Discovering Nontrivial Repeating Patterns in Music Data

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

***Abstract*—A repeating pattern in music data is defined as a sequence of notes which appears more than once in a music object. The themes are a typical kind of repeating patterns. The themes and other nontrivial repeating patterns are important music features which can be used for both content-based retrieval of music data and music data analysis. In this paper, we propose two approaches for fast discovering nontrivial repeating patterns in music objects. In the first approach, we develop a data structure called correlative matrix and its associated algorithms for extracting the repeating patterns. In the second approach, we introduce a string-join operation and a data structure called RP-tree for the same purpose. Experiments are performed to compare these two approaches with others. The results are further analyzed to show the efficiency and the effectiveness of our approaches.**

***Index Terms*—Content-based music data retrieval, music databases, music feature extraction, repeating patterns.**

I. Introduction

RAPID progress of hardware and software technologies makes it possible to manage and access large volumes of multimedia data. The indexing and searching techniques for multimedia data are key issues in the area of multimedia databases. While most research work focus on the content-based retrieval of image and video data [1], [6], [30], [31], [41], less attention has been received to the content-based retrieval of audio and music data. Recently, issues regarding content-based audio data retrieval have been studied [14], [25], [32], [40]. Foote made a survey on audio information retrieval [14]. Wold *et al.* proposed an approach to retrieve audio objects based on their contents [40]. 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 derived from the raw data. The N-vector of the acoustical features is then used to classify sounds for similarity searching. Pfeiffer *et al.* [32] made use of acoustical features of audio data to partition an audio stream into segments and classify them into speech, music, silence, and other sounds. Lohr and Rakow [25] developed functions needed to store and manipulate audio data. Some useful operations for audio data such as *dynamic compression* and *low-pass filtering* are provided.

Previous contributions in the field of music databases are introduced as follows. Ghias *et al.* [15] proposed an approach for modeling the content of music objects. A music object is transformed into a string which consists of three kinds of symbols ("U," "D," and "S" which represent a *note*[1] is higher than, lower than, or the same as its previous note, respectively). The problem of music data retrieval is then transformed into that of approximate string matching. Prather [33], [42] proposed a linked list data structure to store the *scores* of music data. By applying the *harmonic analysis method* to traverse the scores, the *chord* information can be derived. In [7], a system supporting the content-based navigation of music data was presented. A sliding window is applied to cut a music contour into subcontours. All subcontours are organized as an index structure. Previous works on music data retrieval assumed that the music is *monophonic*, and indeed most MIDI files are *polyphonic*. Uitdenbogerd and Zobel [37] proposed a method for extracting the monophonic *melody* from a polyphonic music object in MIDI format.

To develop a content-based music database system, we have implemented a system called *muse* [10], [12], [21]. In this system, we used various methods for content-based music data retrieval. 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 [21]. Similarity measures in the approximate string matching algorithm are based on music theory. Moreover, we consider music objects and music queries as sequences of chords [12]. An index structure is developed to provide efficient partial matching capability. In [10], we propose an approach for retrieving music objects by rhythm. In this approach, the rhythm of a music object is modeled by rhythm strings which consist 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 [15], melody [6], [21], chord [12], or rhythm [10], string matching is the core technique for the music query processing whose performance is dependent on the length of music objects to be matched. Therefore, if the music objects are large,

Manuscript received November 15, 1999; revised June 21, 2001. This work was supported in part by the Ministry of Education, R.O.C., under Contract MOE 89-E-FA04-1-4. The associate editor coordinating the review of this paper and approving it for publication was Prof. Mohammed Ghanbari.

J.-L. Hsu and A. L. P. Chen are with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C. (e-mail: alpchen@cs.nthu.edu.tw).

C.-C. Liu was with National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C. He is now with the Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu 300, Taiwan, R.O.C. (e-mail: ccliu@chu.edu.tw).

Publisher Item Identifier S 1520-9210(01)07787-2.

[1]For the definition of the music terminologies used in this paper, please refer to Appendix A.

the execution time for query processing may become unacceptable. By examining the score of a music object, we find many sequences of notes appearing more than once in the music object. We call these sequences *repeating patterns*. For example, the well-known *motive* "sol-sol-sol-mi" in Beethoven's Symphony no. 5 repeatedly appears in the music object. Many researchers in musicology and music psychology fields agree that repetition is a universal characteristic in music structure modeling [4], [20], [29], [35]. Since the length of repeating patterns is much shorter than that of a music object, to represent the music object by its repeating patterns meets both efficient and semantic requirements for content-based music data retrieval. Therefore, techniques for finding the repeating patterns from the sequence of notes of a music object are needed to be developed.

A *suffix tree* is a data structure which is able to represent all suffixes of a string [11], [28]. It is originally developed for substring matching [28]. Since all suffixes and their appearances in a string are stored in a suffix tree, it can also be used to find the repeating patterns. However, much storage space and execution time are required for traversing the suffix tree to generate the repeating patterns of a given string.

In this paper, we extend our previous works [16], [22], and propose two approaches to efficiently discover the repeating patterns of music objects. For the first approach, the repeating patterns are found based on a data structure called *correlative matrix*. For a music object of n notes, an $(n \times n)$ correlative matrix is constructed to keep the intermediate results during the finding process. In the other approach, the longer repeating pattern of a music object is discovered by repeatedly combining shorter repeating patterns by a *string-join* operation. Thus the storage space and execution time can be reduced. The major extensions of this paper include the discussion of various features of music data, the complete procedures of the two approaches, and a comprehensive performance evaluation.

The paper is organized as follows. We present various music features and introduce the semantics of the repeating patterns in Section II. In Section III, we first go through examples to show the basic idea of the correlative matrix approach for finding repeating patterns. Then, the algorithms for finding repeating patterns are formally described. In Section IV, we motivate the string-join approach and formally describe its related algorithms. Extensive experiments are performed. In Section V, the results are presented and analyzed to show the efficiency and effectiveness of the approaches. Finally, Section VI concludes this paper and presents our future research directions.

II. Music Features and Repeating Patterns

For content-based music data retrieval, one of the techniques needed is to extract music features from the raw data of music objects and organize them as the music index for further processing.

A. Music Features

According to the music characteristics, the music features can be classified into four categories: *1) static music information, 2) acoustical feature, 3) thematic feature, and 4) structural feature.*

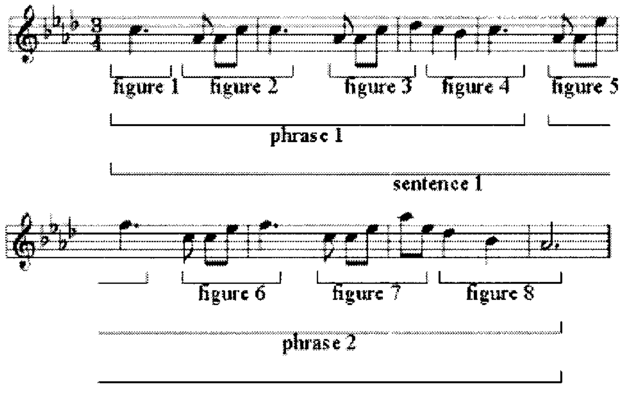

Fig. 1. Brahms Waltz in A flat.

1) Static Music Information: Static music information refers to the intrinsic music characteristics of music objects such as *key*, *beat*, and *tempo*. For example, the Beethoven's Symphony no. 5, Op. 67, is in C minor, 4/4, allegro con brio. This information can be regarded as keywords or descriptive attributes of a music object.

2) Acoustical Feature: Acoustical features include loudness, pitch, duration, bandwidth, and brightness, which can be automatically derived from the raw data of music objects. For example, the loudness of a music object is the root mean square value in *decibels* of its audio signals [40]. The acoustical features of a music object can be represented as a feature vector or feature point in a multi-dimensional feature space [23].

3) Thematic Feature: Thematic features such as melodies, rhythms, and chords are derived from the score information of a music object. Thematic features can be represented in a string form. For example, "sol-sol-sol-mi" is a melody string, "0.5-0.5-0.5-2" is a rhythm string, and "C-Am-Dm-G7" is a chord string. The melody strings, the rhythm strings, and the chord strings can be easily derived from the music objects in the MIDI form [27], [33], [37], [42].

4) Structural Feature: The classic music objects are composed according to a special structure called *musical form* in which there are two basic rules: 1) hierarchical rule and 2) repetition rule [19], [20], [29]. The hierarchical rule says music objects are formed hierarchically, i.e., a music object consists of *movements*, a movement consists of *sentences*, a sentence consists of *phrases*, and a phrase consists of *figures*. For example, Fig. 1 shows the music form of Brahms Waltz in A flat. It contains one sentence, i.e., sentence 1. Sentence 1 consists of phrase 1 and phrase 2. Both phrase 1 and phrase 2 consist of four figures. These figures, phrases, and sentence form a hierarchical structure.

The repetition rule says that some sequences of notes, known as motives, repeatedly appear in a movement. For example, both sequences "C6-Ab5-Ab5-C6" and "F6-C6-C6-Eb6" repeat once in Fig. 1. The repetition rule also applies to pop music.

B. Nontrivial Repeating Patterns

A *theme* is considered one of the most expressive and representative parts of a music object, which can be represented by thematic feature using melody and rhythm strings. The refrains in a pop song and the motives of the classic music are typical themes. A theme usually repeats many times in the music objects, which follows the repetition rule in the musical form. Therefore, theme, motive, and refrain are kinds of repeating patterns. To model a music object by thematic and structural features, the repeating patterns must be found first. We formally define a repeating pattern as follows.

Definition 1: For a substring X of a sequence of notes $\mathbf{S}$, if X appears more than once in $\mathbf{S}$, we call X a *repeating pattern* of $\mathbf{S}$. The *repeating frequency* of the repeating pattern X, denoted as $freq(\mathrm{X})$, is the number of appearances of X in $\mathbf{S}$. The length of the repeating pattern X, denoted $|\mathrm{X}|$, is the number of notes in X.

A note, as defined in [9], is a single symbol on a musical score, indicating the pitch and duration of what is to be sung and played. For easier presentation of our approaches, we only use the pitch information of a note, i.e., consider a melody string instead of a sequence of notes to discover the repeating patterns of a music object in the paper.

Example 1: Consider the melody string "C-D-E-F-C-D-E-C-D-E-F," this melody string has ten repeating patterns, as shown in Table I.

From Table I, "C-D-E" is a repeating pattern with a frequency three, so is "C-D." That is, "C-D" repeats only in the place where "C-D-E" repeats. This kind of repeating patterns can be directly derived and need not be reported in the process of repeating pattern discovery.

Definition 2: A repeating pattern X is *nontrivial* if and only if there does not exist another repeating pattern Y such that $freq(\mathrm{X}) = freq(\mathrm{Y})$ and X is a substring of Y.

