Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases

Chih-Chin Liu, Jia-Lien Hsu and Arbee L. P. Chen

Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C., Email : alpchen@cs.nthu.edu.tw

Abstract

In this paper, we propose an approach for fast discovering all non-trivial repeating patterns in music objects. A repeating pattern is a sequence of notes which appears more than once in a music object. The longest repeating patterns in music objects are typically their themes. The themes and other non-trivial repeating patterns are important music features which can be used for both content-based retrieval of music data and music data analysis. We present a data structure called RT-tree and its associated algorithms for fast extracting all non-trivial repeating patterns in a music object. Experiments are performed to compare with the related approaches. The results are further analyzed to show the efficiency and the effectiveness of our approach.

1. Introduction

In recent years, the searching and indexing techniques for multimedia data are getting more attention in the area of multimedia databases. As most research work were done on the content-based retrieval of image and video data [10][11][14][20][25][29], less attention was received to the content-based retrieval of audio data. Traditionally, the audio objects are treated as a long sequence of bytes. Except for some descriptive attributes such as name, file format, or sampling rate, an audio object is treated as an opaque large object in which its semantics is omitted. Although many meaningful feelings can be comprehended by human beings when listening to an audio object, it is very difficult to automatically extract these feelings from the raw data of the audio object.

For the audio objects, Wold, *et al.* proposed an approach to retrieve them based on their content [34]. In this approach, an N-vector is constructed according to the acoustical features of an audio object. These acoustical features include *loudness*, *pitch*, *brightness*, *bandwidth*, and *harmonicity*, which can be automatically computed from the raw data. The N-vector of the acoustical features is then used to classify sounds for similarity searching. However, the acoustical features are at a level too low for the human beings.

The most straightforward way to query the music databases for a naive user is to hum a piece of music as the query example to retrieve similar music objects. This approach is adopted in [7][9][13][21]. Ghias, *et al.* proposed an approach to transform a music query example into a string which consists of three kinds of symbols, "U", "D", and "S", which represent a note is higher than, lower than, or the same as its previous note, respectively [13]. The problem of music query processing is then transformed into that of approximate string matching. However, only using three kinds of symbols is too rough to represent the melody of a music object. Moreover, the proposed string matching algorithm did not take music characteristics into consideration.

To develop a content-based music database system, we have started a project called *Muse* [7][9][21]. In this project, we have proposed three different methods for the content-based music data retrieval. In [21], we treat the *rhythm*, *melody*, and *chords* of a music object as music feature strings and develop a data structure called *1D-List* to efficiently perform approximate string matching. Similarity measures in the approximate string matching algorithm are based on music theory. In [9], we consider music objects and music queries as sequences of chords. An index structure is developed to provide efficient partial matching ability. In [7], we propose an approach for retrieving music objects by rhythm. In this approach, the rhythm of a music object is modeled by rhythm strings consisting of *mubols*. A mubol is the rhythmic pattern of a measure in a music object. Five kinds of similarity relationships between two mubols are defined and the similarity measure of two rhythm strings can be calculated based on these similarity relationships. An index structure is also proposed for efficiently matching rhythm strings.

No matter the content-based music data retrieval is based on humming, melody, chord, or rhythm, string matching is the basic technique for the music query processing. However, if the music objects are large (for example, 1000 notes), the execution time for query processing will become unacceptable. By examining the *staff* of a music object, we find many sequences of notes, called *repeating patterns*, appear more than once in the music object. For example, the well known "sol-sol-sol-mi" in Beethoven Symphony No. 5 repeatedly appear in the music object. Many researchers in musicology and music psychology fields also agree that repetition is a universal characteristic in music structure modeling [4][17][24][30]. Therefore, to represent a music object by its repeating patterns meets both efficient and semantic

requirements for content-based music data retrieval. An efficient technique for finding the repeating patterns from the sequence of notes of a music object is thus needed to be developed.

In [15], we have proposed an approach to efficiently find all repeating patterns of music objects based on a data structure called *correlative matrix*. For a music object of *n* notes, an (*n* x *n*) correlative matrix is constructed to keep the intermediate results during the finding process. In this paper, we propose a more efficient approach to discover all repeating patterns of music objects. In this approach, the longer repeating pattern of a music object is discovered by repeatedly combining shorter repeating patterns. Thus the storage space and execution time can be reduced.

A *suffix tree* is a tree-like data structure representing all suffixes of a string [8][23]. It is originally developed for substring matching [23]. However, since all suffixes and their appearances in a string are stored in a suffix tree, it can also be used for finding the repeating patterns. Many approaches were proposed for efficiently constructing the suffix tree of a given string [8][23][32][33]. Many applications of suffix trees were also developed, such as to find the substrings which appear regularly [2], to detect all *palindrome subwords* [2], and to find the number of *non-overlapping* occurrences in a given string [3]. However, since suffix tree is not developed for finding repeating patterns, much storage space and execution time is required for traversing the suffix tree to generate the repeating patterns of a given string.

The paper is organized as follows. In Section 2, we use examples to show the basic idea of the proposed method for finding repeating patterns. The algorithms for finding repeating patterns are formally described in Section 3. Some experiments are performed to show the efficiency of the proposed algorithms. The results are further analyzed to show the effectiveness of the approach in Section 4. Finally, Section 5 concludes this paper and presents our future research directions.

2. Approach Overview for Finding Repeating Patterns

In this section we illustrate the basic idea of the proposed approach for finding repeating patterns through examples. The formal algorithms for finding repeating patterns will be described in the next section.

