Content-Based Retrieval for Music Collections

Yuen-Hsien Tseng
Dept. of Library & Information Science
Fu Jen Catholic University
tseng@blue.lins.fju.edu.tw

Abstract

A content-based retrieval model for tackling the mismatch problems specific to music data is proposed and implemented. The system uses a pitch profile encoding for queries in any key and an n-note indexing method for approximate matching in sub-linear time. A distinct function that extracts key melodies for query suggestion is developed. The Web-based system provides flexible user interface for query formulation and result browsing. Users can search the system by a short sequence of notes, by uploading a file created by singing, or by clicking suggested key melodies without input. Experiments show that the pitch profile encoding and a 3-note indexing are able to overcome the key mismatch problem and the random errors caused by pitch error, note deletion and insertion. The use of extracted key melodies improves performance over direct search of the music database. For the type of burst mismatch, a query expansion approach is applied.

Keywords: Music retrieval, key melody extraction, pitch profile encoding, music indexing, approximate string matching, query suggestion

1. Introduction

The problem of music retrieval based on surrogates of music, such as titles, composers, or subject classification, has been solved by many of the existing retrieval systems. In contrast, retrieval of a piece of music based on the music contents, especially based on an incomplete, imperfect recall of a fragment of the music, has not yet fully explored. The desire for retrieving music data by contents may arise in several situations. A composer may like to know if a melody occurs to him/her has already appeared previously. A student may like to analyze music data by associating similar content features. A casual user may like to recall the name of the song when most of that music slips from his/her memory except a few memorable melodies.

Although content-based music retrieval is interesting, there are several tasks to be overcome before a pragmatic retrieval system can be finished. First, since melodies are recognizable regardless of what key they are sung in, the system must allow users to input notes in any key. Second, users may submit, either by singing or by keyboard input, an imperfect music fragment not exactly match the one in the music collection. This is a situation similar to the vocabulary mismatch problem in text retrieval where users' queries contain terms that do not match the terms used to index the relevant documents. Therefore some approximate match functions for occasional pitch error and random note deletion and insertion are desirable. Also those means in text retrieval to reduce the vocabulary mismatch such as query term suggestion or relevance feedback would be helpful. There are still other types of vocabulary mismatch specific to music data, such as consolidation and fragmentation [1]. In consolidation, a sequence of notes is combined into one whose duration is their sum and whose pitch is their average. The reverse of such a case is the fragmentation. In some circumstances that a series of continuous consolidations or fragmentation occurs, the mismatch problem will lead to a worse case than that from the random note insertion or deletion. In addition, music data are not easily perceived visually and expressed literally, especially for those with very few music training. Any facilities for efficient input, browsing, and selection would help in query formulation and result inspection.

Previous works on content-based music retrieval have focused on one of the above problems or another. Pfeiffer et al. [2] developed a set of tools which extracts fundamental frequencies as music features and which uses correlation for matching to identify occurrence of commercial music in a query-by-example approach. Wold et al. [3] proposed an approach to classify sounds for similarity search based on acoustical features consisting of loudness, pitch, brightness, bandwidth, and harmonicity. Although these features can be computed from the waveform representation, most of them do not bear the music contents familiar to most users. Foote [4] also took the approach to extract music features from signals. For similarity matching, he used a tree-structure vector quantizer for reducing computation. All the above approaches have avoided the query problem somewhat by using audio examples as the query.

Another lines of researches are to pursue a more straightforward way with the approaches of query by playing, humming, or singing. Hawley [5] developed a system that allows a note sequence being entered via a MIDI keyboard. It then searches the tunes whose beginnings exactly matched the input. Ghias et al. [6] developed a system that allows input from a microphone. This is followed by a process that converts the input into a melodic contour of only three symbols for key-invariance. The contour is then matched against a collection of 183 songs based on an approximate string matching algorithm which allows for note transposition, deletion, and insertion. The limitations of the system are the time-consuming string matching and the lack of accuracy due to the relatively few symbols in representing the melodies. McNab et al. [7] developed a similar application also with acoustic input, melody transcription, pitch encoding, and approximate string matching. This Web-based application provides a number of match modes including approximate matching for "interval and rhythm" and "contour and rhythm". The approximate matching on 9400 songs based on dynamic programming takes 21 seconds. Although a faster state matching algorithm [8] has been implemented, it does not discriminate as well as dynamic

SIGIR '99 8/99 Berkley, CA USA

programming due to the different distance measure of the algorithm. Chen et al. [9] also developed a system allows for acoustic input and key-invariant encoding, but the Hamming distance-like measure is inadequate for the needs of a music database. As there exits software in the Internet that transcribes acoustic input to melody strings (for example, see [10]), our focus on this topic is to develop an integrated system for efficient and effective music retrieval.