In Example 1, The repeating patterns "D-E-F," "E-F," and "F" are substrings of the repeating pattern "C-D-E-F" and $freq$("C-D-E-F") = $freq$("D-E-F") = $freq$("E-F") = $freq$("F") = 2. Similarly, the repeating patterns "C-D," "D-E," "C," "D" and "E" are substrings of the repeating pattern "C-D-E" and $freq$("C-D-E") = $freq$("C-D") = $freq$("D-E") = $freq$("C") = $freq$("D") = $freq$("E") = 3. Among the ten repeating patterns of the melody string, only "C-D-E-F" and "C-D-E" are *nontrivial*.

III. The Correlative-Matrix Approach

The correlative-matrix approach is the first method to efficiently find nontrivial repeating patterns for a given melody string.

A. Approach Overview

In this subsection, we illustrate the basic idea of the approach through an example. The formal algorithms are described in the next subsection.

Example 2: We excerpt a phrase with 12 notes from Brahms Waltz in A flat, as shown in Fig. 2. Its corresponding melody string $\mathbf{S}$ is "C6-Ab5-Ab5-C6-C6-Ab5-Ab5-C6-Db5-C6-Bb5-C6." There are three repeating patterns in this phrase, as shown in Table II. Note that the proper substrings of a repeating pattern are not included except they also appear elsewhere in the phrase.

TABLE I
All Repeating Patterns of the Melody String "C-D-E-F-C-D-E-C-D-E-F" (RP: Repeating Pattern. RPF: Repeating Pattern Frequency)

RP	C-D-E-F	C-D-E	D-E-F	C-D	D-E
RPF	2	3	2	3	3
RP	E-F	C	D	E	F
RPF	2	3	3	3	2

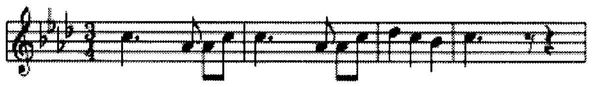

Fig. 2. Phrase excerpted from Brahms Waltz in A flat.

TABLE II
Repeating Patterns in the Phrase Excerpted From Brahms Waltz in A Flat (PL: Pattern Length)

RPF	PL (*notes*)	RP
2	4	C6-Ab5-Ab5-C6
6	1	C6
4	1	Ab5

For example, "C6-Ab5-Ab5-C6" is a repeating pattern and its repeating frequency is two. The repeating pattern "C6-Ab5-Ab5" is a proper substring of "C6-Ab5-Ab5-C6," whose repeating frequency is also two. Since "C6-Ab5-Ab5" appears only in the place where "C6-Ab5-Ab5-C6" appears, it is not listed in the table. However, if the substring also appears elsewhere in the phrase, it will be considered as a result. For example, "C6" is a repeating pattern whose repeating frequency is six, which means it appears more than "C6-Ab5-Ab5-C6" does in the phrase, therefore it is listed in the table.

To find all repeating patterns in the melody string $\mathbf{S}$, an intuitive method is to generate all substrings of $\mathbf{S}$. Then, each substring $\mathbf{P}$ of $\mathbf{S}$ will be compared with $\mathbf{S}$ to decide the number that $\mathbf{P}$ appears in $\mathbf{S}$. This method is simple but very inefficient: if the length of string $\mathbf{S}$ is n, the number of all substrings of the input string $\mathbf{S}$ is $n + (n-1) + \cdots + 1$. For a substring $\mathbf{P}$ of length m, it needs $O(m \times n)$ character comparisons to decide the number that $\mathbf{P}$ appears in $\mathbf{S}$. In the worst case, the total complexity will be $O(n^4)$. For example, for a music object consisting of 1000 notes, it will cost $O(10^{12})$ character comparisons to find all repeating patterns in the music object.

To make the process of finding repeating patterns more efficient, we use a matrix, called *correlative matrix*, to keep the intermediate results of substring matching. For the ith note and the jth note in the melody string, if they are the same, the cell in the ith row and the jth column of the correlative matrix will be set to one. Moreover, if the ith note equals the jth note and the $(i+1)$th note equals the $(j+1)$th note, which means there is a repeating pattern of length two, the cell in the $(i+1)$th row and the $(j+1)$th column of the correlative matrix will be set to two. Therefore, the value of each cell in the matrix denotes the length of a repeating pattern found. By computing the number of nonzero cells in the correlative matrix, the repeating frequencies of all repeating patterns can be derived.

We continue the Example 2 to illustrate the construction of the correlative matrix. For the example shown in Fig. 2, since there are 12 notes in the melody string $\mathbf{S}$, we initialize a 12×12 upper-triangular matrix, denoted by $\mathbf{T}$. We use $\mathbf{T}_{i,j}$ to indicate the cell of the ith row and jth column in the matrix $\mathbf{T}$.

The matrix will be filled row by row. For the first row, we compare the first note "C6" with each note in the melody string $\mathbf{S}$. For each note $\mathbf{S}_j$ (denote the jth note of $\mathbf{S}$), if $\mathbf{S}_j =$ "C6" which means "C6" repeatedly appear in the first and jth note in $\mathbf{S}$, we set $\mathbf{T}_{1,j} = 1$. In this example, since $\mathbf{S}_4$, $\mathbf{S}_5$, $\mathbf{S}_8$, $\mathbf{S}_{10}$, and $\mathbf{S}_{12}$ are "C6," $\mathbf{T}_{1,4}$, $\mathbf{T}_{1,5}$, $\mathbf{T}_{1,8}$, $\mathbf{T}_{1,10}$, and $\mathbf{T}_{1,12}$ are set to one.

Then we process the second note "Ab5." Since $\mathbf{S}_3$ is also "Ab5," $\mathbf{T}_{2,3}$ is set to one. For $\mathbf{T}_{2,6}$, since $\mathbf{S}_6 =$ "Ab5" and $\mathbf{T}_{1,5}$ is one, which implies the substring "C6-Ab5" repeatedly appears, $\mathbf{T}_{2,6}$ is set to two. $\mathbf{T}_{2,7}$ is set to one in the same way.

For any two notes $\mathbf{S}_i$ and $\mathbf{S}_j (i \neq j$ and $i, j > 1)$ in the melody string $\mathbf{S}$, if $\mathbf{S}_i = \mathbf{S}_j$, we set $\mathbf{T}_{i,j} = \mathbf{T}_{i-1,j-1} + 1$. If the value stored in $\mathbf{T}_{i,j}$ is n, it indicates a repeating pattern of length n appearing in the positions $(j - n + 1)$ to j in $\mathbf{S}$. Table III shows the result after all notes are processed.

After we construct the correlative matrix, the next step is to find all repeating patterns and their repeating frequencies. The *candidate set*, denoted $\boldsymbol{CS}$, is employed for this purpose. Each element in the candidate set $\boldsymbol{CS}$ is of the form (*pattern*, *rep_count*, *sub_count*), where *pattern*, *rep_count*, and *sub_count* represent a repeating pattern, the count of matching to the repeating pattern, and the number of the repeating pattern being a proper substring of the other repeating patterns, respectively. The *rep_count* is used to calculate the repeating frequency. The *sub_count* information is used to check whether this repeating pattern only appears as a proper substring of another repeating pattern (i.e., *rep_count* = *sub_count*). If that is the case, the repeating pattern will not be considered as a result.

The candidate set is initially set to an empty set. For every nonzero cell $\mathbf{T}_{i,j}$ in the correlative matrix $\mathbf{T}$, its corresponding repeating patterns and the associated counts are computed and inserted into the candidate set $\boldsymbol{CS}$. According to the conditions $[(\mathbf{T}_{i,j} = 1)$ or $(\mathbf{T}_{i,j} > 1)]$ and $[(\mathbf{T}_{(i+1),(j+1)} = 0)$ or $\mathbf{T}_{(i+1),(j+1)} \neq 0)]$, there are four cases.

Case 1: $(\mathbf{T}_{i,j} = 1$ and $\mathbf{T}_{(i+1),(j+1)} = 0)$. For example, the value of $\mathbf{T}_{1,4}$ is one, which indicates there is a matching to the corresponding substring "C6." Moreover, the value of $\mathbf{T}_{2,5}$ is zero, which means "C6" is not a proper substring of another repeating pattern at this position. We insert ("C6," 1, 0) into the candidate set $\boldsymbol{CS}$.

Case 2: $(\mathbf{T}_{i,j} = 1$ and $\mathbf{T}_{(i+1),(j+1)} \neq 0)$. For example, the value of $\mathbf{T}_{1,5}$ is one, which indicates there is a matching to the corresponding substring "C6." Moreover, the repeating frequency of $\mathbf{T}_{2,6}$ is not zero, which means "C6" is a proper substring of another repeating pattern ("C6-Ab5") here. Since "C6" is already in the candidate set $\boldsymbol{CS}$, we modify ("C6," 1, 0) into ("C6," 2, 1) to record that another repetition of "C6" is found.

Case 3: $(\mathbf{T}_{i,j} > 1$ and $\mathbf{T}_{(i+1),(j+1)} \neq 0)$. For example, the value of $\mathbf{T}_{\mathbf{2},\mathbf{6}}$ is two, which indicates the substrings "C6-Ab5" and "Ab5" repeatedly appear here, we should insert ("C6-Ab5," 1, 0) and ("Ab5," 1, 0) into the candidate set $\boldsymbol{CS}$. However, since $\mathbf{T}_{3,7}$ is three, which indicates the two repeating patterns "C6-Ab5" and "Ab5" are proper substrings of the repeating pattern "C6-Ab5-Ab5." We insert ("C6-Ab5," 1, 1) and ("Ab5," 1, 1) into the candidate set $\boldsymbol{CS}$ instead.

TABLE III
CORRELATIVE-MATRIX AFTER ALL NOTES ARE PROCESSED

	C6	Ab5	Ab5	C6	C6	Ab5	Ab5	C6	Db5	C6	Bb5	C6
C6	—			1	1			1		1		1
Ab5		—	1			2	1					
Ab5			—			1	3					
C6				—	1			4		1		1
C6					—			1		1		1
Ab5						—	1					
Ab5							—					
C6								—		1		1
Db5									—			
C6										—		1
Bb5											—	
C6												—

Case 4: $(\mathbf{T}_{i,j} > 1$ and $\mathbf{T}_{(i+1),(j+1)} = 0)$. For example, the value of $\mathbf{T}_{\mathbf{4},\mathbf{8}}$ is four, which indicates all of the four substrings "C6-Ab5-Ab5-C6," "Ab5-Ab5-C6," "Ab5-C6," and "C6," repeatedly appear here. Moreover, since $\mathbf{T}_{5,9}$ is zero, which indicates the repeating pattern "C6-Ab5-Ab5-C6" is not a proper substring of another repeating pattern here. We insert ("C6-Ab5-Ab5-C6," 1, 0), ("Ab5-Ab5-C6," 1, 1), and ("Ab5-C6," 1, 1) into the candidate set $\boldsymbol{CS}$ and change ("C6," 6, 1) into ("C6," 7, 2).