For a substring Y of a music feature string X, if Y appears more than once in X, we call Y a *repeating pattern* of X. The *frequency* of the repeating pattern Y, denoted as *freq*(Y), is the number of appearances of Y in X. The *length* of the repeating pattern Y, denoted as |Y|, is the number of notes in Y.

Consider the melody string "do-re-mi-fa-do-re-mi-do-re-mi-fa", this melody string has ten repeating patterns as is shown in Table 1. However, the repeating patterns "re-mi-fa", "mi-fa", and "fa" are substrings of the repeating pattern "do-re-mi-fa" and *freq*("do-re-mi-fa") = *freq*("re-mi-fa") = *freq*("mi-fa") = *freq*("fa") = 2. Similarly, the repeating patterns "do-re", "re-mi", "do", "re" and "mi" are substrings of the repeating pattern "do-re-mi" and *freq*("do-re-mi") = *freq*("do-re") = *freq*("re-mi") = *freq*("do") = *freq*("re") = *freq*("mi") = 3. Among the ten repeating patterns of the melody string, only "do-re-mi-fa" and "do-re-mi" are *non-trivial*. To make the results more concise, instead of finding all repeating patterns of a given melody string, we only find its non-trivial repeating patterns. The definition of a non-trivial repeating pattern is described as follows.

RP	do-re-mi-fa	do-re-mi	re-mi-fa	do-re	re-mi
Freq.	2	3	2	3	3
RP	mi-fa	do	re	mi	fa
Freq.	2	3	3	3	2

Table 1: All repeating patterns of the melody string "do-re-mi-fa-do-re-mi-do-re-mi-fa".

Definition 1 A repeating pattern X is *non-trivial* if and only if there does not exist another repeating pattern Y such that *freq*(X) = *freq*(Y) and X is a substring of Y.

Consider all repeating patterns of the song "Five Hundred Miles". Its longest repeating pattern is its *refrain* "do-do-do-mi-mi-re-do-mi-mi-re-do-re-mi-re-do-la-la-do-re-mi-re-do-la-sol-sol-la-do-do". The length of the repeating pattern is 28. In our previous approach [15], we find the repeating patterns incrementally based on the lengths. In this example, we will sequentially find the following repeating patterns: "do", "do-do", "do-do-do", "do-do-do-mi", …,"do-do-do-mi-mi-re-do-mi-mi-re-do-re-mi-re-do-la-la-do-re-mi-re-do-la-sol-sol-la-do-do". This is inefficient since most substrings of the refrain are trivial. In this paper, we propose a new approach for finding all non-trivial repeating patterns based on an exponential increase of the lengths of the repeating patterns. For example, considering the melody string "do-re-mi-fa-do-re-mi-do-re-mi-fa", we first scan this string to find all repeating patterns of length one. We use the form {X, *freq*(X), (*position1*, *position2*, …)} to represent the results, where *freq*(X), *position1*, and *position2* represent the frequency, the first position, and the second position of the repeating pattern X in the melody string, respectively. All repeating patterns of length one are {"do", 3, (1, 5, 8)}, {"re", 3, (2, 6, 9)}, {"mi", 3, (3, 7, 10)}, and {"fa", 2, (4, 11)}

A repeating pattern of length two can be found by *joining* (denoted as "∞") two repeating patterns of length one. The join operation will be formally defined in the next section and we use an example to explain it here. Assume we want to check whether "do-re" is a repeating pattern. Since we know "do" and "re" appear at (1, 5, 8) and (2, 6, 9), respectively, we can confirm that "do-re" also appears at (1, 5, 8). This can be represented as the following equation.

{"do", 3, (1, 5, 8)} ∞ {"re", 3, (2, 6, 9)} = {"do-re", 3,

(1, 5, 8)}

Similarly, we have

{“re”, 3, (2, 6, 9)} ∞ {“mi”, 3, (3, 7, 10)} = {“re-mi”, 3, (2, 6, 9)}

{“mi”, 3, (3, 7, 10)} ∞ {“fa”, 2, (4, 11)} = {“mi-fa”, 2, (3, 10)}

Similarly, we can join two repeating patterns of length two to generate a repeating pattern of length four:

{“do-re”, 3, (1, 5, 8)} ∞ {“mi-fa”, 2, (3, 10)} = {“do-re-mi-fa”, 2, (1, 8)}

Since *freq*(“do-re-mi-fa”) = *freq*(“mi-fa”) = 2, we can conclude that not only “mi-fa” is trivial, “re-mi-fa” must be also trivial (otherwise, *freq*(“mi-fa”) must be greater than two). We only have to join “do-re” and “re-mi” to check whether “do-re-mi” is a repeating pattern. The result is {“do-re-mi”, 3, (1, 5, 8)}. Since *freq*(“do-re-mi”) > *freq*(“do-re-mi-fa”), “do-re-mi” is non-trivial. Therefore, the non-trivial repeating patterns in “do-re-mi-fa-do-re-mi-do-re-mi-fa” are “do-re-mi-fa” and “do-re-mi”.

3. Algorithms for the String-Join Approach

In this section, we formally describe the proposed approach for finding all non-trivial repeating patterns in the music feature string of a music object. This approach is based on repeatedly *joining* two shorter repeating patterns to form a longer one. We first define the join operation named *string-join* as follows.