In this paper, a content-based music retrieval model aimed at tackling the foregoing problems is presented. A pitch profile encoding is used for allowing queries in any key levels and an n-note indexing method is adopted for approximate matching in sub-linear time. The most distinct feature of this system is the key melody extraction module that extracts representative and memorable melodies from the music collection for query suggestion and effective retrieval. Users can search the key melodies with any pieces of familiar fragments occur to them. Relevant key melodies can be matched at a lower score without severely interfered by other irrelevant ones due to the relatively condensed musical data. The relevant key melodies can then be easily identified and used to pull out their corresponding tunes from the music collection. The overall effect is that the response time and search failure are both reduced even with more discrepancy between queries and the music data.

The system is developed on the Web and provides flexible user interface for query formulation and result browsing. Several input methods can be used to make query in this system. The system can randomly generate a set of key melodies for users' browsing, listening, and selection such that making queries without any musical data input is possible. Users can also choose to input a short sequence of notes with simplified notation to quickly focus on the pieces of music they are interesting. Another option is to make a query by uploading a file created locally by singing or humming with some melody transcription software available in the Internet. In any way, the system responses a playback button, a set of automatically expanded query candidates, and a result set based on the current query. The playback button allows users to hear what they enter, the query candidates are ready to be submitted in case of search failure for the current query, and each of the key melodies or music pieces present in the result set can be played and selected for further searching.

The rest of the paper is organized as follows. Section 2 presents the key melody extraction algorithm. Section 3 describes the proposed music retrieval model, where the encoding, indexing, retrieval models, and user interface are discussed. Section 4 reports our experiments in evaluating several retrieval approaches. Finally Section 5 concludes this paper and presents some future work to be finished.

2. Key Melody Extraction

Key melodies are the representative fragments of music. They may be the themes or other memorable parts that people may easily recall once they heard the name of the song or part of it. Thus it is worthwhile to extract them to meet possibly most users' queries and at the same time to reduce the response time as matching against them requires less computation. Moreover, they can be used as the abstracts of the songs and hence speed up (at least in a network environment) the process of "browsing" and selection, an important design in interactive retrieval systems.

As Hsu et al [11] pointed out, thematic features of music such as melodies, rhythms, and chords can be represented in string form. Based on this representation, they have proposed an algorithm to extract repeated patterns as key music features. Their algorithm uses a correlative matrix and has time complexity of $O(n^2)$.

Repetition is one of the basic composing rules [12] and is thus a clue for automatic key melody extraction. Repetition also occurs in text documents: documents concentrating on a topic tend to mention a set of words in a specific sequence a number of times. Based on this assumption, Tseng [13] has proposed a fast algorithm for multilingual keyword (or key-phase) extraction. This algorithm has some distinct features: it requires no extra resources such as lexicons, corpora, training data, or NLP parsers; the threshold of term frequency, the only parameter in this algorithm, is easily tuned (usually set to value 1 or 2); key phrases of any length can be identified; when used in character level, single words or word stems can be identified as well as multiple-word phrases; the accuracy rate is 90% for bibliographic data and 86% for full-text news articles [14]. These features meet the requirements for our application in the key melody extraction. Although the algorithm can be applied, some minor modifications are needed. The original algorithm may extract patterns not exist in the original text when the alphabet in the input stream contains too few symbols. Although this case is not likely to happen in ordinary text, this could happen in the melody strings. A simple remedy is to insert special tokens to separate those intermediate repeating patterns that are not adjacent in the original input string. Another modification is to omit the stop lists or stop rules for filtering illegal terms. Although the extracted melody may start and end anywhere in a piece of music, there are no "illegal strings" as those in the text case. Hence, there is no need to filter out any such melodies. The modified version of the algorithm is shown as follows.

1. Convert the input string into a *LIST* of 2-note sequence.
2. Do Loop
2.1 Set *MergeList* to *empty*.
2.2 Put a *separator* to the end of *LIST* as a sentinel and set the occurring frequency of the *separator* to 0.
2.3 For each adjacent sequences *K1* and *K2* in *LIST*, do
    If *K1* is the *separator*, Go to Step 2.3.
    If *K1* and *K2* are mergeable and both of their occurring frequencies are greater than a *threshold*, then
        Merge *K1* and *K2* into *K* and push *K* into *MergeList*.
        Accumulate the occurring frequency of *K*.
    Else
        If the occurring frequency of *K1* is greater than a threshold and *K1* did not merge with the term that precedes in *LIST*, then
            Put *K1* in *FinalList*.
        If the last element of *MergeList* is not the *separator*, then
            Push the *separator* into *MergeList*.
2.4 Set *LIST* to *MergeList*.
Until *LIST* is *empty*.
3. Filter the *FinalList* and sort the result according to some criteria.

Table 1 is an execution example of the algorithm. After the first iteration in step 2, only the last item in list L is put in the final list because it does not merge with others and it occurs more than the threshold frequency. All the other items in list L are discarded because they merge with others or they occur so rarely that they are assume to be of no importance. Similar cases happen in the second iteration, where item CCC

is put in the final list and all other separators are discarded. To show the effect of the separator, consider the input string : "**CCC**D**CCC**". If the separator is not inserted and handled properly, the list L = (CC:4, CC:4, CD:1, DC:1, CC:4, CC:4) would be merged into L1 = (CCC:2, CCC:2) after iteration 1, and then L2 = (CCCC:1) after iteration 2. The items CCC in L1 are not put in the final list. Instead, since they are adjacent in L1, they are merged into L2, resulting in a pattern not present in the original string.