After examining every nonzero cell in the correlative matrix, the candidate set $\boldsymbol{CS}$ becomes {("C6," 15, 2), ("Ab5," 6, 2), ("C6-Ab5," 1, 1), ("C6-Ab5-Ab5," 1, 1), ("Ab5-Ab5," 1, 1), ("C6-Ab5-Ab5-C6," 1, 0), ("Ab5-Ab5-C6," 1, 1), and ("Ab5-C6," 1, 1)}. There are two more tasks we have to do. First, if a repeating pattern has the same *rep_count* and *sub_count*, it will be removed from the candidate set $\boldsymbol{CS}$. In this example, the elements ("C6-Ab5," 1, 1), ("C6-Ab5-Ab5," 1, 1), ("Ab5-Ab5," 1, 1), ("Ab5-Ab5-C6," 1, 1), and ("Ab5-C6," 1, 1) are removed.

Second, we calculate the repeating frequency for every repeating pattern found. For the repeating pattern "C6," there are 15 nonzero cells associated with it in the correlative matrix. However, the repeating frequency of "C6" is six. It is because for a repeating pattern whose repeating frequency is f, there will be $C_2^f = f(f-1)/2$ matchings associated with this repeating pattern when constructing the correlative matrix. Each matching causes the value of the corresponding *rep_count* increased by one. Therefore, for each element (*pattern*, *rep_count*, *sub_count*) in the candidate set $\boldsymbol{CS}$, the repeating frequency f of the repeating pattern can be derived by the following equation:

$$rep\_count = \frac{1}{2} f(f-1), \text{ i.e.,}$$
$$f = \left(1 + \sqrt{1 + 8 \times rep\_count}\right) \Big/ 2. \quad (1)$$

For example, since the *rep_count* of the element ("C6," 15, 1) is 15, the repeating frequency f of the repeating pattern "C6" will be $f = (1 + \sqrt{1 + 8 \times 15})/2 = 6$. Similarly, the repeating frequencies of the repeating patterns "Ab5" and "C6-Ab5-Ab5-C6" are 4 and 2, respectively.

B. Algorithms for the Correlative-Matrix Approach

In this subsection, we formally describe algorithms of the correlative-matrix approach for finding all nontrivial repeating patterns in a music object. Assume that the input is a sequence of notes. The result is a set of repeating patterns and their repeating frequencies, $\boldsymbol{RP} = \{(p_1, f_1), (p_2, f_2), \ldots, (p_i, f_i)\}$, where p_i is a repeating pattern and f_i is its repeating frequency. The *Find_RP_CM* algorithm, shown in Algorithm C1, consists of two major phases: constructing the correlative matrix $\mathbf{T}$ for a given music feature string and generating the repeating pattern set $\boldsymbol{RP}$ from the correlative matrix $\mathbf{T}$. The details of the two phases are explained as follows.

```
Algorithm C1 Find_RP_CM(S, RP)
/* input: the music feature string S[1..n] */
/* output: the set of all nontrivial repeating
  patterns and their repeating frequencies
  $RP = \{(p_1, f_1), (p_2, f_2), \ldots, (p_i, f_i)\}$ */
Begin
1. Construct_Correlative_Matrix (S, T)
2. Generate_Repeating_Pattern (T, RP)
End
```

Let $\mathbf{S}_i$ denote the ith note in the input string $\mathbf{S}$ of length n. An $n \times n$ correlative matrix $\mathbf{T}$ is constructed by the *Construct_Correlative_Matrix* procedure (Algorithm C2) included in Appendix B. The correlative matrix keeps all positions where a repetition of a substring takes place. The correlative matrix $\mathbf{T}$ is initialized to an $n \times n$ zero matrix. Then, for the first row of $\mathbf{T}$, if $\mathbf{S}_1 = \mathbf{S}_j$, $\mathbf{T}_{1,j}$ is set to one, for $2 \le j \le n$. Next, every remaining cell $\mathbf{T}_{i,j}$ of the upper triangular part of $\mathbf{T}$ is filled by the following equations:

$$\mathbf{T}_{i,j} = \begin{cases} \mathbf{T}_{i-1,j-1} + 1, & \text{if } \mathbf{S}_i = \mathbf{S}_j \\ \quad \text{for } 2 \le i \le (n-1),\ 3 \le j \le n, \text{ and } i < j & \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

After the correlative matrix is constructed, the next step is to generate all repeating patterns and calculate their repeating frequencies. We use $\mathbf{S}[a:b]$ to denote the substring of the string $\mathbf{S}$ from the ath note to the bth note. The candidate set $\boldsymbol{CS}$ is initialized to an empty set. For every cell $\mathbf{T}_{i,j}$ in the correlative matrix $\mathbf{T}$, if $\mathbf{T}_{i,j}$ is greater than zero, it denotes that the corresponding pattern $P = \mathbf{S}[(j - \mathbf{T}_{i,j} + 1) : j]$ and all its substrings are repeating patterns. However, only the *suffix strings* of P and P itself are needed to be processed here, since the other substrings of P, if any, will be checked when processing other cells. For each suffix string *pat* of P, we accumulate its count of matching and store it in the candidate set $\boldsymbol{CS}$. There are four cases (Algorithm C3, lines 6 $\sim$ 21):

Case 1) If $\mathbf{T}_{(i+1),(j+1)} = 0$ and *pat* does not exist in $\boldsymbol{CS}$, insert (*pat*, 1, 0) into $\boldsymbol{CS}$.

Case 2) If $\mathbf{T}_{(i+1),(j+1)} = 0$ and *pat* exists in $\boldsymbol{CS}$, update (*pat*, *rep_ count*, *sub_count*) to (*pat*, *rep_count*+1, *sub_count*).

Case 3) If $\mathbf{T}_{(i+1),(j+1)} \neq 0$ and *pat* does not exist in $\boldsymbol{CS}$, insert (*pat*, 1, 1) into $\boldsymbol{CS}$.

Case 4) If $\mathbf{T}_{(i+1),(j+1)} \neq 0$ and *pat* exists in $\boldsymbol{CS}$, update (*pat*, *rep_count*, *sub_count*) to (*pat*, *rep_count*+1, *sub_count*+1).

After all repeating patterns are inserted into the candidate set $\boldsymbol{CS}$, the trivial results need to be removed and the repeating frequencies of all nontrivial repeating patterns calculated. As we have discussed, the former can be done by removing the element (*pattern*, *rep_count*, *sub_count*) with the same *rep_count* and *sub_count* (Algorithm C3, lines 26 $\sim$ 27), and the latter can be done by applying (1) (Algorithm C3, line 30). Finally, the nontrivial repeating patterns and their repeating frequencies in the candidate set $\boldsymbol{CS}$ are copied to the repeating pattern set $\boldsymbol{RP}$ as the result (Algorithm C3, line 31).

IV. The String-Join Approach

In the first subsection, we illustrate the basic idea of the string-join approach for finding nontrivial repeating patterns through examples. The formal algorithms are described in the next subsection.

A. Approach Overview

Consider all repeating patterns of the song "Five Hundred Miles." Its longest repeating pattern is its refrain "C-C-C-E-E-D-C-E-E-D-C-D-E-D-C-A-A-C-D-E-D-C-A-G-G-A-C-C." The length of the repeating pattern is 28. In the correlative-matrix approach, we find the repeating patterns incrementally based on the lengths. In this example, we will sequentially find the following repeating patterns: "C," "C-C," "C-C-C," "C-C-C-E," …, "C-C-C-E-E-D-C-E-E-D-C-D-E-D-C-A-A-C-D-E-D-C-A-G-G-A-C-C." This is inefficient since most substrings of the refrain are trivial. In this section, we propose an alternative approach for finding all nontrivial repeating patterns based on an exponential increase of the lengths of the repeating patterns.

For example, considering the melody string "C-D-E-F-C-D-E-C-D-E-F," we first scan this string to find all repeating patterns of length one. We use the form $\{\text{X}, freq(\text{X}), (position1, position2, \ldots)\}$ to represent the results, where $freq(\text{X})$, $position1$, 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 $\{$"C," 3, (1, 5, 8)$\}$, $\{$"D," 3, (2, 6, 9)$\}$, $\{$"E," 3, (3, 7, 10)$\}$, and $\{$"F," 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 subsection and we use an example to explain it here. Assume we want to check whether "C-D" is a repeating pattern. Since we know "C" and "D" appear at (1, 5, 8) and (2, 6, 9), respectively, we can confirm that "C-D" also appears at (1, 5, 8). This can be represented in the following equation:

$$\begin{aligned} &\{\text{“C,”}\ 3,\ (1,\ 5,\ 8)\}\infty\{\text{“D,”}\ 3,\ (2,\ 6,\ 9)\} \\ &\quad = \{\text{“C-D,”}\ 3,\ (1,\ 5,\ 8)\}. \end{aligned}$$

Similarly, we have

$$\{\text{“D,”}\ 3,\ (2,\ 6,\ 9)\}\infty\{\text{“E,”}\ 3,\ (3,\ 7,\ 10)\} = \{\text{“D-E,”}\ 3,\ (2,\ 6,\ 9)\},$$

and

$$\{\text{“E,”}\ 3,\ (3,\ 7,\ 10)\}\infty\{\text{“F,”}\ 2,\ (4,\ 11)\} = \{\text{“E-F,”}\ 2,\ (3,\ 10)\}.$$

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

$$\{\text{“C-D,”}\ 3,\ (1,\ 5,\ 8)\}\infty\{\text{“E-F,”}\ 2,\ (3,\ 10)\} = \{\text{“C-D-E-F,”}\ 2,\ (1,\ 8)\}.$$

Since $freq$(“C-D-E-F”) $=$ $freq$(“E-F”) $= 2$, we can conclude that not only “E-F” is trivial, so is “D-E-F”; otherwise, $freq$(“E-F”) must be greater than two. We only have to join “C-D” and “D-E” to check whether “C-D-E” is a repeating pattern. The result is {“C-D-E,” 3, (1, 5, 8)}. Since $freq$(“C-D-E”) $>$ $freq$(“C-D-E-F”), “C-D-E” is nontrivial. Therefore, the nontrivial repeating patterns in “C-D-E-F-C-D-E-C-D-E-F” are “C-D-E-F” and “C-D-E.”

B. *Algorithms for the String-Join Approach*

In this section, we formally describe the string-join approach for finding all nontrivial 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 3: Assume $\{\alpha_1\alpha_2\cdots\alpha_m,\ freq(\alpha_1\alpha_2\cdots\alpha_m),\ (p_1,\ p_2,\ \ldots,\ p_i)\}$ and $\{\beta_1\beta_2\cdots\beta_n,\ freq(\beta_1\beta_2\cdots\beta_n),\ (q_1,\ q_2,\ \ldots,\ q_j)\}$ 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

$$\begin{aligned}&\{\alpha_1\alpha_2\cdots\alpha_m,\ freq(\alpha_1\alpha_2\cdots\alpha_m),\ (p_1,\ p_2,\ \ldots,\ p_i)\}\\ &\quad\infty_k\{\beta_1\beta_2\cdots\beta_n,\ freq(\beta_1\beta_2\cdots\beta_n),\ (q_1,\ q_2,\ \ldots,\ q_j)\}\\ &\quad= \{\gamma_1\gamma_2\cdots\gamma_l,\ freq(\gamma_1\gamma_2\cdots\gamma_l),\ (o_1,\ o_2,\ \ldots,\ o_h)\}\end{aligned}$$

where

1) $i = freq(\alpha_1\alpha_2\cdots\alpha_m)$, $j = freq(\beta_1\beta_2\cdots\beta_n)$, $h = freq(\gamma_1\gamma_2\cdots\gamma_l)$
2) $\gamma_t = \alpha_t$, for $1 \leq t \leq m$,
3) $\gamma_t = \beta_{t-m+k}$, for $m+1 \leq t \leq l = m+n-k$,
4) $o_t = x = y - m + k$, 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 $1 \leq s \leq k$.