Definition 2: Assume $\{\alpha_1\alpha_2\ldots\alpha_m, freq(\alpha_1\alpha_2\ldots\alpha_m), (p_1, p_2, \ldots, p_i)\}$ and $\{\beta_1\beta_2\ldots\beta_n, freq(\beta_1\beta_2\ldots\beta_n), (q_1, q_2, \ldots, q_j)\}$ are two repeating patterns in the music feature string of a music object. We define *order-k string-join* ($k \geq 0$) of the two repeating patterns as $\{\alpha_1\alpha_2\ldots\alpha_m, freq(\alpha_1\alpha_2\ldots\alpha_m), (p_1, p_2, \ldots, p_i)\} \infty_k \{\beta_1\beta_2\ldots\beta_n, freq(\beta_1\beta_2\ldots\beta_n), (q_1, q_2, \ldots, q_j)\} = \{\gamma_1\gamma_2\ldots\gamma_l, freq(\gamma_1\gamma_2\ldots\gamma_l), (o_1, o_2, \ldots, o_h)\}$ where $i = freq(\alpha_1\alpha_2\ldots\alpha_m)$, $j = freq(\beta_1\beta_2\ldots\beta_n)$, *and* $h = freq(\gamma_1\gamma_2\ldots\gamma_l)$ and

$\gamma_t = \alpha_t$, for $1 \leq t \leq m$,

$\gamma_t = \beta_{t-m+k}$, for $m+1 \leq t \leq l = m+n-k$,

$o_t = x = y - m + k$, where $x \in \{p_1, p_2, \ldots, p_i\}$ and $y \in \{q_1, q_2, \ldots, q_j\}$,

$o_t < o_{t+1}$, for $1 \leq t \leq h-1$,

if $k > 0$, $\alpha_{m-k+s} = \beta_s$, for $1 \leq s \leq k$.

As we have discussed in section 2, instead of finding all repeating patterns in a music feature string, we only need to find all non-trivial repeating patterns. The algorithm for finding all non-trivial repeating patterns is shown in Algorithm 1. It consists of two parts: in the first part (line 1~9), the repeating patterns of length one (line 2), the repeating patterns of length 2^k, $k > 0$ (line 3~8), and the longest repeating pattern (line 9) are found. In the second part, we remove the repeating patterns which are trivial (line 10~11) and generate the other non-trivial repeating patterns (line 12). The first part is explained as follows.

The repeating patterns of length one can be found by scanning the music feature string, *i.e.* the procedure of *Find_RP1*. Due to the space limitation, all procedures and proofs are omitted. Then, we can generate a repeating pattern of length 2n by joining two length n repeating patterns using an order-0 string-join. Assume the length of the longest repeating pattern is L, by repeatedly joining the repeating patterns found, we can generate all repeating patterns whose length is 2^k, for k = 1, 2, …, m, where m is the largest number such that $2^m \leq L$. The process is done by the *Find_RP2K* procedure. After the repeating patterns of length 2^m are found, we can generate the longest repeating pattern $\gamma_1\gamma_2\ldots\gamma_L$ by joining two repeating patterns of length 2^m:$\{\gamma_1\gamma_2\ldots\gamma_L, freq(\gamma_1\gamma_2\ldots\gamma_L), (o_1, o_2, \ldots, o_h)\} = \{\alpha_1\alpha_2\ldots\alpha_2{}^m, freq(\alpha_1\alpha_2\ldots\alpha_2{}^m), (p_1, p_2, \ldots, p_i)\} \infty_\lambda \{\beta_1\beta_2\ldots\beta_2{}^m, freq(\beta_1\beta_2\ldots\beta_2{}^m), (q_1, q_2, \ldots, q_j)\}$, where $\lambda = 2^{m+1} - L$. This is because the length of the repeating pattern generated by joining two length n repeating patterns using an order-λ string-join will be $(2n-\lambda)$. In the above example, the length of the resultant repeating pattern will be $(2x2^m - (2^{m+1} - L)) = L$. However, since we do not know the length of the longest repeating pattern in advance, we have to use an efficient method to determine it. Since we know $2^m \leq L < 2^{m+1}$, the best way to find L is using a binary search order in the generating processing, *i.e.* the procedure *Find_Longest_RP*

Algorithm 1 *Find_Repeating_Pattern(**S**)*

/ input: the music feature string **S** of a music object */*

/ output : the set of all non-trivial repeating patterns and their frequencies in **S** */*

```
0:   begin
1:       RP = φ
2:       RP[1] = Find_RP1(S)
3:       k = 0
4:       while RP[2^k] <> φ do
5:       begin
6:           k = k + 1
7:           RP[2^k] = Find_RP2K(RP[2^(k-1)])
8:       end
9:       RP[L] = Find_Longest_RP(RP[2^k])
10:      Build_RP_Tree()
11:      Refine_RP_Tree()
12:      Generate_Non-trivial_RP(RP)
13:      return RP
14:  end
```

Example 1 Assume there is a music feature string ***S*** = ABCDEFGHABCDEFGHIJABC. To find all non-trivial repeating patterns in ***S***, we first apply the procedure *Find_RP1* on ***S***. The result is the set of all length one repeating patterns as shown in the following:

RP[1] = {{A, 3, (1, 9, 19)}, {B, 3, (2, 10, 20)}, {C, 3, (3, 11, 21)}, {D, 2, (4, 12)}, {E, 2, (5, 13)}, {F, 2, (6, 14)}, {G, 2, (7, 15)}, {H, 2, (8, 16)}}

Then we apply the procedure *Find_RP2K* on RP[1]. By joining each pair of length one repeating patterns in RP[1] using an order-0 string-join, we can derive RP[2], the set of

all length two repeating patterns in ***S***.

RP[2] = {{AB, 3, (1, 9, 19)}, {BC, 3, (2, 10, 20)}, {CD, 2, (3, 11)}, {DE, 2, (4, 12)}, {EF, 2, (5, 13)}, {FG, 2, (6, 14)}, {GH, 2, (7, 15)}}

Similarly, by repeatedly applying the procedure *Find_RP2K*, we can derive RP[4] and RP[8].