Assume input string = " **CCC**D**CCC**E**CC**", the threshold = 1 and the separator = X. Step 1: create a list of 2-note L =(CC:5, CC:5, CD:1, DC:1, CC:5, CC:5, CE:1, EC:1, CC:5) Step 2: merging After 1st iteration: merge L into L1=(CCC:2, X:0, CCC:2, X:0) drop : (CC:5, CC:5, CD:1, DC:1, CC:5, CC:5, CE:1, EC:1) *FinalList* : (CC:5) After 2nd iteration: merge L1 into L2 = (X:0) drop : (X:0, X:0) *FinalList* : (CC:5, CCC:2) After 3rd iteration: merge L2 into L3 = ( ) drop : (X:0) *FinalList* : (CC:5, CCC:2) Step 3 : filtering : the item CC may be removed since it is a substring of the other.

Table 1. An execution example of the algorithm. The number following the semicolon is the occurring frequency.

The algorithm consists of three major steps. The first step requires only linear-time complexity for converting the input string into a list. The second step involves condition testing and merging of the note sequences. This process is repeated until no element remained to be merged. Because it loops *m* times to produce a melody string of length *m*, the total number of testing and merging is bounded by *mn*, where *n* is the length of the input string and *m* is the length of the longest melody (substring). For each testing, the term frequency is looked up to determine whether to discard the term or not. Since looking up a term frequency through hashing requires only constant time on average [14], the second step is bounded by O(*mn*) on average. The complexity of the last step depends on *m*, the length of the longest string extracted and *k*, the number of strings in the final list. Since the extracted melodies are used for matching the users' queries, it would be advantageous to retain only those distinct longest ones. This would reduce the number of extracted melodies without lost of any information. An operation like this would require sorting on the extracted melodies and comparing the short ones to the long ones. This would be bounded by $O(mk^2)$, if a linear-time string matching algorithm is used. Since *k* is normally far smaller than *n*, the total complexity of this algorithm is bounded by O(*mn*) on average.

The extraction algorithm is implemented in Perl [15], for Perl provides convenient manipulators for strings, lists and hashes. Two sets of MIDI (Musical Instruments Digital Interface) files are tested. For 1052 melody strings derived from each track of 135 Chinese pop songs, the average length is 746 notes and the average longest repeated melodies contains 149 notes, a ratio of 149/746=20% (the average over all ratios is 23%). The extraction time for 70% pop melody strings is within 4 seconds and 80% within 9 seconds. For 203 melody strings derived from each track of 30 classic music, the average length is 973 notes and the average longest repeated melodies contains 135 notes, a ratio of 135/973=18% (the average over all ratios is 14%). The extraction time for 70% classic melody strings is within 4 seconds and 80% within 10 seconds. The above statistics show no major different between these two types of music.

3. A Music Retrieval Model

The architecture of the developed system for music retrieval is shown in Figure 1. Music in MIDI format is collected from the Internet. Usually a MIDI file consists of several tracks, each track records the score information for an instrument. From these files, melodies, rhythms, and chords information can be extracted. Since melodies bear major contents of music, only they are extracted at the current stage.

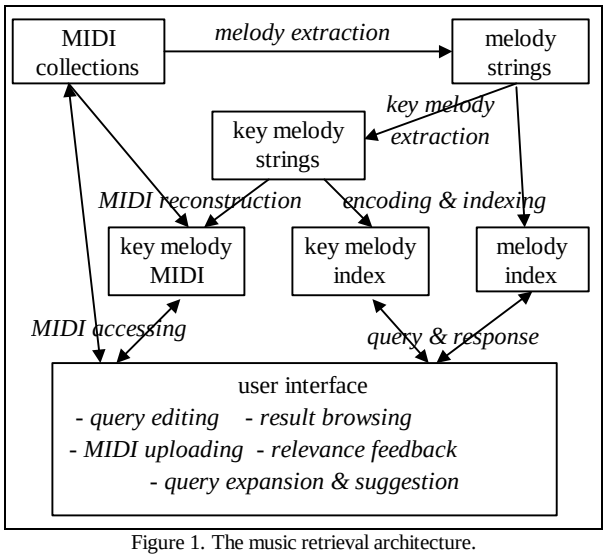

Figure 1. The music retrieval architecture.

The melody strings can then be analyzed by the above extraction algorithm to get repeated patterns. The repeated patterns are matched against the original MIDI files to reconstruct back the original tunes in MIDI format. These tunes consist of another MIDI collection to be played on users' request at the process of result browsing and selection.