The algorithm of the string-join approach for finding all nontrivial repeating patterns is shown in Algorithm S1. It consists of two parts: in the first part (lines 1∼8), the repeating patterns of length one (line 2), the repeating patterns of length 2^k where $k > 0$ (lines 3∼8), and the longest repeating pattern (line 9) are found. In the second part, we remove the repeating patterns which are trivial (lines 10∼11) and generate the other nontrivial repeating patterns (line 12). The algorithm is explained in detail as follows. All the procedures used in the algorithm are included in Appendix B.

The repeating patterns of length one can be found by scanning the music feature string, described in the procedure of *Find_RP1* (Algorithm S2). 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 $k = 1, 2, \ldots, m$, where m is the largest number such that $2^m \leq L$. The process is done by the *Find_RP2K* procedure (Algorithm S3). After the repeating patterns of length 2^m are found, we can generate the longest repeating pattern $\gamma_1\gamma_2\cdots\gamma_L$ by joining two repeating patterns of length 2^m: $\{\gamma_1\gamma_2\cdots\gamma_L,\ freq(\gamma_1\gamma_2\cdots\gamma_L),\ (o_1,\ o_2,\ \ldots,\ o_h)\} = \{\alpha_1\alpha_2\cdots\alpha_{2^m},\ freq(\alpha_1\alpha_2\cdots\alpha_{2^m}),\ (p_1,\ p_2,\ \ldots,\ p_i)\} \infty_\lambda \{\beta_1\beta_2\cdots\beta_{2^m},\ freq(\beta_1\beta_2\cdots\beta_{2^m}),\ (q_1,\ q_2,\ \ldots,\ q_j)\}$, 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 $(2\times 2^m - (2^{m+1} - L)) = L$. 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 to use a binary search order in the generating process, i.e., the procedure *Find_Longest_RP* (Algorithm S4).

```
Algorithm S1 Find_RP_SJ(S, RP)
/*  input: the music feature string S */
/*  output: the set of all nontrivial re-
  peating patterns and their frequencies in
  S */
Begin
1.  RP = $\varnothing$
2.  RP[1] = Find_RP1(S)
3.  k = 0
4.  while RP[$2^k$] <> $\varnothing$ 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_Nontrivial_RP(RP)
End
```

Example 3: Assume there is a music feature string $\mathbf{S}$ = ABCDEFGHABCDEFGHIJABC. To find all nontrivial repeating patterns in $\mathbf{S}$, we first apply the procedure *Find_RP1* (Algorithm S2) on $\mathbf{S}$. The result is the set of all length one repeating patterns as shown in the following:

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

Then we apply the procedure *Find_RP2K* (Algorithm S3) 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 $\mathbf{S}$

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

Similarly, by repeatedly applying the procedure *Find_RP2K* (Algorithm S3), we can derive RP[4] and RP[8]

$$\begin{aligned}\text{RP}[4] = \{&\{\text{ABCD}, 2, (1, 9)\}, \{\text{BCDE}, 2, (2, 10)\},\\ &\{\text{CDEF}, 2, (3, 11)\}, \{\text{DEFG}, 2, (4, 12)\},\\ &\{\text{EFGH}, 2, (5, 13)\}\}\\ \text{RP}[8] = \{&\{\text{ABCDEFGH}, 2, (1, 9)\}\}.\end{aligned}$$

For the next step, we apply the procedure *Find_Longest_RP* (Algorithm S4) on RP[8] to find the longest repeating pattern in $\mathbf{S}$. 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 $\boldsymbol{S}$.

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

Lemma 1: Let X be a repeating pattern of a music feature string $\mathbf{S}$. For each substring Y of X, $\mathit{freq}(\text{Y}) \geq \mathit{freq}(\text{X})$.

Proof: For each appearance of X in $\mathbf{S}$, there must exist at least a Y. Moreover, Y may appear in $\mathbf{S}$ at the position where X does not appear. Therefore, $\mathit{freq}(\text{Y}) \geq \mathit{freq}(\text{X})$.

Lemma 2: Let X be a repeating pattern of a music feature string $\mathbf{S}$. For each substring Y of X, Y is also a repeating pattern of $\mathbf{S}$.

Proof: According to Lemma 1, we have $\mathit{freq}(\text{Y}) \geq \mathit{freq}(\text{X}) \geq 2$. By Definition 1, Y is also a repeating pattern.

Theorem 1: Let X be a repeating pattern of a music feature string $\mathbf{S}$, Y be a substring of X, and Z be a substring of Y. If $\mathit{freq}(\text{X}) = freq(\text{Z})$, Y is trivial.

Proof: According to Lemma 2 and Lemma 1, Y is a repeating pattern and we have $\mathit{freq}(\text{Z}) \geq \mathit{freq}(\text{Y}) \geq \mathit{freq}(\text{X})$. If $\mathit{freq}(\text{X}) = freq(\text{Z})$, $\mathit{freq}(\text{Y})$ must be equal to $\mathit{freq}(\text{X})$. By Definition 2, Y is a trivial repeating pattern.

Theorem 1 can be used to determine whether the length 1, $2^1, \ldots, 2^k$ repeating patterns, which are found by the procedure *Find_RP1* and *Find_RP2K*, are nontrivial. To make the checking process more efficient, we introduce a tree structure called *RP-tree*. Each node in an 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 compact. The construction of an RP-tree is formally described in the procedure *Build_RP_Tree* (Algorithm S5).

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

After all repeating patterns whose length is a power of two are generated, the final step is to generate all nontrivial repeating patterns whose length is not a power of two. Assume $\{\delta_1\delta_2\cdots\delta_n,\ \mathit{freq}(\delta_1\delta_2\cdots\delta_n),\ (s_1, s_2, \ldots, s_g)\}$ is a nontrivial 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 nontrivial. Theorem 1 can be applied again in this case. The resultant RP-tree will contain all nontrivial repeating patterns. The final result can be generated by traversing the resultant RP-tree. The procedure *Generate_Nontrivial_RP* (Algorithm S7) formally describes the process for generating all nontrivial repeating patterns.

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

To refine this RP-tree, we apply procedure *Refine_RP_Tree* on this RP-tree. Since $\mathit{freq}(\text{A}) = \mathit{freq}(\text{B}) = \mathit{freq}(\text{C}) = \mathit{freq}(\text{AB}) = \mathit{freq}(\text{BC}) = 3$ and $\mathit{freq}(\text{D}) = \mathit{freq}(\text{E}) = \mathit{freq}(\text{F}) = \mathit{freq}(\text{G}) = \mathit{freq}(\text{H}) = \mathit{freq}(\text{DE}) = freq(EF) = \mathit{freq}(\text{FG}) = \mathit{freq}(\text{GH}) = 2$, 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 Fig. 4(a). Similarly, since $\mathit{freq}(\text{CD}) = \mathit{freq}(\text{DE}) = \mathit{freq}(\text{EF}) = \mathit{freq}(\text{FG}) = \mathit{freq}(\text{GH}) = \mathit{freq}(\text{CDEF}) = \mathit{freq}(\text{DEFG}) = \mathit{freq}(\text{EFGH}) = 2$, the repeating patterns CD, DE, EF, FG, and GH are trivial and their corresponding nodes are removed from the RP-tree, as shown in Fig. 4(b). Finally, since $\mathit{freq}(\text{ABCD}) = \mathit{freq}(\text{BCDE}) = \mathit{freq}(\text{CDEF}) = \mathit{freq}(\text{DEFG}) = \mathit{freq}(\text{EFGH}) = \mathit{freq}(\text{ABCDEFGH}) = 2$, the repeating patterns ABCD, BCDE, CDEF, DEFG, and EFGH are removed from the RP-tree, as shown in Fig. 4(c).

After the RP-tree is refined by removing the trivial repeating patterns whose length is a power of two, the final step is to generate all nontrivial repeating patterns. Apply procedure *Generate_Nontrivial_RP* (Algorithm S7) on the RP-tree shown in Fig. 4(c), a new repeating pattern $\{\text{ABC}, 3, (1, 9, 19)\}$ is generated by joining $\{\text{AB}, 3, (1, 9, 19)\}$ and $\{\text{BC}, 3, (2, 10, 20)\}$

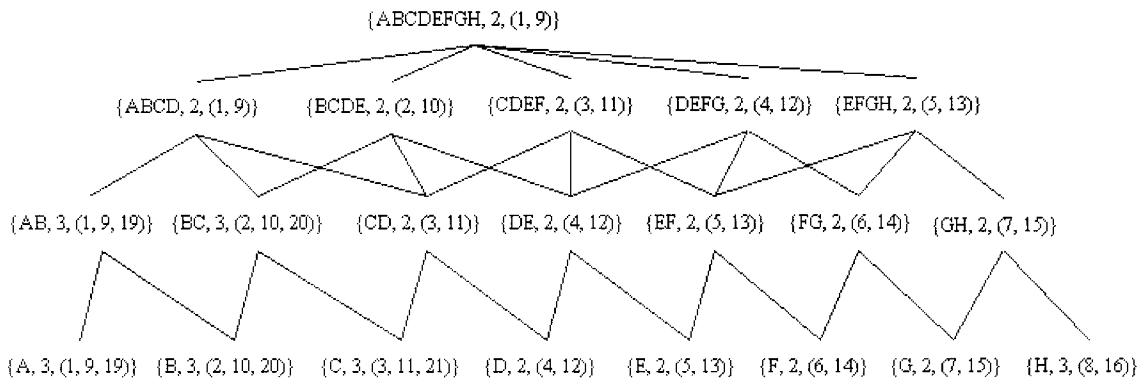
{ABCDEFGH, 2, (1, 9)}
{ABCD, 2, (1, 9)} {BCDE, 2, (2, 10)} {CDEF, 2, (3, 11)} {DEFG, 2, (4, 12)} {EFGH, 2, (5, 13)}
{AB, 3, (1, 9, 19)} {BC, 3, (2, 10, 20)} {CD, 2, (3, 11)} {DE, 2, (4, 12)} {EF, 2, (5, 13)} {FG, 2, (6, 14)} {GH, 2, (7, 15)}
{A, 3, (1, 9, 19)} {B, 3, (2, 10, 20)} {C, 3, (3, 11, 21)} {D, 2, (4, 12)} {E, 2, (5, 13)} {F, 2, (6, 14)} {G, 2, (7, 15)} {H, 3, (8, 16)}

Fig. 3. RP-tree for the music feature string $\mathbf{S}$ = ABCDEFGHABCDEFGHIJABC.

