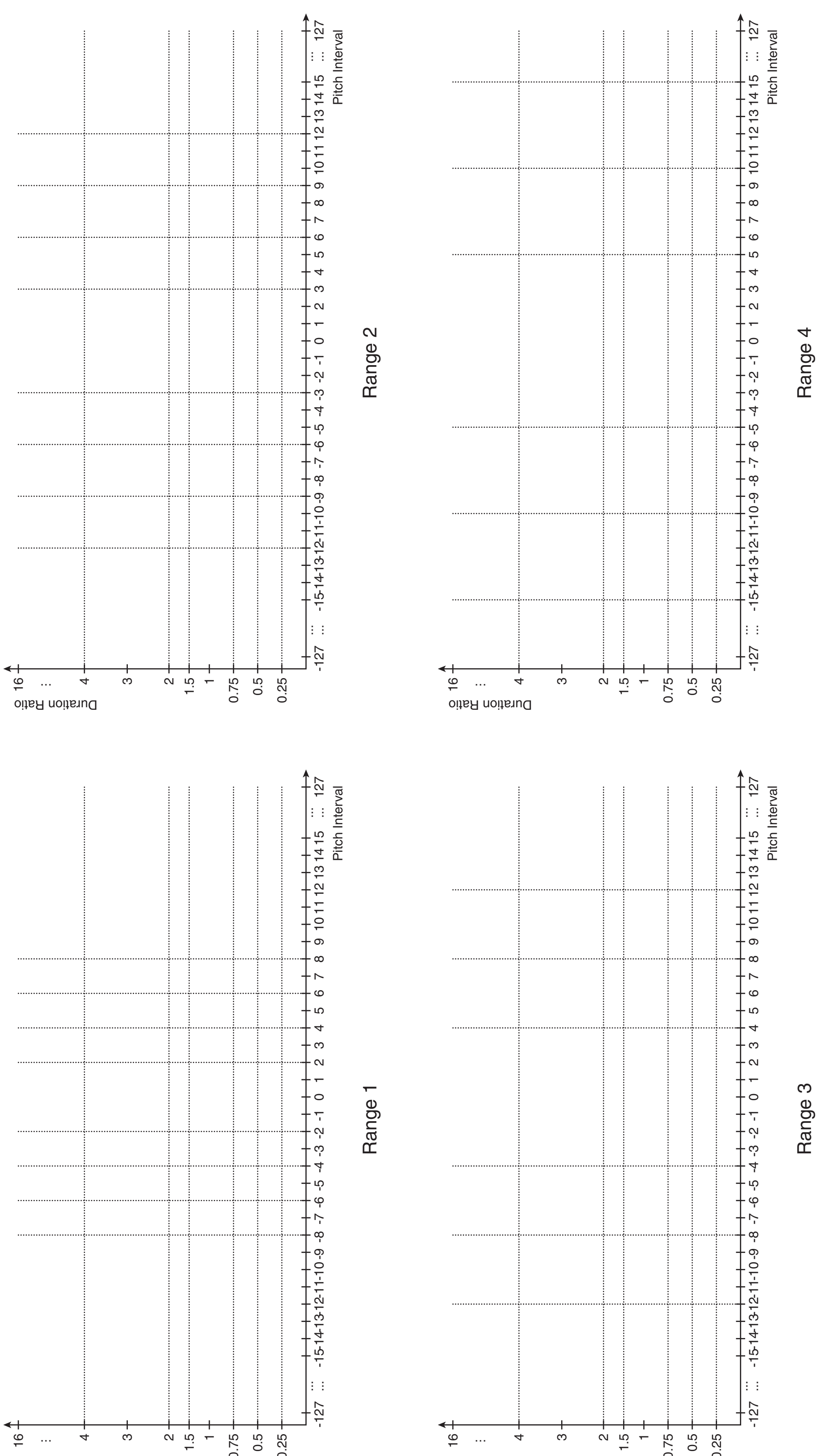

Fig. 21. Four types of character encoding standard.