Before the melody strings are indexed, they are encoded to allow input in any key. We use symbols *Pn*, *Mn*, and *R* for such encoding, where *P* stands for an ascending note compared to the note that precedes, *M* a descending note, *n* the number of semitones ascending or descending, and *R* a repeating note. The first note is retained for restoring the original melody. So "E4 E4 E4 E4 G4 C4 D4 E4" in standard pitch name in the song "Jingle Bells" is encoded into " *E4 R R R P3 M7 P2 P3*". It can be verified that *m* random errors caused by pitch error or note deletion and insertion in the original string will result in at most *2m* errors in the encoded string, an error ratio of 2. This will require a more error-tolerant retrieval model for the encoded strings to maintain the same level of performance.

Usually MIDI files contain more than score information. They also hold other text data, such as titles, composers, track names, or other description, provided by their creators. To be able to exploit such information, the data extracted from the

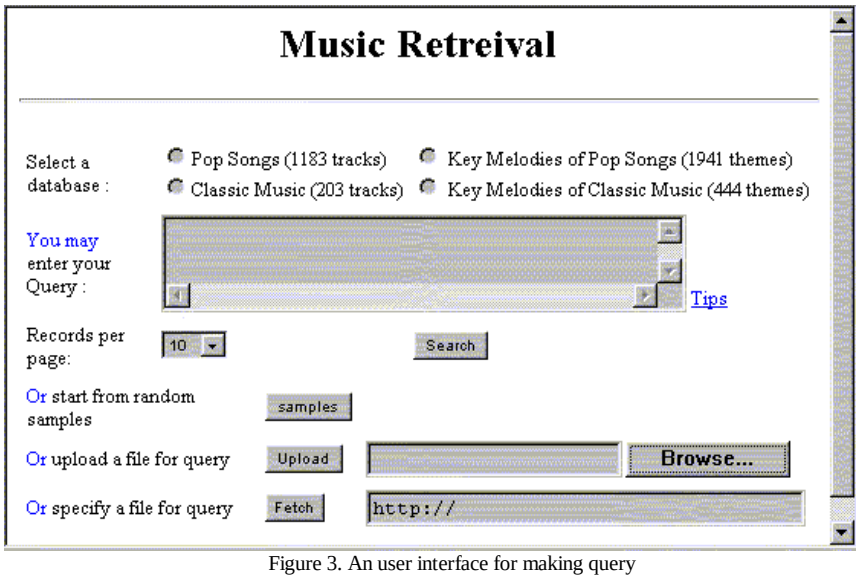

Figure 3. An user interface for making query

```
<mir><composer>Schubert</composer>
<title>Ave Maria</title>
<MIDI_file>/classic/sch_ave-4-1.mid</MIDI_file>
<track_No>4</track_No>
<instrument>not specified</instrument>
<key_melody_No>1</key_melody_No>
<frequency>2</frequency>
<melody>As5 A5 As5 D6 C6 As5</melody></mir>
```

Figure 2. A melody document example in SGML-like format

MIDI files are stored in an SGML-like format as in Figure 2. With an existing text search engine, these text information can be indexed and used for queries as well as the melody strings, both within the same indexing scheme and user interface. Thanks to this merit, we have indexed these data with a retrieval system which we have developed for our library's OPAC system [16] without any code revision in the indexing module. Only the user interface of the system is slightly changed.

N-gram indexing has been us ed in several retrieval tasks such as those for OCR-degraded text [17] and for documents in multiple languages [18]. This is possible because the n-gram approach imposes no assumption on the input text. N-grams are any n consecutive characters in a text. Since the basic element in a melody string is a note, we apply the n-gram method in note level (or in word level for text data). In our n-note approach, all m-notes, where m ranges from 1 to n, are all indexed. The shorter m-notes will match the input query in the presence of random errors, while the longer m-notes will favor the ones with more accurate melodies. The weight of each m-note in queries is given by 2(m-1)+1 while the weight of each m-note in documents takes the value of 1 or 0, indicating its presence in the documents or not. The similarity between document d_i and query q_j is then calculated by

$$Sim(d_i, q_j) = \frac{\sum_{k=1}^{T} d_{i,k} q_{j,k}}{\sum_{k=1}^{T} q_{j,k}}$$

where T is the total number of indexed m-notes. We have tried 2-note and 3-note indexing. Their performance can be seen in the experiments below.

As mentioned previously, the system provides several facilities to help query formulation. As shown in Figure 3, a button is provide for random generation of a batch of key melodies, from which users could choose one or more as their initial query to retrieve the whole tunes or similar melodies. If none were suitable, another random set could be requested. Users with, for example, a weak knowledge of music could choose this approach. Since this approach can be easily implemented by querying the system with a few random notes, there is no reason not to do so. However, without key melodies extracted as suggested items, the browsing and selection process definitely takes users more efforts. This query suggestion approach might be improved by clustering the key melodies in advance and presenting the results in a hierarchical way. We plan to discuss this issue in future work.

Users may also create a MIDI file locally and upload the file as a query. They may use their own auxiliaries or any music transcription tools available for creating such a file. Since a MIDI file can hold scores for multiple instruments, we have a special design that matches each track of notes according to the specified instrument, if this description is available in the file. Another similar function a user may exploit is to point to a MIDI file by giving its URL and ask the system to fetch it and search accordingly.