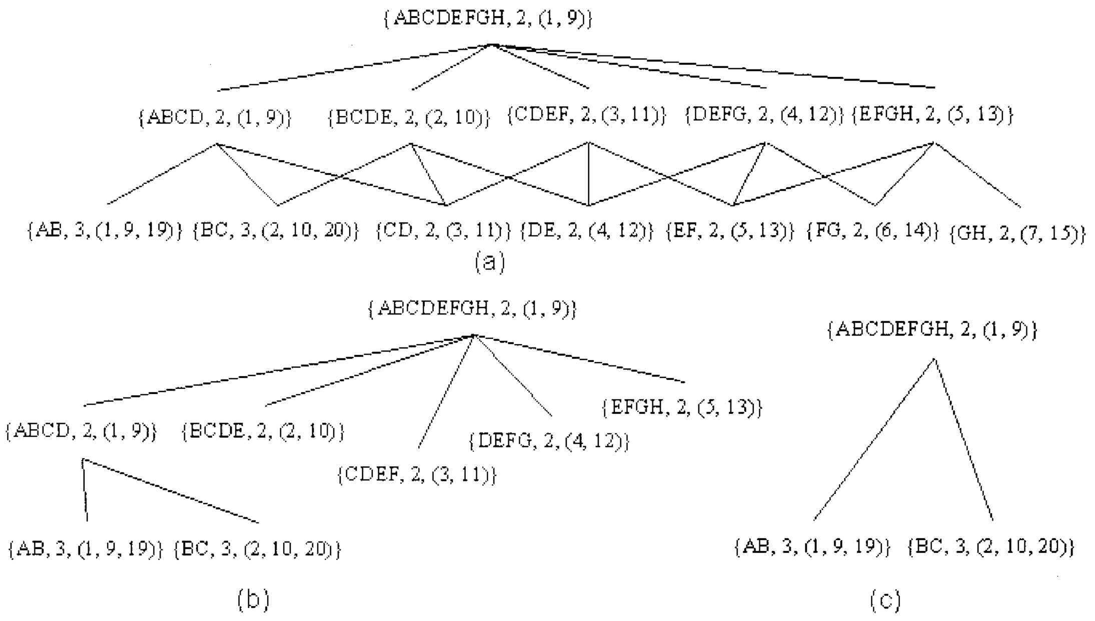
{ABCDEFGH, 2, (1, 9)}
{ABCD, 2, (1, 9)} {BCDE, 2, (2, 10)} {CDEF, 2, (3, 11)} {DEFG, 2, (4, 12)} {EFGH, 2, (5, 13)}
{AB, 3, (1, 9, 19)} {BC, 3, (2, 10, 20)} {CD, 2, (3, 11)} {DE, 2, (4, 12)} {EF, 2, (5, 13)} {FG, 2, (6, 14)} {GH, 2, (7, 15)}
(a)
{ABCDEFGH, 2, (1, 9)}
{ABCD, 2, (1, 9)} {BCDE, 2, (2, 10)} {CDEF, 2, (3, 11)} {DEFG, 2, (4, 12)} {EFGH, 2, (5, 13)}
{AB, 3, (1, 9, 19)} {BC, 3, (2, 10, 20)}
(b)
{ABCDEFGH, 2, (1, 9)}
{AB, 3, (1, 9, 19)} {BC, 3, (2, 10, 20)}
(c)

Fig. 4. (a) RP-tree after all trivial repeating patterns of length one are removed. (b) RP-tree after all trivial repeating patterns of length two are removed. (c) RP-tree after all trivial repeating patterns of length four are removed.

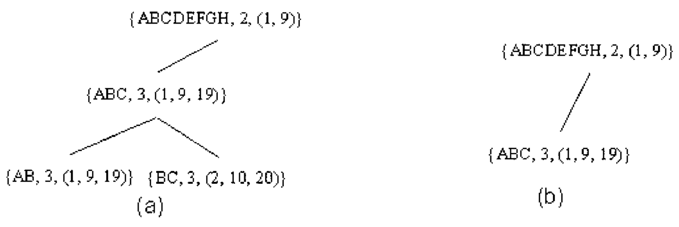

Fig. 5. (a) RP-tree after all repeating patterns of length three are generated. (b) RP-tree after all trivial repeating patterns are removed.

using an order-1 string-join, as shown in Fig. 5(a). Since $\mathit{freq}(\text{ABC}) > \mathit{freq}(\text{ABCDEFGH})$, ABC is nontrivial. However, since $\mathit{freq}(\text{ABC}) = \mathit{freq}(\text{AB}) = \mathit{freq}(\text{BC}) = 3$, the nodes corresponding to AB and BC are removed as shown in Fig. 5(b). Traversing this RP-tree, we find all the nontrivial repeating patterns of the music feature string $\mathbf{S}$, which are ABCDEFGH and ABC.

V. Performance Evaluation and Semantic Analysis

To show the efficiency of the proposed algorithms, a series of experiments are performed. We describe the environment of our testing platform and the characteristics of the testing data, and then illustrate the experiment results regarding performance issues. We also provide an analysis to show that the nontrivial repeating patterns possess good semantics of the music objects.

A. Experiment Set-Up

We implement the proposed algorithms for finding nontrivial repeating patterns by ANSI C and perform a series of experiments. 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. They are classical music objects of the 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

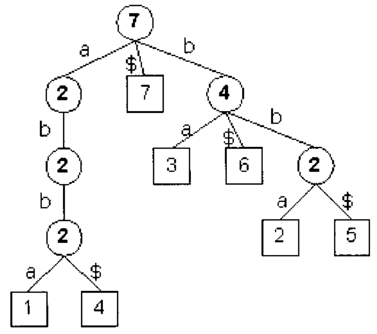

Fig. 6. Suffix tree T of the music feature string $\mathbf{S}$ = "abbabb."

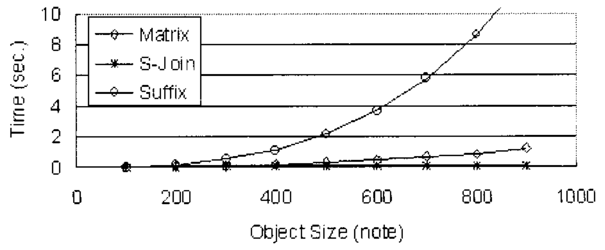

Fig. 7. Elapsed time versus object size of music objects for the synthetic data set.

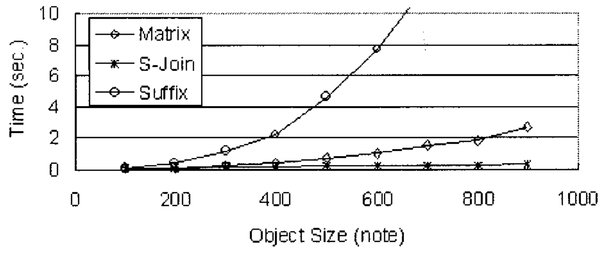

Fig. 8. Elapsed time versus object size of music objects for the real data set.

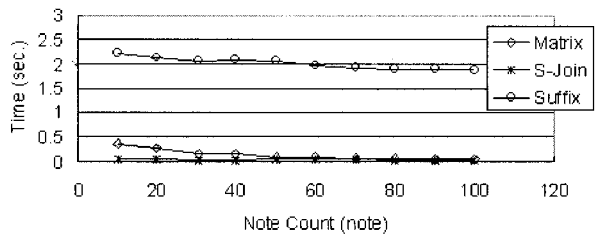

Fig. 9. Elapsed time versus note count of music objects for the synthetic data set.

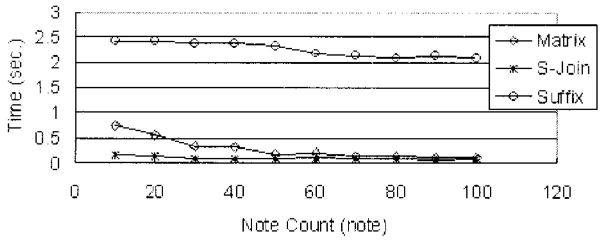

Fig. 10. Elapsed time versus note count of music objects for the real data set.

in a melody string. According to the MIDI Standard [27], 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 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 $\mathbf{S}$ is a $(|A|+1)$-ary rooted tree where $|A|$ is the alphabet size [28]. Each leaf (denoted by a box) of T corresponds to a position in $\mathbf{S}$, and each link is labeled with a symbol α, where $\alpha \in A \cup \{\$\}$ ('\$' is a special symbol denoting "end-of-string"). Each nonleaf node P (denoted by a circle) 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 nonleaf node P represents a substring of $\mathbf{S}$. 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 $\mathbf{S}$. For example, let $\mathbf{S}$ = "abbabb" be a music feature string. Its corresponding suffix tree T is shown in Fig. 6. The root node is associated with 7, which represents there are seven leaf nodes descending from the root. The path 7-2-2-2 represents the substring "abb." Since the number associated with its last node is two, "abb" is a repeating pattern whose frequency is two. By traversing the suffix tree of a music feature string $\mathbf{S}$, we can generate all repeating patterns in $\mathbf{S}$. In this example, the results are "a," "ab," "abb," "b," and "bb." Notice that the repeating patterns generated can be trivial.

B. Performance Analysis

Four factors which dominate the performance of the proposed algorithms are investigated: the object size of music objects, the note count of music objects, the length of the longest repeating patterns, and the number of nontrivial repeating patterns in a music feature string. The first two experiments illustrate the elapsed time versus the object sizes of music objects for the synthetic music data set and real music data set, as shown in Figs. 7 and 8, respectively. In both cases, the suffix tree approach is the worst since it enumerates all suffixes. 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. 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 approach for the real music data set. It 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.

Figs. 9 and 10 show the elapsed time versus the note count of the music objects for the synthetic and real music data sets, respectively. The elapsed time of both of our approaches decreases as the note count increases. This is because the length of repeating patterns tends to be shorter with larger note counts. When dealing with the cases having shorter repeating patterns, less iterations are needed when applying both of our approaches, which causes less elapsed time. However, for the suffix tree approach, the effect is limited.

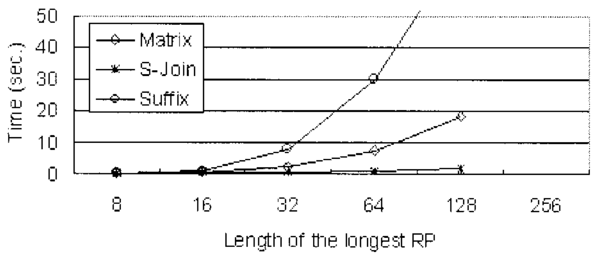

Fig. 11. Elapsed time versus length of the longest repeating patterns of the music objects for the synthetic data set.

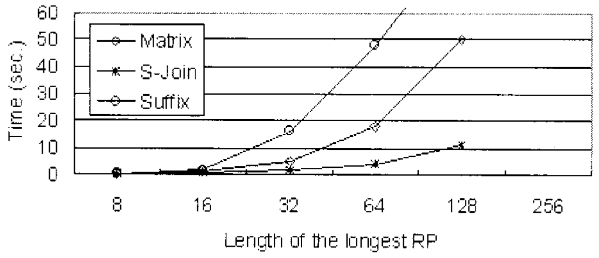

Fig. 12. Elapsed time versus length of the longest repeating patterns of the music objects for the real data set.