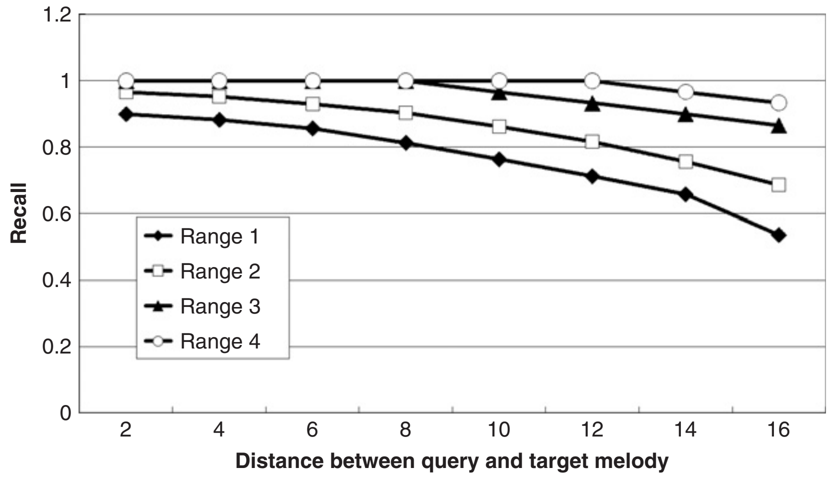

Fig. 22. Recall of four character encoding standards with increasing distance between the queries and their corresponding target music (BST = 2, FMLT = 6, 3000 songs, query length = 15 s).

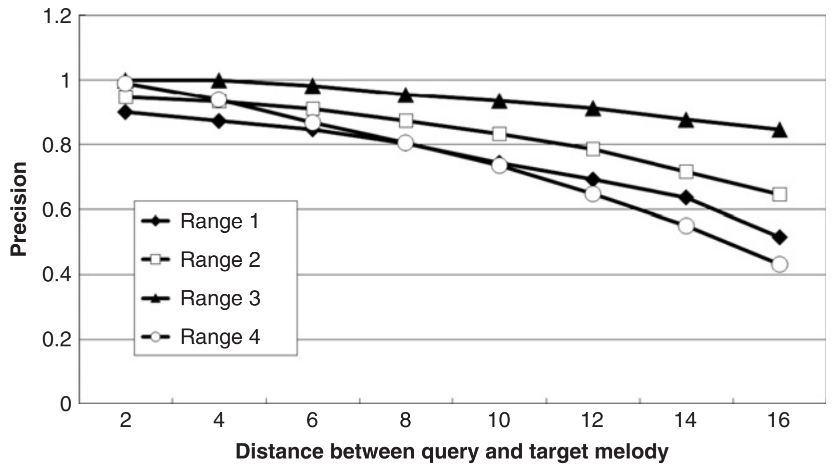

Fig. 23. Precision of four character encoding standards with increasing distance between the queries and their corresponding target music (BST = 2, FMLT = 6, 3000 songs, query length = 15 s).

are largest. Considering the trade-offs between the pattern changes in recall and precision, Range 3 was selected as the optimal standard of character encoding.

5.4. Comparison with the N-gram method

To evaluate the performance improvements of the proposed QBH system, we compared the search time and accuracy of the proposed system with those of the N-gram method. As shown in Table 4, the methods involved in this comparison are

1. a naive DB scan that searches only the feature database without looking up any indexes;
2. the two-step search method that first searches all of the music suffix tree index and then, if necessary, searches the feature database;