As to quickly making a query that directly focus on the interested tunes, users may input a sequence of notes in several notations. One internal notation is the standard pitch name: "C4 D4 … G4 A4 B4" for "Do Re … Sol La Si" in the fourth octave. Another simplified notation uses the digits "1 2 … 5 6 7" to stand for the seven pitches and operators "#", "-", and "+" following a digit to stand for a sharp, one octave below, and one octave above, respectively.

Additionally, the system allow queries by composers, titles, instruments, file names, and other text data with the same indexing scheme, i.e., n-note (n-word) indexing. All these text information are collected from the MIDI files as described. This makes the system an integrated one that can be searched by contents and by surrogates as well.

A successful retrieval might involve a series of query reformulation and result inspections. As shown in Figure 4, the system assist users in these processes by providing a playback button for users to make sure what they enter, a set of query

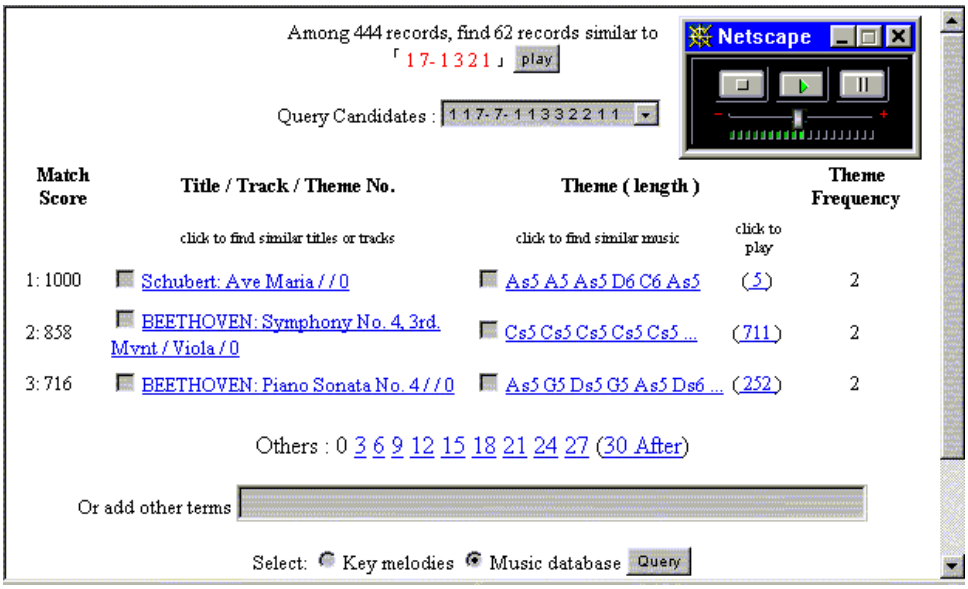

Figure 4. A search example.

candidates for other possible query formulations, and hyperlinks and check boxes on each returned items for relevance feedback by mouse clicking. The playback button is implemented by creating a MIDI file dynamically based on the input if the query is a sequence of notes in any notations allowable. The query candidates are generated according to a prior analysis of the MIDI collections. The types of mismatch that can not be overcome by the key-invariant encoding and n-note indexing are identified and used as guidelines for query suggestion. As shown in the experiment below, the most salient type of mismatch is a series of consolidation or fragmentation. So query candidates are generated accordingly.

4. Experiments

Although the system provides flexible ways for query formulation and friendly interface for query suggestion and relevance feedback, retrieval effectiveness is highly depends on the indexing and retrieval models. It is important to see what effects the mismatch problems specific to content-based music retrieval might have on the foregoing retrieval methods. Since each music work is created so uniquely, it is very likely that any two pieces of music under the same subject share little in common. Thus traditional recall and precision metrics in text retrieval is not used here. Instead, the experiment is to test the ability of various methods to retrieve a complete music work from a user's recall of a fragment of the music.

The ability to correct the key mismatch problem can be tested by comparing two representation schemes: one with key-invariant encoding and one without. By varying the value of n, it can be shown to what degree the n-note indexing can handle the mismatch problem caused by random pitch error or note insertion or deletion. The effects of consolidation, fragmentation, and other mismatch cases can be analyzed by examining users' queries and music data. Last, whether the use of key melodies improve the retrieval effectiveness can be illustrated by comparing direct retrieval of the music collection with indirect retrieval by first searching the key melody collection and then the music collection. Since the key melodies are exact subsequences of the original ones, the correct pieces of music can be retrieved without difficulty once their key melodies are found. Thus there are six retrieval modes to be tested in this experiment:

key melody search with original pitch encoding and 2-note indexing

1. direct music search with original pitch encoding and 2-note indexing
2. key melody search with key-invariant encoding and 2-note indexing
3. direct music search with key-invariant encoding and 2-note indexing
4. key melody search with key-invariant encoding and 3-note indexing
5. direct music search with key-invariant encoding and 3-note indexing