RP[4] = {{ABCD, 2, (1, 9)}, {BCDE, 2, (2, 10)}, {CDEF, 2, (3, 11)}, {DEFG, 2, (4, 12)}, {EFGH, 2, (5, 13)}}

RP[8] = {{ABCDEFGH, 2, (1, 9)}}

For the next step, we apply the procedure *Find_Longest_RP* on RP[8] to find the longest repeating pattern in ***S***. Since there does not exist any repeating pattern whose length is greater than 8, RP[8] is the set of the longest repeating patterns in ***S***.

By applying the procedures of *Find_RP1*, *Find_RP2K*, and *Find_Longest_RP* on a music feature string **S**, we can generate the sets of all length 1, 2^1, …, 2^k repeating patterns and the set of the longest repeating patterns in **S**. However, we still have to check whether there exist non-trivial repeating patterns of other lengths. Moreover, except for the longest repeating patterns, which must be non-trivial, we do not know whether the repeating patterns of length 1, 2^1, …,2^k are non-trivial. We have the following properties for solving these problems.

Lemma 1 Let X be a repeating pattern of a music feature string ***S***. For each substring Y of X, *freq*(Y) ≥ *freq*(X).

Lemma 2 Let X be a repeating pattern of a music feature string ***S***. For each substring Y of X, Y is also a repeating pattern of ***S***.

Theorem 1 Let X be a repeating pattern of a music feature string ***S***, Y be a substring of X, and Z be a substring of Y. If *freq*(X) = *freq*(Z), Y is trivial.

Proof: According to Lemma 1, we have *freq*(Z) ≥ *freq*(Y) ≥ *freq*(X). If *freq*(X) = *freq*(Z), *freq*(Y) will be equal to *freq*(X). According to Lemma 2, Y does not appear in ***S*** at the position where X does not appear. By the definition, Y is trivial. ∎

Theorem 2 Let X be a repeating pattern of a music feature string ***S*** and the length of X be 2^n. For a length 2^{n-1} substring Y of X, if *freq*(Y) = *freq*(X), every substring of X which contain Y and whose length is between 2^n–1 and 2^{n-1}+1 is trivial.

Theorem 1 and Theorem 2 can be used to determine whether the length 2^0, 2^1, …, 2^k repeating patterns, which are found by the procedure of *Find_RP1* and *Find_RP2K*, are non-trivial. To make the checking processing more efficient, we introduce a tree structure call *RP-tree*. Each node in a RP-tree represents a repeating pattern found. If a repeating pattern Y is a substring of another repeating pattern X, we build a link between Y and X. However, for three repeating patterns X, Y, and Z, if Y is a substring of X and Z is a substring of Y, we do not link Z to X to make the RP-tree more compact. The construction of a RP-tree is formally described in the procedure *Build_RP_Tree*.

For two linked nodes $\{\alpha_1\alpha_2\ldots\alpha_m, freq(\alpha_1\alpha_2\ldots\alpha_m), (p_1, p_2, \ldots, p_i)\}$ and $\{\beta_1\beta_2\ldots\beta_n, freq(\beta_1\beta_2\ldots\beta_n), (q_1, q_2, \ldots, q_j)\}$ in an RP-tree, if $\beta_1\beta_2\ldots\beta_n$ is a substring of $\alpha_1\alpha_2\ldots\alpha_m$, and $freq(\alpha_1\alpha_2\ldots\alpha_m)$ equals $freq(\beta_1\beta_2\ldots\beta_n)$, according to Theorem 1, we can decide $\beta_1\beta_2\ldots\beta_n$ is a trivial repeating pattern. Moreover, according to Theorem 2, every substring of $\alpha_1\alpha_2\ldots\alpha_m$ which contains $\beta_1\beta_2\ldots\beta_n$ and whose length is between m and n is trivial. We can remove $\{\beta_1\beta_2\ldots\beta_n, freq(\beta_1\beta_2\ldots\beta_n), (q_1, q_2, \ldots, q_j)\}$ from the RP-tree. These rules are utilized in the following algorithm to refine the RP-tree.

After all repeating patterns whose length is power of two are generated, the final step is to generate all non-trivial repeating patterns whose length is not power of two. Assume $\{\delta_1\delta_2\ldots\delta_n, freq(\delta_1\delta_2\ldots\delta_n), (s_1, s_2, \ldots, s_g)\}$ is a non-trivial repeating pattern where $2^k < n < 2^{k+1}$. This repeating pattern can be generated by joining two repeating patterns whose length is 2^k. After inserting this repeating pattern in the RP-tree, we should check whether this repeating pattern and all its substrings in the RP-tree are non-trivial. Theorem 1 and Theorem 2 can be applied again in this case. The resultant RP-tree will contain all non-trivial repeating patterns. The final result can be generated by traversing the resultant RP-tree. The procedure *Generate_Non-trivial_RP* formally describes the processing for generating all non-trivial repeating patterns.

Example 2 To construct the RP-tree corresponding to the repeating patterns found in Example 1, we apply procedure *Buile_RP* on RP[1], RP[2], RP[4], and RP[8]. First we build a link from {AB, 3, (1, 9, 19)} to {A, 3, (1, 9, 19)} and {B, 3, (2, 10, 20)} since both A and B are substrings of AB. Other links are constructed in the same way. The RP-tree constructed is shown in Figure 1.