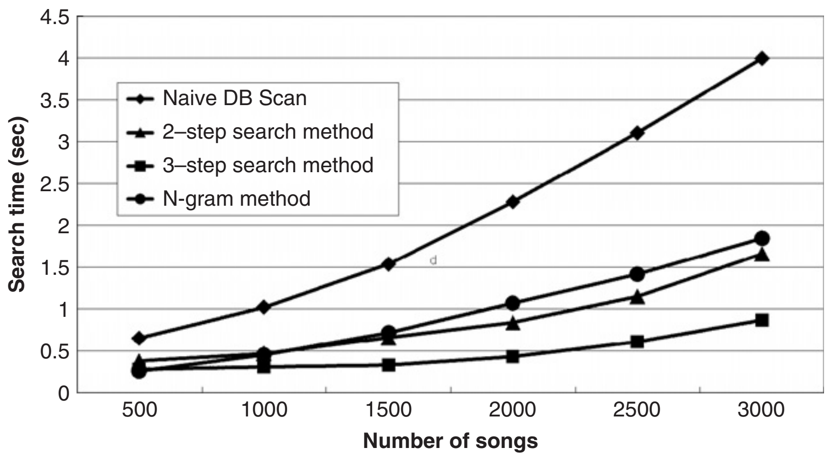

Fig. 24. Search time of four methods with increasing number of songs contained in music database (BST = 2, FMLT = 6, query length = 15 s).

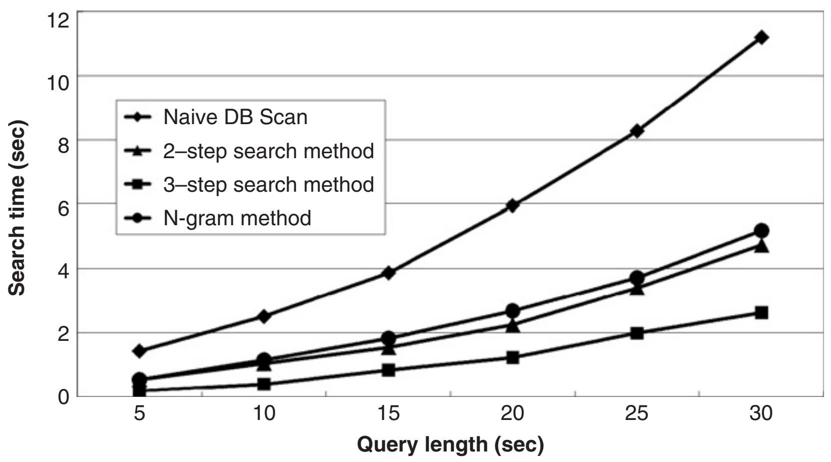

Fig. 25. Search time of four methods with increasing length of queries from 5 s to 30 s (BST = 2, FMLT = 6, 3000 songs).

3. the three-step search method that first searches the frequent melody suffix tree index and rest-unit melody suffix tree index in parallel and then, if necessary, searches all of the music suffix tree index and lastly, if necessary, searches the feature database; and
4. the N-gram method.

For this experiment, a music database was constructed with up to 3000 songs; and 300 queries that last for up to 15 s, and whose distances to the corresponding target music are 4, were collected. The standard of character encoding was Range 4 in Figure 21, and the values of BST and FMLT were 2 and 6, respectively.

Figure 24 compares the search time of the four methods with an increasing number of songs, from 500 to 3000 by increments of 500, contained in the music database. It is clear that the search time of all of the methods increases as the size of the database increases. However, the three-step search method consistently takes the least amount of search time irrespective of the database size. The reason for this

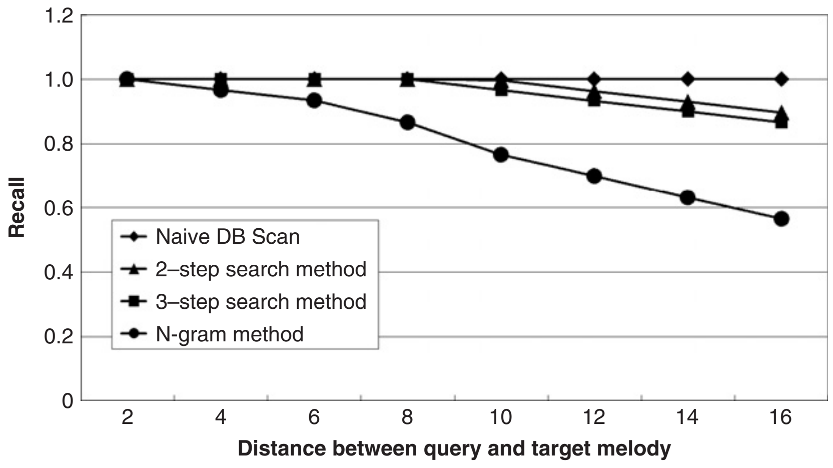

Fig. 26. Recall of four methods with increasing distance between queries and their corresponding target music (BST = 2, FMLT = 6, 3000 songs, query length = 15 s).