Eight users are invited in this experiment. Three of them had a few years of training on piano, one had a year of training on guitar, and the other four had only minimum musical training in school. Subjects are each presented with a title list of songs, consists of 30 classic music and 135 Chinese pop music. Queries are formulated based on their best recall of the songs they choose. If they cannot recall the melodies at once, they are allowed to listen to the music and make their queries based on their perception of the tunes. Subjects are suggested to examine the first 100 results. If they cannot find the target within the suggested range, they are advised the cause of mismatch and allowed to modify their queries accordingly. For example, in retrieval mode 1 the correct beginning pitch level is given if retrieval failure occurs due to key mismatch. Such modifications may allow us to induce more guidelines for query suggestion. Before formal experiments, subjects are allowed to practice the whole process with two songs. A total of 20 songs are searched in this experiment.

Table 2 shows the queries made by the subjects. The first seven songs are classic music and the remaining are pop music. For reference, the classic music searched are Toreador Song by Bizet, Piano Sonata No. 4 by Beethoven, Ave Maria by Schubert, Morning Mood by Grieg, and The Four Seasons 1st movement by Vivaldi. Two subjects choose the same melody fragment from Toreador Song for making queries. Their queries are in song 1 and 2 in Table 2. Another two subjects choose the same fragment from Piano Sonata No. 4, but with different melody length. Their queries are in song 3 and 4. The other 3 classic music are in song 5 to 7, in the order listed above. For each song, two different

Song mode	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	3	2	15	7	1	3	3	12	7	1	10	7	1	1	1	77	6	35	52	6
2	20	4	6	4	3	13	12	14	21	5	96	27	36	22	5	189	27	44	119	40
3	26	12	3	1	1	38	13	156	102	85	26	63	43	1	11	85	2	7	49	3
4	88	63	9	4	3	6	13	245	256	78	421	232	53	8	7	356	4	8	18	53
5	26	4	1	1	1	38	9	5	13	8	3	24	19	1	23	63	1	1	8	1
6	87	46	2	1	1	4	8	16	24	12	79	72	30	1	54	222	2	1	3	2

Table 3: Hit positions for 20 songs in 6 retrieval modes.

song	type	query	length
1	I	G5 A5 G5 E5 E5 E5 D5 E5 F5 E5	10
	I	5 6 5 3 3 3 2 3 4 3	
2	M	C6 C6 D6 D6 C6 C6 A5 A5 A5 A5 A5 G5 A5 B5 A5	15
	I	1+ 1+ 2+ 2+ 1+ 1+ 6 6 6 6 6 5 6 7 6	
3	I	G5 E5 C5 E3 G5 C6	6
	I	5 3 1 3 5 1+	
4	M	As5 G5 D5 G5 As5 C6 E6 D6 C6 E6 As5	12
	I	5 3 1 3 5 1+ 3+ 2+ 7 1+ 3+ 5	
5	M	As5 A5 As5 D6 C6 As5	6
	I	1 7- 1 3 2 1	
6	M	B4 Gs4 Fs4 D4 Fs4 Gs4	6
	I	5 3 2 1 2 3	
7	M	Gs6 Gs6 Gs6 Fs6 E6 B6	6
	I	3 3 3 2 1 5	
8	M	E5 D5 C5 C5 D5 E5 G5 A5 B5 C6 C6 A5 G5 E5 G5	15
	M	3 3 2 2 1 1 1 1 2 2 3 3 5 5 6 6 7 7 1+ 1+ 1+ 1+ 6 6 5 5 3 3 5 5	30
9	M	C6 C6 C6 C6 A5 G5 A5 F5 F5 G5 A5 D5	12
	M	5 5 5 5 5 5 5 5 3 3 2 2 3 3 1 1 1 1 2 2 3 3 6- 6-	24
10	M	Fs6 A6 B6 B6 A6 A6 Fs6 E6 D6 E6 Fs6 E6 D6 B5	14
	M	3 3 5 5 6 6 6 6 5 5 5 5 3 3 2 2 1 1 2 2 3 3 2 2 1 1 6- 6-	28
11	M	E5 G5 E5 E5 D5 E5 E5	7
	M	3 3 5 5 3 3 3 3 2 2 3 3 3 3	14
12	I	C6 D6 E6 G6 G6 G6 E6 D6 D6 D6 D6 E6 D6	13
	I	1 2 3 5 5 5 3 2 2 2 2 3 2	
13	M	G5 D6 D6 E6 C6	5
	I	5 2 2 3 1	
14	M	A5 E5 G5 G5 E5 D5 C5 D5 E5 A5	10
	I	6 3 5 5 3 2 1 2 3 6	
15	M	C5 C5 C5 C5 C5 D5 C5 D5 D5 D5 D5 D5 G5 A5	14
	I	1 1 1 1 1 2 1 2 2 2 2 2 5 6	
16	M	C5 C5 C5 A5 G5 D5 D5 D5 E5 D5 C5	11
	I	1 1 1 6 5 2 2 2 3 2 1	
17	M	C7 D7 E7 E7 E7 E7 E7 E7 E7 D7 C7 D7 E7	13
	I	1 2 3 3 3 3 3 3 3 2 1 2 3	
18	M	Fs5 Gs5 B5 Cs6 Cs6 B5 B5 B5	8
	I	5 6 1+ 2+ 2+ 1+ 1+ 1+	
19	M	C5 C5 D5 D5 E5 E5 F5 F5 A4 A4	10
	M	1 1 2 2 3 3 4 4 6- 6-	
20	M	G5 D6 D6 E6 C6 G5 D6 D6 E6 C6 E6 E6 B5 D6	14
	I	1 5 5 6 4 1 5 5 6 4 6 6 3 5	