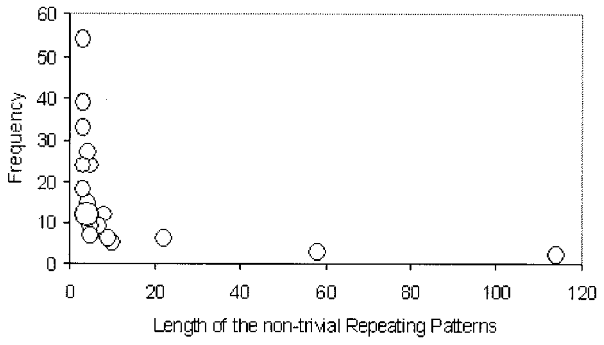

Fig. 13. Distribution of the lengths of all nontrivial repeating patterns in the song "Five Hundred Miles" (the bubble size indicates the number of nontrivial RPs with the same length and frequency).

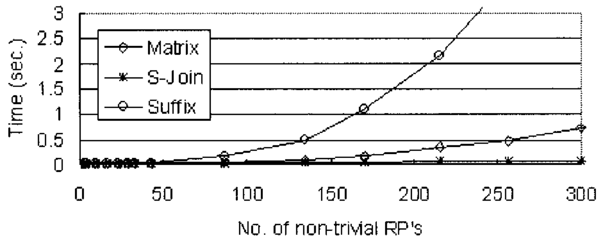

Fig. 14. Elapsed time versus number of nontrivial repeating patterns of the music objects for the synthetic music data set.

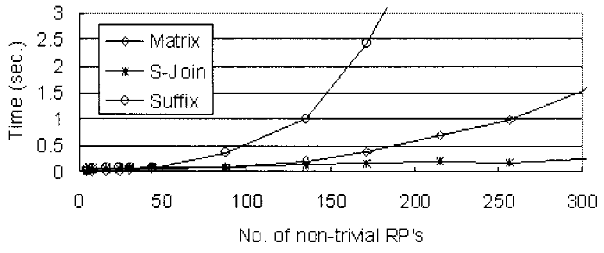

Fig. 15. Elapsed time versus number of nontrivial repeating patterns of the music objects for the real music data set.

Figs. 11 and 12 show the elapsed time versus the length of the longest repeating patterns of the music objects for the synthetic and real music data sets, respectively. The performance of the string-join approach is much better than that of the other approaches when the length of the longest repeating patterns is large.

The string-join approach will suffer when there are many nontrivial repeating patterns whose lengths are very close to the length of the longest repeating patterns. However, for real music objects this case rarely occurs since the length of the longest repeating patterns of a music object is typically much longer than that of the other nontrivial repeating patterns. For example, Fig. 13 shows the distribution of the lengths of all nontrivial repeating patterns in the song "Five Hundred Miles." The length of the longest repeating pattern is 112, while the length of the other nontrivial repeating patterns is less than or equal to 56 (i.e., smaller than $2^6 = 64$). When constructing the RP-tree, the repeating pattern of length 64 is trivial and removed from the RP-tree in the *Refine_RP_Tree* procedure. Therefore, we do not need to check the nontrivial repeating patterns whose lengths are between 112 and 64, and a large amount of execution time and storage space can be saved.

Figs. 14 and 15 show the elapsed time versus the number of nontrivial repeating patterns of the music objects for the synthetic and real music data sets, respectively. According to our experiments, the average number of all nontrivial 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 execution time easily becomes unsatisfied when the number of nontrivial repeating patterns increases.

C. Semantics Analysis

As we have mentioned before, for a real music object and a synthetic music object of the same object size and note count, the length of the longest repeating pattern of the real music object is much longer than that of the synthetic music object. This is because most of repeating patterns with longer lengths of a real music object are intentionally created by its composer. Therefore, we claim that the repeating patterns possess good semantics of the music objects. In this subsection, we show that better clustering of music data can be achieved by using repeating patterns as the feature of music objects. In the following experiment, sixty music objects are chosen from the real music data set in a random manner. These music data, shown in Table IV, are divided into three categories. Each category has 20 music objects composed by the same composer

For each music object, we apply our approaches on its melody string to find all nontrivial repeating patterns. Among them, we select the longest repeating pattern whose length is shorter than 30 as the *feature melody string* which is most likely to be the theme of the music object. By consulting [5], we observe that the length of most themes in classical music is shorter than 30.

Both of the original music melody strings and the corresponding feature melody strings are used to do music data clustering. For two melody strings, the editing distance, which

TABLE IV
STATISTICAL INFORMATION OF THE TESTING DATA

Object ID	Composer	Object Size	Feature melody string	
			Length	Feature melody string
1	Bach	102	24	C5-C5-C5-C5-A4-D5-C5-A#4-...
2	Bach	121	24	G3-F3-D#3-A2-A#2-C3-D3-D2-...
...				
21	Schubert	103	8	D5-C5-A#4-A4-G4-C6-Ab5-C7
22	Schubert	415	10	C3-G3-A#3-G3-A#3-A#3-G3-C#3- -...
...				
41	Schumann	302	22	D4-B4-G4-F#4-E4-C5-A4-G4-F#4-...
42	Schumann	187	16	E4-G4-G4-E4-G4-E4-F4-G4-F4-...
...				

TABLE V
WITHIN-CLUSTER AND BETWEEN-CLUSTER DISTANCE OF THE MUSIC OBJECTS IN TABLE IV

Data/Composer		W_k	W_k (avg.)	SB
Feature melody string	Bach	1.10	1.02	2.88
	Schubert	0.97		
	Schumann	1.00		
Original melody string	Bach	1.75	1.35	0.81
	Schubert	0.99		
	Schumann	1.32		

W_k: within-class distance
SB: sum of between-class distances

is defined based on their longest common substring, is used as the *dissimilarity measure*. Let $\mathbf{a}$ and $\mathbf{b}$ be two melody strings. The editing distance $d(\mathbf{a}, \mathbf{b})$, given by Wagner and Fischer [34], is defined as follows:

$$d(\mathbf{a}, \mathbf{b}) = \text{length}(\mathbf{a}) + \text{length}(\mathbf{b}) - 2 \times q(\mathbf{a}, \mathbf{b}) \tag{3}$$

where length($\mathbf{a}$), length($\mathbf{b}$), and $q(\mathbf{a}, \mathbf{b})$ denote the lengths of repeating patterns $\mathbf{a}$ and $\mathbf{b}$, and the longest common substring between $\mathbf{a}$ and $\mathbf{b}$, respectively.

Two 60×60 dissimilarity matrices for the original music melody strings and the corresponding feature melody strings, respectively, are then computed. By normalizing the dissimilarity matrices and applying the multi-dimensional scaling technique [8], [13] on the matrices, we map the music objects into points on a two-dimensional Euclidean space and measure the *validity* of the clustering accordingly.

The validity is measured by the *within-cluster distance* and *between-cluster distance* [17], [18]. The within-cluster distance W_k, for cluster k, is defined as [17]

$$W_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \left(\sum_{j=1}^{2} \left(x_{ij}^{(k)} - m_j^{(k)} \right)^2 \right)^{1/2} \tag{4}$$

where

n_k	number of music objects in cluster k;
$m_j^{(k)}$	cluster center which is defined as the arithmetic mean of the points for cluster k, along the jth dimension;
$x_{ij}^{(k)}$	value of the jth dimension for the ith music object belonging to cluster k.

Moreover, the sum of the between-cluster distances, SB, is defined as

$$SB = \sum_{1 \le a < b \le 3} \left(\sum_{j=1}^{2} \left(m_j^{(a)} - m_j^{(b)} \right)^2 \right)^{1/2}. \tag{5}$$

A compact and well-isolated clustering suggests a smaller within-cluster distance and a larger between-cluster distance [17], [18]. The results in Table V show that the clustering by the feature melody strings is better than the one by the original melody strings. This experiment supports our claim that the repeating pattern is a better music feature for clustering music objects.

VI. CONCLUSION

In this paper, we discuss various features of music objects for content-based data retrieval and define nontrivial repeating patterns of a music object. Two approaches are proposed for extracting all nontrivial repeating patterns in the feature string of a music object. In the correlative-matrix approach, we make use of the correlative matrix to generate all nontrivial repeating patterns. In the string-join approach, the longest repeating pattern can be found by repeatedly joining shorter repeating patterns using the string-join operation.

A series of experiments are conducted. The execution time of both approaches is dramatically less than the suffix approach. The semantic analysis shows that the discovered repeating patterns are a better music feature for both music data clustering and content-based retrieval.

Future research includes the following issues. Music data mining such as music data generalization and music association rules are currently being studied. We will analyze the semantics of various music features in both pop and classic music and try to derive high-level music features, such as *Doloroso* (sorrowfully), *Giocoso* (playfully), and *Vivo* (lively), to provide users with a more friendly interface to retrieve music data. Our approaches can be directly applied for monophonic music objects. Considering polyphonic music objects, we can first extract monophonic melody from polyphonic music objects, and then apply our approaches. We are also currently working on methods to directly extract repeating patterns from polyphonic music objects. Finally, since some modifications on a motive can be done by the composer to make the music more

TABLE VI
MUSIC TERMINOLOGIES USED IN THIS PAPER [9], [36]

Term	Definition
Beat	The pulse of music as directed by the combination of tempo and meter signatures. For example, an "allegro in 4/4" indicates four fast beats per measure.
Chord	The sounding of three or more notes at the same time, to identify the key, interval relationships, development and texture of music.
Doloroso	Sorrowfully.
Figure	The arrangement of a series of notes in a recognizable pattern.
Giocoso	Playfully.
Harmonic analysis	The study of chords and harmony in composition.
Key	In tonal music, the center or primary note that identifies the tonic. Thus, in the key of C, the note C is the tonic.
Measure	A recognizable grouping of beats, set off by bar lines.
Melody	The combination of notes that, when given rhythm, constitutes the vertical movement of music.
Monophonic	One sound, a melody without accompaniment.
Motive	A recognizable, combined rhythmic and melodic pattern, which runs through a movement or several movements of one composition.
Movement	The distinct sections of a larger work.
Musical form/Form	The structure and organization of music into controlled and predictable sound, pitch, harmony, texture and rhythm. Every composition follows one form or another, even beyond this basic definition.
Note	The name given to a single symbol on a musical score, indicating the pitch and duration of what is to be sung or played, and to the single sound actually heard.
Phrase	A musical statement, comparable to a sentence in a narrative. The four-and eight-measure phrases are traditional in Western music.
Polyphony	Music with two or more distinct lines of melody and harmony.
Refrain	(1) A section of vocal music that recurs following each stanza. (2) A section in an instrument composition that is repeated several times.
Rhythm	The value, duration and relationship of notes, and the timing of beats in each measure.
Score	The notation of all instruments, written out for each part, or summarized in a condensed version.
Sentence	Refer to "phrase".
Tempo	Time, the pace of a composition, indicated by words (allegro, adagio, andante, *etc.*), or by showing the exact number of beats per minute.
Theme	The subject of a composition, that is introduced and then developed.
Vivo	Lively.

fruitful and variant, a repeating pattern may be approximately repeated. We are currently also extending our algorithms such that approximate repeating patterns can be discovered.