To refine this RP-tree, we apply procedure *Refine_RP_Tree* on this RP-tree. Since *freq*(A) = *freq*(B) = *freq*(C) = *freq*(AB) = *freq*(BC) = 3 and *freq*(D) = *freq*(E) = *freq*(F) = *freq*(G) = *freq*(H) = *freq*(DE) = *freq*(EF) = *freq*(FG) = *freq*(GH) = 2, the repeating patterns A, B, C, D, E, F, G, and H are trivial and their corresponding nodes are removed from the RP-tree, as shown in Figure 2(a). Similarly, since *freq*(CD) = *freq*(DE) = *freq*(EF) = *freq*(FG) = *freq*(GH) = *freq*(CDEF) = *freq*(DEFG) = *freq*(EFGH)= 2, the repeating patterns CD, DE, EF, FG, and GH are trivial and their corresponding nodes are removed from the RP-tree, as shown in Figure 2(b). Finally, since *freq*(ABCD) = *freq*(BCDE) = *freq*(CDEF) = *freq*(DEFG) = *freq*(EFGH) = *freq*(ABCDEFGH) = 2, the repeating patterns ABCD, BCDE, CDEF, DEFG, and EFGH are removed from the RP-tree, as shown in Figure 2(c).

After the RP-tree is refined by removing the trivial repeating patterns whose length is power of two, the final step is to generate all non-trivial repeating patterns. Apply procedure *Generate_Non-trivial_RP* on the RP-tree shown in Figure 2(c), a new repeating pattern {ABC, 3, (1, 9, 19)} is generated by joining {AB, 3, (1, 9, 19)} and {BC, 3, (2,

10, 20)} using an order-1 string-join, as shown in Figure 3(a). Since *freq*(ABC) > *freq*(ABCDEFGH), ABC is non-trivial. However, since *freq*(ABC) = *freq*(AB) = *freq*(BC) = 3, the nodes corresponding to AB and BC are removed as shown in Figure 3(b). Traversing this RP-tree, we find all the non-trivial repeating patterns of the music feature string ***S***, which are ABCDEFGH and ABC.

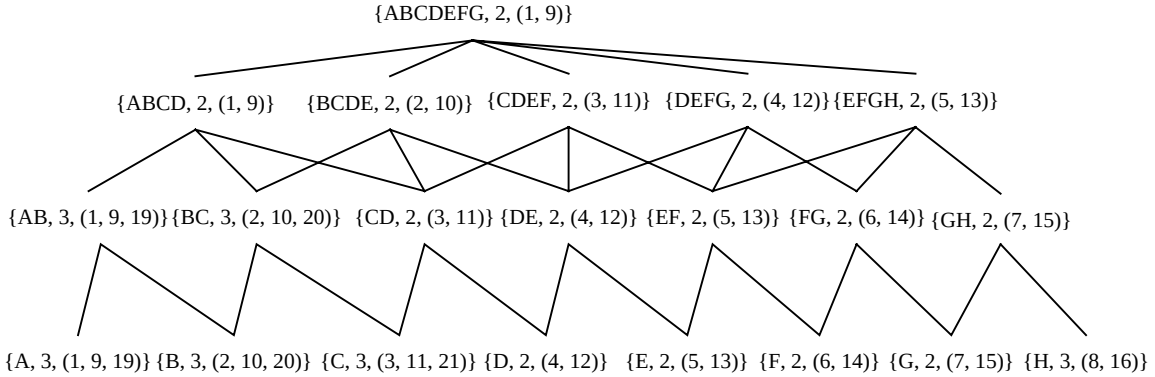

Figure 1: The RP-tree for the music feature string S = ABCDEFGHABCDEFGHIJABC.

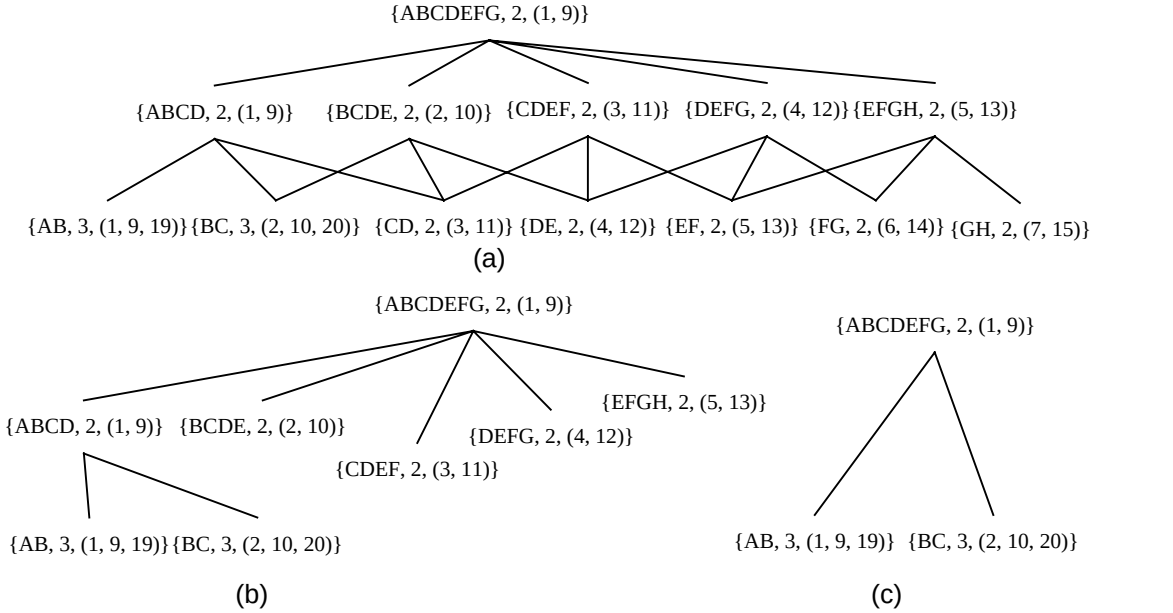

Figure 2: (a) The RP-tree after all trivial repeating patterns of length one are removed. (b) The RP-tree after all trivial repeating patterns of length two are removed. (c) The RP-tree after all trivial repeating patterns of length four are removed.