Table 2: Queries made in the experiment.

queries are made: one in absolute pitch using standard notation for mode 1 and 2, and the other in relative pitch using simplified notation for mode 3, 4, 5, and 6. The second column indicates the type of the queries. Type I stands for initial queries and type M for modified queries. The number of notes used in the queries is shown in the last column.

mode	1	2	3	4	5	6
1		**18**	**12**	**15**	9	**12**
2	2		10	**13**	4	8
3	6	10		**15**	1	6
4	5	5	4		2	1
5	9	**15**	**14**	**18**		**13**
6	6	**12**	10	**19**	3	

Table 4. Comparisons between any two retrieval modes. The number in row i and column j indicates the number of cases that mode i outperforms mode j. For examples, there are 18 cases that mode 1 leads to earlier hits compared to mode 2. Since there is a total of 20 cases, numbers over half of the total cases are shown in **bold** face for ease of reading.

The retrieval results are shown in Table 3. In the results from mode 1 and 2, the use of key melodies improve performance since all the songs, except song 3 and 4, can be found earlier in the hit list. Similar situations occur in comparing retrieval results from mode 3 with 4 (15 out of 20 cases), and 5 with 6 (13 out of 20 cases). To facilitate comparison, Table 4 shows the number of cases that one retrieval mode outperforms the other.

By comparing mode 1 with 3 and 2 with 4, it suggests that the use of absolute pitch encoding performs slightly better (12 out of 20 cases and 13 out of 20 cases, respectively) than those using key-invariant encoding. However, this is under the condition that users input in a correct key or in a closely related key (for example, see the queries in song 1 and 2 or 3 and 4 in Table 2). If an incorrect key is chosen in the queries, search failure can be expected. Table 2 shows that only 3 queries are made in correct pitch level at the first time without further query modification.

The relatively less indexing symbols resulting from key-invariant encoding leads to inferior discrimination among music documents. Thus a larger n for n-note indexing should be used for generating more indexing terms to maintain the performance. The superior performance in mode 5 compared to 3 and mode 6 compared to 4 confirms this observation.

The mismatch caused by consolidation or fragmentation cannot be easily solved by n-note indexing. Some query reformation is needed as shown in Table 2. Inspection of the music data reveals that there is no serious consolidation or fragmentation in the five classic music, so no modification is made for the queries in the simplified notation. Even though the queries do not exact match the melody strings stored in the collection, the key-invariant encoding and the n-note indexing work as desired. For the popular songs in the experiment, however, each notes are repeated in at least 5 cases (song 8, 9, 10, 11, and 19). The burst errors makes n-note retrieval fail, requiring query reformulation either by users manually or by the system automatically, if a prior knowledge of this type of mismatch is known.

5. Conclusions and Future Work

Several types of mismatch specific to content-based music

retrieval are discussed. Strategies, such as key-invariant encoding and n-note indexing, to overcome these mismatches are presented. Experiments shows that key mismatch and random errors can be handled by the proposed strategies. For burst errors caused by a series of consolidation or fragmentation, query expansion is required manually or automatically. This can be facilitated by a flexible and friendly interface. One distinct feature of the developed system is the key melody extraction. Based on the assumption that memorable segments are often repeated in music, an algorithm is presented for efficient extraction. Note that by appropriate quantization to eliminate small variations on the input sequences, this algorithm can also apply to audio waveforms for repeated pattern extraction. This might help in other audio retrieval applications.

In this work, we assume music in e ach track from the MIDI files is monophonic, i.e., no notes occur simultaneously in the note sequence. This assumption eases the work of matching the extracted melodies against the original MIDI files for obtaining the sub-MIDI files, because no garbling is made during the melody extraction and the MIDI reconstruction processes. However, retrieval performance is affected if polyphonic melodies are encountered: users tend to have more difficulty in preparing a query that contains simultaneous notes. Uitdenbogerd et al [19], discussed 4 algorithms for extracting a melody from polyphonic music. They showed that a simplest technique in which the extracted melody consists of highest pitch-notes appearing in any track is the most effective. With this technique, the discrepancy between queries and polyphonic melodies may be reduced, which in turn may thus lead to less search failure.