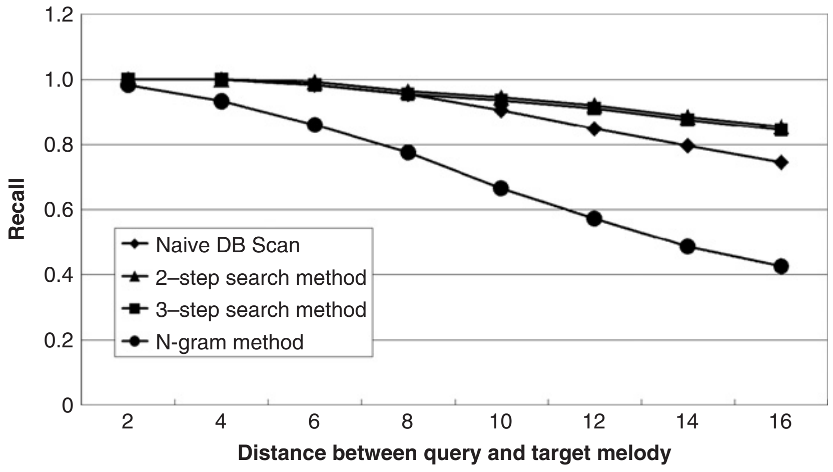

Fig. 27. Precision of four methods with increasing distance between queries and their corresponding target music (BST = 2, FMLT = 6, 3000 songs, query length = 15 s).

result is that a large portion of the queries are processed by just searching the frequent melody suffix tree index or the rest-unit melody suffix tree index. Since the N-gram method has to sequentially examine the strings stored in the music database, it took about twice as long as the three-step search method on average.

Figure 25 compares the search time of the four methods with increasing length of queries from 5 to 30 s. Just like the result of the previous experiment, the search time of all of the methods increases as the length of the queries increases and the three-step search method takes the least amount of search time with any query length.

Next, we compared the recall and precision of the four methods while increasing the distance between the queries and their corresponding target music. Figures 26 and 27 show the result. As clearly shown in these two figures, the three-step search method produces a more accurate result compared to the N-gram method with any distance between the queries and their corresponding target music.

6. Conclusion

Query-By-Humming is one of the content-based method studies in the area of music information retrieval. Since QBH systems are capable of retrieving the desired music even when the users' humming is inaccurate, they are useful for people who are not talented singers or who have a speech impediment.

In this paper, we proposed an efficient QBH system. Considering the characteristics of humming, the feature data included the pitch interval and the duration ratio between adjacent notes. Since the operation for calculating the distance of melodies of numeric type is expensive, we converted the melodies of numeric type into the corresponding strings with the consideration of errors involved in humming. The strings were then compared using existing string matching algorithms. For an additional improvement in search speed, significant melodies were extracted, which have a high chance of being queried, and then a couple of indexes were built from them. The definition of significant melodies is divided into two types in this paper:

1. the most frequent melody that appears in a single piece of music; and
2. the rest-unit melody that begins after a long rest within the music;

In addition, it was suggested that a three-step index search method be used for minimizing database access and obtaining query answers as early as possible. Through experiments with real-world data, it was verified that users are more likely to look for the desired music by humming some parts of frequent melodies or rest-unit melodies and the proposed QBH system is much faster and more accurate than the N-gram method.

As a further study, it is planned to devise a systematic tool to find an optimal standard for character encoding and optimal values of two parameters, the bar selection threshold (BST) and the frequent melody length threshold (FMLT). Compression of the suffix trees employed in the proposed QBH system is another direction of further research.