APPENDIX A

(See Table VI).

APPENDIX B

```
Algorithm C2 Construct_Correlative_Matrix
(S, T)
/*  input: a melody string $S[1..n]$ */
/*  output: a correlative matrix $T[1..n, 1..n]$ */
Begin
1.  /* Fill the first row of the matrix T*/
2.  for $j = 2$ to n do
3.   if ($S[i] == S[j]$)then
4.       $T[1, j] = 1$
5.   else
6.       $T[1, j] = 0$
7.   endif
8.  /* Fill the remains of the matrix T */
9.  for $i = 2$ to $(n - 1)$ do
10.    for $j = (i + 1)$ to n do
11.      if ($S[i] == S[j]$) then
12.        $T[i, j] = T[i - 1, j - 1] + 1$
13.      else
14.        $T[i, j] = 0$
15.      endif
End
```

```
Algorithm C3 Generate_Repeating_Pattern (T, RP)
/*input: the correlative matrix $T[1..n, 1..n]$ */
/*output: a set of all repeating patterns and
their repeating frequencies $RP = \{(p_1, f_1\}),
(p_2, f_2), \ldots, (p_i, f_i)\}$ */
```

```
Begin
1.  /* Generate all repeating patterns */
2.  $CS = \emptyset$
3.  for each nonzero cell $T[i, j]$ of matrix $T$ do
4.  begin
5.    for $l = 1$ to $T[i, j]$ do
6.      begin
7.        $pat = S[(j - l + 1): j]$
8.        if $(T[i + 1, j + 1] == 0)$ then
9.            if $((pat, f_i, s_i) \in CS)$ then
10.             replace $(pat, f_i, s_i)$ by
    $(pat, f_i + 1, s_i)$
11.           else
12.             $CS = CS \cup \{(pat, 1, 0)\}$
13.           endif
14.       else
15.           if $((pat, f_i, s_i) \in CS)$ then
16.             replace $(pat, f_i, s_i)$ by
    $(pat, f_i + 1 s_i + 1)$
17.           else
18.             $CS = CS \cup \{(pat, 1, 1)\}$
19.           endif
20.       endif
21.     end
22.  end
23.  /* Calculate the repeating frequencies */
24.  for each $(p_i, f_i, s_i) \in CS$ do
25.  begin
26.    if $(f_i == s_i)$ then
27.      delete $(p_i, f_i, s_i)$
28.    else
29.      begin
30.        $f_i = (1 + \sqrt{1 + 8f_i})/2$
31.        insert $(p_i, f_i)$ into RP
32.      end
33.    endif
34.  end
End
```

Algorithm S2 Find_RP1(S)

```
/* input: the music feature string S =
  $\alpha_1\alpha_2\cdots\alpha_n$ */
/* output: all length one repeating patterns
  in S */
Begin
1.  RP[1] = $\emptyset$
2.  for $i = 1$ to $n$ do
3.    begin
4.      if $\alpha_i$ is not in RP[1]
5.        add $\{\alpha_i, 1, (i)\}$ to RP[1]
6.      else
      /* Assume $\{\alpha_i, k, (p_1, p_2, \ldots, p_k)\}$ is in RP[1]
  */
7.        replace $\{\alpha_i, k, (p_1, p_2, \ldots, p_k)\}$ by $\{\alpha_i$
    $, k + 1, (p_1, p_2, \ldots, p_k, i)\}$
8.      endif
9.    end
10.  return RP[1]
End
```

Algorithm S3 Find_RP2K(RP[2^{k-1}])

```
/* input: the set of all length $2^{k-1}$ repeating
patterns in S */
/* output: the set of all length $2^k$ repeating
  patterns in S */
Begin
1.  RP[$2^k$] = $\emptyset$
2.  for each pair of repeating patterns
  $\{\alpha_1\alpha_2\cdots\alpha_{2^{k-1}}$, freq$(\alpha_1\alpha_2\cdots\alpha_{2^{k-1}})$, $(p_1, p_2, \ldots, p_i)\}$
  and
  $\{\beta_1\beta_2\cdots\beta_{2^{k-1}}$, freq$(\beta_1\beta_2\cdots\beta_{2^{k-1}})$, $(q_1, q_2, \ldots, q_j)\}$
  in
  RP[$2^{k-1}$]
3.  begin
4.  $\{\gamma_1\gamma_2\cdots\gamma_{2^k}$, freq$(\gamma_1\gamma_2\cdot\gamma_{2^k})$, $(o_1, o_2, \ldots, o_h)\} =$
      $\{\alpha_1\alpha_2\cdots\alpha_{2^{k-1}}$, freq$(\alpha_1\alpha_2\cdots\alpha_{2^{k-1}})$, $(p_1, p_2, \ldots, p_i)\}\infty_0$
      $\{\beta_1\beta_2\cdots\beta_{2^{k-1}}$, freq$(\beta_1\beta_2\cdots\beta_{2^{k-1}})$, $(q_1, q_2, \ldots, q_j)\}$
5.  if freq$(\gamma_1\gamma_2\cdots\gamma_{2^k}) \geq 2$
6.    add $\{\gamma_1\gamma_2\cdots\gamma_{2^k}$, freq$(\gamma_1\gamma_2\cdots\gamma_{2^k})$, $(o_1,$
  $o_2, \ldots, o_h)\}$ to RP[$2^k$]
7.    endif
8.  end
9.  return RP[$2^k$]
End
```

Algorithm S4 Find_Longest_RP(RP[2^k])

```
/* input: the set of all length $2^k$ repeating
  patterns in S */
/* output: the set of the longest repeating
  patterns in S */
Begin
1.  RP[L] = $\emptyset$
2.  for each pair of repeating patterns
     $\{\alpha_1\alpha_2\cdots\alpha_{2^k}$, freq$(\alpha_1\alpha_2\cdots\alpha_{2^k})$, $(p_1, p_2, \ldots, p_i)\}$ and
  $\{\beta_1\beta_2\cdots\beta_{2^k}$, freq$(\beta_1\beta_2\cdots\beta_{2^k})$, $(q_1, q_2, \ldots, q_j)\}$ in
    RP[$2^k$] do
3.  begin
4.    $m = k/2$, $d = m/2$
5.    do
6.      $\{\gamma_1\gamma_2\cdots\gamma_{2^k+1-m}$, freq$(\gamma_1\gamma_2\cdots\gamma_{2^k+1-m})$, $(o_1, o_2, \ldots, o_h)\} =$
        $\{\alpha_1\alpha_2\cdots\alpha_{2^k}$, freq$(\alpha_1\alpha_2\cdots\alpha_{2^k})$, $(p_1, p_2, \ldots, p_i)\}\infty_m$
        $\{\beta_1\beta_2\cdots\beta_{2^k}$, freq$(\beta_1\beta_2\cdots\beta_{2^k})$, $(q_1, q_2, \ldots, q_j)\}$
7.      if freq$(\gamma_1\gamma_2\cdots\gamma_{2^k+1-m}) \geq 2$
8.        $m = m + d$
9.      else
10.        $m = m - d$
11.      endif
12.      $d = d/2$
13.    while $(d > 1)$
14.    add
  $\{\gamma_1\gamma_2\cdots\gamma_{2^k+1-m}$, freq$(\gamma_1\gamma_2\cdots\gamma_{2^k+1-m})$, $(o_1, o_2, \ldots,$
$o_h)\}$ to RP[L]
15.  end
16.  return RP[L]
End
```

```
Algorithm S5 Build_RP_Tree()
/* input: RP[1], RP[2], RP[4], ..., RP[$2^k$], and
  RP[L] which represent the sets of all length
  $2^0$, $2^1$, ..., $2^k$ repeating patterns and the set
  of the longest repeating patterns in S, re-
  spectively. */
/* output: the corresponding RP-tree */
Begin
1.  RP-tree $= \varnothing$
2.  add all repeating patterns in RP[1],
    RP[2], ...,  RP[$2^k$] and RP[L] as nodes in
    RP-tree
3.  for each pair of repeating patterns
      $\{\alpha_1\alpha_2\cdots\alpha_{2^k}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^k}), (p_1, p_2, \ldots, p_i)\}$ and
      $\{\beta_1\beta_2\cdots\beta_L, \mathrm{freq}(\beta_1\beta_2\cdots\beta_L), (q_1, q_2, \ldots, q_j)\}$ in
    RP[$2^k$] and RP[L]
4.    if $\alpha_1\alpha_2\cdots\alpha_{2^k}$ is a substring of $\beta_1\beta_2\cdots\beta_L$
5.      link $\{\alpha_1\alpha_2\cdots\alpha_{2^k}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^k}),$
        $(p_1, p_2, \ldots, p_i)\}$ and
        $\{\beta_1\beta_2\cdots\beta_L, \mathrm{freq}(\beta_1\beta_2\cdots\beta_L), (q_1, q_2, \ldots, q_j)\}$
6.    endif
7.  for $n = 1$ to $k - 1$ do
8.    for each pair of repeating patterns
        $\{\alpha_1\alpha_2\cdots\alpha_{2^n}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}), (p_1, p_2, \ldots, p_i)\}$ and
        $\{\beta_1\beta_2\cdots\beta_{2^{n+1}}, \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^{n+1}}), (q_1, q_2, \ldots, q_j)\}$
        in RP[$2^n$] and RP[$2^{n+1}$]
9.    if $\alpha_1\alpha_2\cdots\alpha_{2^k}$ is a substring of $\beta_1\beta_2\cdots\beta_L$
10.      link $\{\alpha_1\alpha_2\cdots\alpha_{2^n}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}),$
        $(p_1, p_2, \ldots, p_i)\}$ and
        $\{\beta_1\beta_2\cdots\beta_{2^{n+1}}, \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^{n+1}}),\ (q_1, q_2, \ldots, q_j)\}$
11.    endif
12.  return RP-tree
End
```

```
Algorithm S6 Refine_RP_Tree()
/* input: the RP-tree for a given music fea-
  ture string S */
/* output: the refined RP-tree in which
  trivial repeating patterns are removed */
Begin
1.  scan the RP-tree from the bottom level,
    each time select two linked nodes
    $\{\alpha_1\alpha_2\cdots\alpha_m, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_m), (p_1, p_2, \ldots, p_i)$ and
    $\{\beta_1\beta_2\cdots\beta_n, \mathrm{freq}(\beta_1\beta_2\cdots\beta_n), (q_1, q_2, \ldots, q_j)\}$,
    where the latter is a subnode of the former.
2.    if $(\mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_m) == \mathrm{freq}(\beta_1\beta_2\cdots\beta_n))$
3.      remove $\{\beta_1\beta_2\cdots\beta_n, \mathrm{freq}(\beta_1\beta_2\cdots\beta_n),$
        $(q_1, q_2, \ldots, q_j)\}$ from the RP-tree
4.    endif
End
```

```
Algorithm S7 Generate_Nontrivial_RP(RP)
/* input: the RP-tree for a given music fea-
  ture string S */
/* output: the all nontrivial repeating pat-
  terns in S */
Begin
1.  for $n = 1$ to $k - 1$ do
2.    for each pair of repeating patterns
      $\{\alpha_1\alpha_2\cdots\alpha_{2^n}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}), (p_1, p_2, \ldots, p_i)\}$ and
      $\{\beta_1\beta_2\cdots\beta_{2^n}, \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^n}), (q_1, q_2, \ldots, q_j)\}$
      which are subnodes of the repeating
      pattern do
     $\{\gamma_1\gamma_2\cdots\gamma_{2^{n+1}}, \mathrm{freq}(\gamma_1\gamma_2\cdots\gamma_{2^{n+1}}), (o_1, o_2, \ldots, o_h)\}$
3.      begin
4.        for $m = 1$ to $2^n-1$ do
5.          begin
6.            $\{\delta_1\delta_2\cdots\delta_{2^{n+1}-m}, \mathrm{freq}(\delta_1\delta_2\cdots\delta_{2^{n+1}-m}),$
            $(s_1, s_2, \ldots, s_g)\} =$
            $\{\alpha_1\alpha_2\cdots\alpha_{2^n}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}),$
            $(p_1, p_2, \ldots, p_i)\}\infty_m$
            $\{\beta_1\beta_2\cdots\beta_{2^n}, \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^n}),$
            $(q_1, q_2, \ldots, q_j)\}$
7.            if
    $(\mathrm{freq}(\delta_1\delta_2\cdots\delta_{2^{n+1}-m}) > \mathrm{freq}(\gamma_1\gamma_2\cdots\gamma_{2^{n+1}}))$
8.              add
    $\{\delta_1\delta_2\cdots\delta_{2^{n+1}-m}, \mathrm{freq}(\delta_1\delta_2\cdots\delta_{2^{n+1}-m}),$
              $(s_1, s_2, \ldots, s_g)\}$ to the refined
    RP-tree
9.            endif
10.           if $(\mathrm{freq}(\delta_1\delta_2\cdots\delta_{2^{n+1}-m}) ==$
            $\mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}))$
11.           remove $\{\alpha_1\alpha_2\cdots\alpha_{2^n}, \mathrm{freq}(\alpha_1\alpha_2\cdots\alpha_{2^n}),$
              $(p_1, p_2, \ldots, p_i)\}$
12.             endif
13.             if
    $(\mathrm{freq}(\delta_1\delta_2\cdots\delta_{2^{n+1}-m}) == \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^n}))$
14.           remove $\{\beta_1\beta_2\cdots\beta_{2^n}, \mathrm{freq}(\beta_1\beta_2\cdots\beta_{2^n}),$
              $(q_1, q_2, \ldots, q_j)\}$
15.             endif
16.           end
17.  traverse RP-tree and output all its nodes
End
```