Repetitions are observed in almost every music work. However, repetitions may have slight variations to enrich the melody and to facilitate its better learning. For these cases, Bakhmutova, et al [20], has presented a human-computer procedure for revealing similar melodies, i.e., melodies of not only identical repetitions, but also fragments that were close in some sense. This procedure requires a learning stage to estimate some parameters by forming a set of melodies with several versions each. Besides, the final decision about closeness of melodies is made after listening, by preferably playing melodies in the same rhythm if their rhythms are different. For efficient key melody extraction allowing variations, fully automatic methods are worth of further investigation.

Future work will also examine the possibility of clustering the extracted key melodies in a hierarchical way for efficient browsing and navigation. Retrieval effectiveness may be further compared with other retrieval modes under different circumstances for best design rules. This might involve variations on different weighting schemes and indexing methods, or comparisons between conventional retrieval models and other approximate string matching approaches.

References:

[1] Rodger J. McNab, Lloyd A. Smith, Ian H. Witten, Clare L. Henderson and Sally Jo Cunningham, "Towards the Digital Music Library: Tune Retrieval from Acoustic Input," In Proceedings of the ACM Digital Libraries 1996, pp. 11-18.

[2] Silvia Pfeiffer, Stephan Fischer and Wolfgang Effelsberg, "Automatic Audio Content Analysis," In Proceedings of the Fourth ACM International Multimedia Conference, 1996, pp. 21-30.

[3] Wold, E., Blum, T., Keislar, D., and Wheaton, J., "Content-based Classification, Search, and Retrieval of Audio," IEEE Multimedia, Vol. 3, No. 3, 1996, pp. 27-36.

[4] Jonathan T. Foote, "Content-Based Retrieval of Music and Audio," Multimedia Storage and Archiving Systems II, Proceedings of SPIE, Vo. 3229, 1997, pp. 138-147. Http://svr-www.emg.cam.ac.uk/jtf/papers/spie97-abs.html

[5] Hawley, M., "The Personal Orchestra," Computing Systems, Vol. 3, No. 2, 1990, pp.289-329.

[6] Ghias, A., Logan, H., chamberlin, D., and Smith, B. C., "Query by Humming: Musical Information Retrieval in an Audio Database," In Proceedings of Third ACM International Conference on Multimedia, 1995, pp. 231-236.

[7] Rodger J. McNab, Lloyd A. Smith, David Bainbridge and Ian H. Witten, "The New Zealand Digital Library MELody inDEX," D-Lib Magazine, May 1997. Http://www.dlib.org/dlib/may97/meldex/05witten.html

[8] Wu, S. and Manber, U. "Fast Text Searching Allowing Errors," Communications of ACM, Vol. 10, Oct. 1992, pp. 83-91.

[9] Benjamin Chen and J .-S. Roger Jang, "Query by Singing," In Proceedings of the 11st Conference on Computer Vision, Graphics, and Image Processing, Taiwan, 1998, pp. 529-536.

[10] "AKoff Music Composer" AKoff Web Site: http://www.geocities.com/ SiliconValley/Hills/4313/

[11] Jia-Lien Hsu, Chih-Chin Liu, and Arbee L. Chen, "Efficient Repeating Pattern Finding in Music Databases," Proceedings of ACM Seventh International Conference on Information and Knowledge Management.

[12] Jones, G. T., *Music Theory*, Harper & Row, Publishers, New York, 1974.

[13] Yuen-Hsien Tseng, "Multilingual Keyword Extraction for Term Suggestion," Proceedings of 21st International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '98, Aug. 24-28, Australia, 1998, pp.377-378.

[14] William B. Frakes and Ricardo Baeza-Yates, Editors, *Information Retrieval: Data Structure and Algorithms*, Prentice Hall, 1992.

[15] Larry Wall and Randal L. Schwartz, *Programming Perl*, O'Reilly & Associates, Inc., 1991.

[16] Yuen-Hsien Tseng and Yu-I Lin "Evaluation of Fuzzy Search, Term Suggestion, and Term Relevance Feedback in an OPAC System," Bulletin of the Library Association of China, No. 61, 1998, pp.103-126.

[17] Harding, W. B. Croft, and C. Weir, "Probabilistic Retrieval of OCR Degraded Text Using N-Grams," 1995, in Research and Advanced Technology for Digital Libraries, Carol Peters and Costantino Thanos, Editors, 1997. pp. 345-359. http://ciir.cs.umass.edu/info/psfiles/irpubs/ir-115.ps.gz

[18] Jonathan D. Cohen, "Highlights: Language- and Domain-Independent Automatic Indexing Terms for Abstracting", Journal of the American Society for Information Science, 46(3), 1995, pp.162-174.

[19] Alexandra L. Uitdenbogerd and Justin Zobel, "Manipulation of Music for Melody Matching," Proceedings of the ACM Multimedia 98, Sep. 11-15, England, 1998, pp.235-240.

[20] Irene V. Bakhmutova, Vladimir D. Gusev and Tatiana N. Titkova, "The Search for Adaptations in Song Melodies," Computer Music Journal, Vol. 21, No. 1, 1997, pp. 58-67.