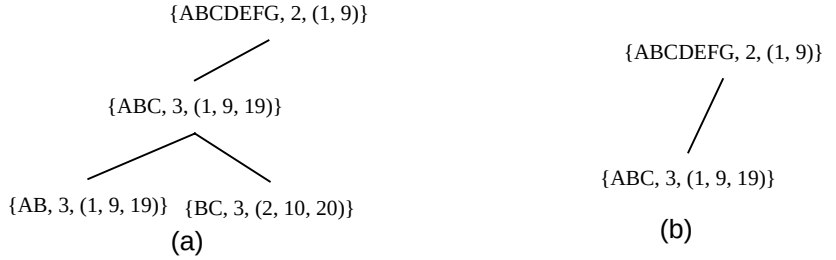

Figure 3: (a) The RP-tree after all repeating patterns of length three are generated. (b) The RP-tree after all trivial repeating patterns are removed.

4. Experiments

To show the efficiency of the proposed algorithm, a series of experiments are performed to compare with the related approaches.

4.1 Experiment Set-up

We implement the proposed algorithm for finding non-trivial repeating patterns by ANSI C and perform a series of experiments on a Sun Sparc20 workstation. The performance is measured by elapsed time. There are two data sets used in these experiments. One contains synthetic music objects and the other contains real music objects. The synthetic music objects are generated with uniformly distributed notes. The real music objects are collected from websites, mainly the *Classical MIDI Archives Site* (http://www.prs.net). They are classical music objects of MIDI form composed by various composers. All music objects are parsed to extract their corresponding melody strings. The object size of a music object is measured by the length of the corresponding melody string. For the real music objects, their average object size is 619.3. The *note count* is defined as the number of distinct notes appearing in a melody string. According to the MIDI Standard [22], the alphabet size is 128. Therefore, the note count of every melody string is between 1 and 128. According to the experiments, the average note count of the real music objects is 23.2.

All experiments are performed with three repeating pattern finding approaches. The details of our previous approach, named the correlative matrix approach, can be found in [15]. The procedure for generating all repeating patterns in a music feature string by the suffix tree approach is briefly described as follows

The *suffix tree T* of a music feature string $\boldsymbol{S}$ is a ($|A|$ +1)-ary rooted tree where $|A|$ is the alphabet size [23]. Each leaf of *T* corresponds to a position in $\boldsymbol{S}$, and each link is labeled with a symbol α, where $\alpha \in A \cup \{\$\}$ ('$' is a special symbol denoting "end-of-string"). Each non-leaf node *P* is associated with a value which is the number of all leaf nodes descending from the node *P*. Each path X from the root to a non-leaf node *P* represents a substring of $\boldsymbol{S}$. If the number associated with *P* is equal to or larger than two, the corresponding substring of the path X will be a repeating pattern in $\boldsymbol{S}$.

4.2 Performance Analysis

There are four factors dominating the performance of the proposed algorithm: the number of music objects, the object size of music objects, the number of non-trivial repeating patterns in a music feature string, and the length of the longest repeating pattern in a music feature string. The first two experiments illustrate the elapsed time versus the number of music objects for the synthetic music data set and real music data set. For the synthetic music data, the object size and note count are set to 500 and 30, respectively. The elapsed time of the three approaches in both data set increases linearly as the number of music objects increases. In our experiments, the elapsed time of the real music data set is greater than that of the synthetic music data set. Moreover, the elapsed time of the string-join approach is about one third of that of the correlative matrix approach for the synthetic music data set while it is about one fifth of that of the correlative matrix for the real music data set. This is because in the synthetic music data set, the average length of the repeating patterns is much shorter than that in the real music data set. According to the experiments, the average length of the longest repeating patterns of the real music objects is 70 while that of the synthetic music objects is 5.

Figure 4 and Figure 5 show the elapsed time versus the object size of music objects for the synthetic and real music data sets. The elapsed time of the string-join approach is shorter than that of the correlative matrix approach and both grow as the size of the music objects increases for both data sets.

Figure 6 and Figure 7 show the elapsed time versus the length of the longest repeating patterns of the music objects for the synthetic and real music data sets. As we have discussed in Section 3, the time complexity of finding the longest repeating pattern is log*N* where *N* is the length of the longest repeating pattern. Therefore, the performance of the string-join approach is much better than that of the other approaches when the length of the longest repeating pattern is large.

Our approach will suffer when there are many non-trivial repeating patterns whose length is very close to the length of the longest repeating patterns. For example, there are non-trivial repeating patterns of length 1, 2, 3, 4, 5, 6, 7, and 8 in the music feature string "AABABCABCDABCDE ABCDEFABCDEFGABCDEFGHABCDEFGH." However, for real music objects this case rarely occurs since the length of themes of a music object is typically much longer than that of the other non-trivial repeating patterns. For example, Figure 8 shows the distribution of the length of all non-trivial repeating patterns in the song "Five Hundred Miles". The length of the longest repeating pattern, which is its theme, is 112, while the length of the other non-trivial repeating patterns is less than or equal to 56 ($< 2^6 = 64$). Therefore, we do not need to check the non-trivial repeating patterns whose length is between 112 and 64 after the longest repeating pattern is found. Thus a large amount of execution time and storage space can be saved.