REFERENCES

[1] D. A. Adjeroh and K. C. Nwosu, "Multimedia database management: Requirements and issues," *IEEE Multimedia*, vol. 4, no. 3, pp. 24–33, 1997.

[2] A. Apostolico, "The myriad virtues of subwords trees," in *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds. Berlin, Germany: Springer-Verlag, 1984, vol. 12, NATO ASI Series F: Computer and System Sciences, pp. 97–107.

[3] A. Apostolico and F. Preparata, "Data structures and algorithms for the string statistics problem," *Algorithmica*, vol. 15, pp. 481–494, 1996.

[4] V. Bakhmutova, V. D. Gusev, and T. N. Titkova, "The search for adaptations in song melodies," *Computer Music J.*, vol. 21, no. 1, pp. 58–67, Spring 1997.

[5] H. Barlow and S. Morgenstern, *A Dictionary of Musical Themes*. New York: Crown, 1975.

[6] P. B. Berra and A. Ghafoor, "Data and knowledge management in multimedia systems: The guest editors' introduction to the special section," *IEEE Trans. Knowledge Data Eng.*, vol. 10, no. 6, pp. 868–871, 1998.

[7] S. Blackburn and D. DeRoure, "A tool for content-based navigation of music," in *Proc. ACM Multimedia Conf.*, 1998, pp. 361–368.

[8] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. New York: Springer-Verlag, 1997.

[9] J. Burke, *The Illustrated Dictionary of Music*. London, U.K.: Warner, 1988.

[10] J. C. C. Chen and A. L. P. Chen, "Query by rhythm: An approach for song retrieval in music databases," in *Proc. IEEE IWRIDE*, 1998, pp. 139–146.

[11] M. T. Chen and J. Seiferas, "Efficient and elegant subword-tree construction," in *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds. Berlin, Germany: Springer-Verlag, 1984, vol. 12, NATO ASI Series F: Computer and System Sciences, pp. 97–107.
[12] T. C. Chou, A. L. P. Chen, and C. C. Liu, "Music databases: Indexing techniques and implementation," in *Proc. IEEE IWMDBMS*, 1996, pp. 46–53.
[13] T. F. Cox and M. A. A. Cox, *Multidimensional Scaling*. London, U.K.: Chapman and Hall, 1994.
[14] J. Foote, "An overview of audio information retrieval," in *Multimedia Systems*. New York: ACM Press/Springer-Verlag, 1999, vol. 7, pp. 2–10.
[15] A. Ghias, H. Logan, D. Chamberlin, and B. C. Smith, "Query by humming: Musical information retrieval in an audio database," in *Proc. ACM Multimedia Conf.*, 1995, pp. 231–236.
[16] J. L. Hsu, C. C. Liu, and A. L. P. Chen, "Efficient repeating pattern finding in music databases," in *Proc. ACM ICIKM*, 1998, pp. 281–288.
[17] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[18] M. James, *Classification Algorithms*. London, U.K.: William Collins, 1985.
[19] G. T. Jones, *Music Theory*. New York: Harper & Row, 1974.
[20] C. L. Krumhansl, *Cognitive Foundations of Musical Pitch*. London, U.K.: Oxford Univ. Press, 1990.
[21] C. C. Liu, J. L. Hsu, and A. L. P. Chen, "An approximate string matching algorithm for content-based music data retrieval," in *Proc. ICMCS*, 1999, pp. 451–456.
[22] ——, "Efficient theme and nontrivial repeating pattern discovering in music databases," in *Proc. ICDE*, 1999, pp. 14–21.
[23] ——, "Efficient near neighbor searching using multi-indexes for content-based multimedia data retrieval," *Multimedia Tools Applicat.*, vol. 13, no. 3, pp. 235–254, 2001.
[24] F. Lerdahl and R. Jackendoff, *A Generative Theory of Tonal Music*. Cambridge, MA: MIT Press, 1983.
[25] M. Lohr and T. C. Rakow, "Audio support for an object-oriented database-management system," *ACM Multimedia Syst. J.*, pp. 286–297, 1995.
[26] R. W. Lundin, *An Objective Psychology of Music*. Melbourne, FL: Krieger, 1985.
[27] MIDI Manufacturers Association (MMA). MIDI 1.0 specification. [Online]. Available: http://www.midi.org/
[28] E. M. McCreight, "A space economical suffix tree construction algorithm," *J. Assoc. Comput. Mach.*, vol. 23, pp. 262–272, 1976.
[29] E. Narmour, *The Analysis and Cognition of Basic Melodic Structures*. Chicago, IL: Univ. of Chicago Press, 1990.
[30] K. C. Nwosu, B. Thuraisingham, and P. B. Berra, "Multimedia database systems: A new frontier," *IEEE Multimedia*, vol. 4, no. 3, pp. 21–23, 1997.
[31] M. T. Ozsu and S. Christodoulakis, "Introduction to the special issue on multimedia databases," *VLDB Journal*, vol. 7, no. 4, p. 205, 1998.
[32] S. Pfeiffer, S. Fischer, and W. Effelsberg, "Automatic audio content analysis," in *Proc. ACM Multimedia Conf.*, 1996, pp. 21–30.
[33] R. E. Prather, "Harmonic analysis from the computer representation of a musical score," *Commun. ACM*, vol. 39, no. 12, p. 119, Dec. 1996.
[34] D. Sankoff and J. B. Kruskal, Eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA: Addison-Wesley, 1983.
[35] J. A. Sundberg, A. Friberg, and L. Fryden, "Common secrets of musicians and listeners: An analysis-by-synthesis study of musical performance," *Representing Musical Structure*, 1991.
[36] M. C. Thomsett, *Musical Terms, Symbols, and Theory: An Illustrated Dictionary*. Chicago, IL: St. James, 1985.
[37] A. Uitdenbogerd and J. Zobel, "Manipulation of music for melody matching," in *Proc. ACM Multimedia Conf.*, 1998, pp. 235–240.
[38] E. Ukkonen, "On-line construction of suffix tree," *Algorithmica*, vol. 14, pp. 249–260, 1995.
[39] P. Weiner, "Linear pattern matching algorithms," in *Proc. IEEE Annu. Symp. Switching and Automata Theory*, 1973, pp. 1–11.
[40] E. Wold, T. Blum, D. Keislar, and J. Wheaton, "Content-based classification, search, and retrieval of audio," *IEEE Multimedia*, vol. 3, no. 3, pp. 27–36, Fall 1996.
[41] A. Yoshitaka and T. Ichikawa, "A survey on content-based retrieval for multimedia databases," *IEEE Trans. Knowledge Data Eng.*, vol. 11, no. 1, pp. 81–93, 1999.
[42] R. E. Prather, "Harmonic analysis from the computer representation of a musical score," in *CACM, Virtual Extension Edition*, pp. 239–255.

Jia-Lien Hsu received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C, in 1994, where he is currently pursuing the Ph.D. degree in computer science. His research interests include multimedia databases, music information retrieval, and content-based retrieval.

Chih-Chin Liu (M'99) received the B.S. degree in electrical engineering and the Ph.D. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C, in 1990 and 1998, respectively.

He is currently an Assistant Professor in the Department of Computer Science and Information Engineering at Chung Hua University. His research interests include multimedia databases, multimedia query processing, and multimedia data mining.

Arbee L. P. Chen (S'80–M'84) received the B.S. degree in computer science from National Chiao-Tung University, Taiwan, R.O.C., in 1977, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1984.

He joined National Tsing Hua University, Taiwan, as a National Science Council (NSC) sponsored Visiting Specialist in August 1990, and became a Professor in the Department of Computer Science in 1991. He was a Member of Technical Staff at Bell Communications Research from 1987 to 1990, an Adjunct Associate Professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, New York, and a Research Scientist at Unisys from 1985 to 1986. His current research interests include multimedia databases, data mining, and mobile computing. He is an editor of *World Wide Web Journal* (Kluwer).

Dr. Chen has organized (and served as a Program Co-Chair for) the 1995 IEEE Data Engineering Conference and the 1999 International Conference on Database Systems for Advanced Applications (DASFAA) in Taiwan. He is a recipient of the NSC Distinguished Research Award.