Acknowledgements

This research was accomplished as the result of the promotion project for the Culture Contents Technology Research Center supported by Korea Culture and Content Agency (KOCCA).

References

[1] A. Uitdenbogard and J. Zobel, Melodic matching techniques for large music databases. In: *Proceedings of the 7th ACM International Conference on Multimedia 1999* (ACM, Orlando, 1999) 57–66.

[2] P. Rolland, G. Raskinis and J. Ganascia, Musical content-based retrieval: an overview of the Melodiscov approach and system. In: *Proceedings of the 7th ACM International Conference on Multimedia 1999* (ACM, Orlando, 1999) 81–84.

[3] A. Lippincott, Issues in content-based music information retrieval, *Journal of Information Science* 28(2)137–42.

[4] A. Ghias, J. Logan and D. Chamberlin, Query By Humming. In: *Proceedings of the 3rd ACM International Conference on Multimedia 1995* (ACM, San Francisco, 1995) 231–6.

[5] S. Pauws, CubyHum: a fully operational query by humming system. In: M. Fingerhut (ed.), *Proceedings of the 3rd International Conference on Music Information Retrieval*, *ISMIR 2002* (IRCAM, Paris, 2002) 187–96.

[6] S. Doraisamy and S. Ruger, Robust polyphonic music retrieval with N-grams, *Journal of Intelligent Information Systems* 21(1) (2002) 53–70.

[7] R.L. Kline and E.P. Glinert, Approximate matching algorithms for music information retrieval using vocal input. In: *Proceedings of the 11th ACM International Conference on Multimedia 2003* (ACM, Berkeley, 2003) 130–39.

[8] R.B. Dannenberg, W.P. Birmingham, G. Tzanetakis, C. Meek, N. Hu and B. Pardo, The Musart testbed for query-by-humming evaluation. In: H.H. Hoos and D. Bainbridge (eds), *Proceedings of the 4th International Conference on Music Information Retrieval* (Johns Hopkins, Baltimore, 2003) 41–50.

[9] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro and K. Kushima, A practical query-by-humming system for a large music database. In: *Proceedings of the 8th ACM International Conference on Multimedia* (ACM, Los Angeles, 2000) 333–42.

[10] E. Scheirer, Tempo and beat analysis of acoustic musical signals, *Journal of the Acoustic Society of America* 103(1) (1998) 588–601.

[11] A. Pienimaki, Indexing music databases using automatic extraction of frequent phrases. In: M. Fingerhut (ed.), *Proceedings of the 3rd International Conference on Music Information Retrieval* (IRCAM, Paris, 2002) 25–30.

[12] J. Downie, *Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic N-Grams as Text*. Thesis, University of Western Ontario, 1999.

[13] A. Guttman, R-trees: a dynamic index structure for spatial searching. In: B. Yormark (ed.) *Proceedings of the ACM SIGMOD 1984* (ACM, Boston, 1984) 47–57.

[14] K. Andreas, Themefinder: a web-based melodic search tool, melodic similarity: concepts, procedures and applications, *Computing in Musicology* 11 (1998) 231–6.

[15] C.C. Liu, J. Hsu and A.L.P. Chen, Efficient theme and non-trivial repeating pattern discovering in music databases, *Proceedings of the 15th International Conference on Data Engineering 1999* (IEEE, Sydney, 1999) 14–21.

[16] S. Rho, S. Lee and E. Hwang, An efficient audio retrieval scheme in large audio databases, *Journal of SIGDB* 19(3) (2003) 46–57.

[17] D.E. Knuth, J.H. Morris and V.B. Pratt, Fast pattern matching in strings, *SIAM Journal On Computing* 6(2) (1977) 323–50.

[18] G.A. Stephen, *String Searching Algorithms* (World Scientific Publishing, London, 1994).

[19] E.H. Fredkin, Trie memory, *Communications of the ACM* 3(9) (1960) 490–500.