Figure 9 shows the elapsed time versus the number of non-trivial repeating patterns of the music objects for the real music data set. According to our experiments, the average number of all non-trivial repeating patterns in a music object is 77, which is much smaller than the average number of all repeating patterns in a music object, which is 4156. Since the suffix tree approach stores all repeating

patterns, the requirement of the storage space is easy to become unsatisfied, as can be seen in Figure 9.

5. Conclusion

Music is an important type of media. However, few work were focused on the music databases. Since repeating patterns can be found in both classical and pop music objects, we choose them as music features for content-based music data retrieval. In this paper, we propose an approach for extracting all non-trivial repeating patterns in the melody string of a music object. By repeatedly joining shorter repeating patterns using the string-join, the longest repeating pattern in a music feature string can be found. Then all non-trivial repeating patterns can be generated by removing trivial repeating patterns. According to our experiments, the execution time can be dramatically reduced compared with the correlative matrix approach and the suffix tree approach.

The musical semantics of the repeating patterns are currently being investigated. Issues on music data mining such as music data generalization, music data clustering, and music association rules are to be studied. Finally, since some *modifications* can be done on a motive by the composer to make the music sound more fruitful and variant, a repeating pattern may be approximately (not exactly) repeated. We will extend the proposed algorithm to consider this more general situation.

References:

[1] Abraham, G., ed., *The New Oxford History of Music, Vol. VIII and Vol. IX*, Oxford University Press, 1988.

[2] Apostolico A. , "The Myriad Virtues of Subwords Trees", in A. Apostolico and Z. Galil. Editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI* Series F: Computer and System Sciences, pages 97~107, Springer-Verlag, Berlin, Germany, 1984.

[3] Apostolico A. and F. Preparata, "Data Structures and Algorithms for the String Statistics Problem," *Algorithmica*, 15:481~494, 1996.

[4] Bakhmutova, V., V. D. Gusev, and T. N. Titkova, "The Search for Adaptations in Song Melodies," *Computer Music Journal*, Vol. 21, No. 1, pages 58~67, Spring 1997.

[5] Balaban, M., "The Music Structures Approach to Knowledge Representation for Music Processing," *Computer Music Journal*, Vol. 20, No. 2, pages 96~111, Summer 1996.

[6] Berndt, D. J., and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," in *Proceedings of AAAI-94 Workshop on Knowledge Discovery in Databases*, 1994, pages 359~370.

[7] Chen, J. C. C. and A. L. P. Chen, "Query by Rhythm: An Approach for Song Retrieval in Music Databases," in *Proc. of 8th Intl. Workshop on Research Issues in Data Engineering*, pages 139~146, 1998.

[8] Chen M. T. and J. Seiferas, "Efficient and Elegant Subword-Tree Construction," in A. Apostolico and Z. Galil. Editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI* Series F: Computer and System Sciences, pages 97~107, Springer-Verlag, Berlin, Germany, 1984.

[9] Chou, T. C., A. L. P. Chen, and C. C. Liu, "Music Databases: Indexing Techniques and Implementation," in *Proc. of IEEE Intl. Workshop on Multimedia Data Base Management System*, 1996.

[10] Davenport, G., T.A. Smith, and N. Pincever. "Cinematic Primitives for Multimedia," *IEEE Computer Graphics & Applications*, pages 67~74, July 1991.

[11] Day, Y. F., S. Pagtas, M. Iino, A. Khokhar, and A. Ghafoor, "Object-Oriented Conceptual Modeling of Video Data," in *Proc. of IEEE Data Engineering*, pages 401~408, 1995.

[12] Dowling, W. J., "Pitch Structure," in *Representing Musical Structure*, P. Howell, R. West, and I. Cross, ed., Academic Press, London, 1991.

[13] 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, pages 231~236.

[14] Hjelsvold, R. and R. Midtsraum, "Modeling and Querying Video Data," in *Int'l Conf. on Very Large Data Bases*, pages 686~694, 1994.

[15] Hsu, J. L., C. C. Liu, and Arbee, L. P. Chen, "Efficient Repeating Pattern Finding in Music Databases," in *Proc. of Seventh International Conference on Information and Knowledge Management* (*CIKM'98*), 1998.

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

[17] Krumhansl, C. L., *Cognitive Foundations of Musical Pitch*, Oxford University Press, New York, 1990.

[18] Leman, M., "The Ontogenesis of Tone Semantics: Results of a Computer Study," in *Music and Connectionism*, P. Todd and G. Loy, ed., MIT Press, Cambridge, 1991.

[19] Leman, M. "Tone Context by Pattern Integration over Time," in *Readings in Computer-Generated Music*, D. Baggi , ed., IEEE Computer Society Press, Los Alamitos, 1992.

[20] Li, J. Z. et al., "Modeling of Video Spatial Relationships in an Object Database Management," in *Proc. of IEEE Workshop on Multimedia Database Management systems*, pages 124~132, August 1996.

[21] Liu, C. C., J. L. Hsu, and A. L. P. Chen, "1D-List: An Approximate String Matching Algorithm for Content-Based Music Data Retrieval," submitted for publication.

[22] MIDI Manufacturers Association (MMA), *MIDI 1.0 Specification*, http://www.midi.org/.

[23] McCreight E. M., "A Space Economical Suffix Tree Construction Algorithm," *J. Assoc. Comput. Mach.*, 23:262~272, 1976.

[24] Narmour, E., *The Analysis and Cognition of Basic Melodic Structures*, The University of Chicago Press, Chicago, 1990.

[25] Oomoto, E., and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Transactions on Knowledge and Data Engineering*, pages 629~643, August 1993.

[26] Pfeiffer, S., S. Fischer, and W. Effelsberg, "Automatic Audio Content Analysis," in *Proceedings of the Fourth ACM International Multimedia Conference*, 1996, pages 21~30.

[27] Prather, R. E., "Harmonic Analysis from the Computer Representation of a Musical Score," *Communication of the ACM*, Vol. 39, No. 12, Dec. 1996, pages 119, (pages 239~255 of *Virtual Extension Edition of CACM*).

[28] Sankoff, D., and J. B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Addison-Wesley Publishing Company, 1983.

[29] Smoliar, S. W. and HongJiang Zhang, "Content-Based Video Indexing and Retrieval," *IEEE Multimedia Magazine*, pages 62~72, 1994.
[30] Sundberg, J., A. Friberg, and L. Fryden, "Common Secrets of Musicians and Listeners: An Analysis-by-synthesis Study of Musical Performance," in *Representing Musical Structure*, P. Howell, R. West, and I. Cross, ed., Academic Press, London, 1991.
[31] Szpankowski W., "Asymptotic Properties of Data Compression and Suffix Trees, " *IEEE Trans. on Information Theory*, Vol. 39, No. 5, Sep. 1993.
[32] Ukkonen E., "On-Line Construction of Suffix Tree," *Algorithmica*, 14:249~260, 1995.
[33] Weiner P., "Linear pattern matching algorithms," in *Proc. of IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1~11, 1973.
[34] Wold, E., T. Blum, D. Keislar, and J. Wheaton, "Content-based Classification, Search, and Retrieval of audio," *IEEE Multimedia*, Vol. 3, No. 3, Fall 1996, pages 27~36.

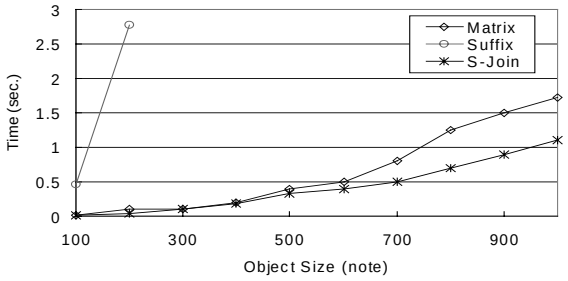

Figure 4: Elapsed time vs. object size of music objects for the synthetic data set.

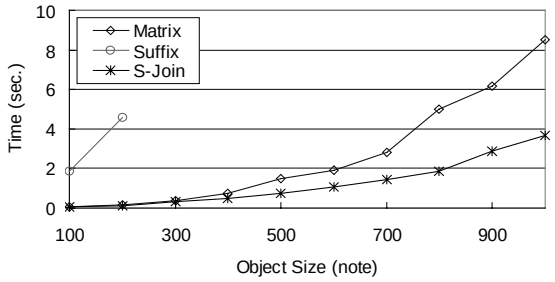

Figure 5: Elapsed time vs. object size of music objects for the real data set.

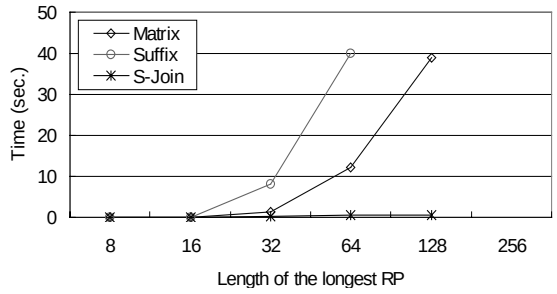

Figure 6: Elapsed time vs. length of the longest repeating patterns of the music objects for the synthetic data set.

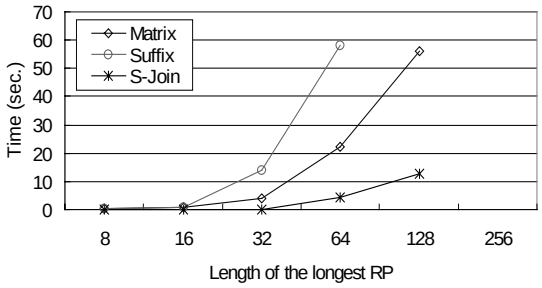

Figure 7: Elapsed time vs. length of the longest repeating patterns of the music objects for the real data set.

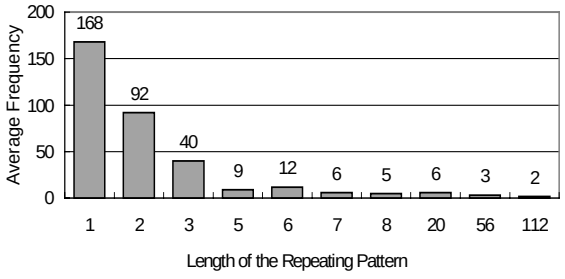

Figure 8: The distribution of the length of all non-trivial repeating patterns in the song "Five Hundred Miles".

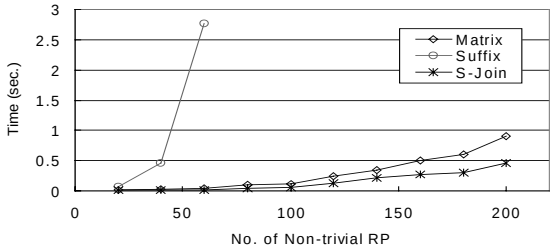

Figure 9: Elapsed time vs. number of non-trivial repeating patterns of the music objects for the real music data set.