Efficient Feature Mining in Music Objects

Jia-Ling Koh and William D.C. Yu

Department of Information and Computer Education
National Taiwan Normal University
Taipei, Taiwan 106, R.O.C.
Email: jlkoh@ice.ntnu.edu.tw

Abstract. This paper proposes novel strategies for efficiently extracting repeating patterns and frequent note sequences in music objects. Based on bit stream representation, the bit index sequences are designed for representing the whole note sequence of a music object with little space requirement. Besides, the proposed algorithm counts the repeating frequency of a pattern efficiently to rapidly extracting repeating patterns in a music object. Moreover, with the assist of appearing bit sequences, another algorithm is proposed for verifying the frequent note sequences in a set of music objects efficiently. Experimental results demonstrate that the performance of the proposed approach is more efficient than the related works.

1. Introduction

Data mining has received increasing attention in the area of database, with the conventional alphanumeric data having been extensively studied [4]. However, the mining of multimedia data has received lesser attention. Some interesting patterns or rules in multimedia data can be mined to reveal the hidden useful information. In the melody of a music object, many sequences of notes, called repeating patterns, may appear more than once in the object. For example, “sol-sol-sol-mi” is a well-known melody that repeatedly appears in Beethoven's Fifth Symphony. The repeating pattern, an efficient representation for content-based music retrieval, can represent the important characteristics of a music object. Moreover, the sequence of notes that frequently appear in a set of music objects, an interesting pattern called a frequent note sequence, can be used for music data analysis and classification.

Hsu [5] proposed an effective means of finding repeating patterns of music objects based on a data structure called *correlative matrix*. The correlative matrix was used to record the lengths and appearing positions of note patterns that are the intermediate results during the extracting process. However, as the lengths of music objects increase, the memory requirement increases rapidly. The authors of [5] proposed a new approach in 1999 for discovering repeating patterns of music objects [8]. In this approach, the longer repeating pattern was discovered by using string-join operations to repeatedly combine shorter repeating patterns. Therefore, the storage space and execution time is reduced. However, the approach is inefficient when there exist many repeating patterns whose length are extremely close to the length of the longest repeating patterns.

Mining frequent note sequences that differ from mining frequent item sets must consider the order of data items. Agrawal [2] extended the Apriori approach [1] to mine the sequential orders that frequently appear among the item sets in customer transactions. Bettini [3] adopted the finite automata approach to extract the frequent sequential patterns. However, the lengths and the ending of the repeating patterns and frequent note sequences in music objects can not be predicted, such that the terminating state of the finite automata can not be defined. Wang [9] designed suffix tree structure for representing the data set. Although capable of confirming the appearance of a certain data sequence efficiently by tracing the path of the suffix tree, that approach requires much storage space.

In light of above developments, this paper presents an efficient approach for extracting all maximum repeating patterns in a music object. Based on bit stream representation, the *bit index sequences* are designed for representing the whole note sequence of a music object with little space requirement. In addition, the repeating frequency of a candidate pattern can be counted by performing bit operations on bit index sequences so that repeating patterns can be verified efficiently. Moreover, an extended approach for mining frequent note sequences in a set of music objects is provided. Also based on bit stream representation, the *appearing bit sequences* are designed for representing the music objects in which a note sequence appears. With the aid of appearing bit sequences and bit index sequences, the number of music objects in which a note sequence appears can be counted by performing bit operations. Therefore, the frequent note sequences can be extracted efficiently.

The rest of this paper is organized as follows. Section 2 introduces the basic definitions of the problem domain. Section 3 describes the proposed data structure and algorithm for extracting repeating patterns. By extending the idea proposed in Section 3, Section 4 introduces the algorithm for extracting frequent note sequences in a set of music objects. Next, Section 5 summarizes the experimental results that demonstrate the efficiency of the proposed algorithm. Finally, we conclude with a summary and directions for future work in Section 6.

2. Repeating Patterns and Frequent Note Sequences

A repeating pattern is a consecutive sequence of notes appearing more than once in a music object and a frequent note sequence is the one appearing among a set of music objects frequently. As important features in music objects, these two kinds of patterns can be used for content-based retrieval and analysis of music data.

[*Def. 2.1*] Let X denote a note sequence consisting of $P_1P_2...P_n$, where n denotes a positive integer and each P_i (i=1, ..., n) denotes a note. The length of X is denoted by length(X), whose value is n.

[*Def. 2.2*] Let X denote a note sequence consisting of $P_1P_2...P_n$ and X' denote another note sequence consisting of $Q_1Q_2...Q_m$, where m and n are positive integers and each P_i (i=1, ..., n) and Q_j (j=1, ..., m) denotes a note, respectively. X' is called a *sub-pattern* of X if

(1) $m \le n$,

(2) and $\exists$ positive integer i, $i \le n$, such that $P_iP_{i+1}...P_{i+m-1} = Q_1Q_2...Q_m$.

[*Def. 2.3*] Let X denote a note sequence consisting of $P_1P_2...P_n$, where *n* denotes a positive integer and each P_i (i=1, ..., *n*) denotes a note. The set containing all the sub-patterns of X is {X'| X'= $P_i...P_j$, where i, j ∈ positive integers, i ≥ 1, j ≤ n, and i ≤ j }, which is denoted by SUBP(X).

[*Def. 2.4*] A note sequence X is a *repeating pattern* in music object M, if X satisfies

(1) X ∈ SUBP(the entire note sequence of M), and

(2) $Freq_M(X) \geq$ *min-freq*.

$Freq_M(X)$ denotes the repeating frequency of X in M. Besides, *min-freq* denotes a constant value, which can be specified by users. Hereinafter, *min-freq* is set to be 2.

[*Def. 2.5*] The note sequence X is a *maximum repeating pattern* in M if X is a repeating pattern in M, and there does not exist another repeating pattern X' in M such that X is a sub-pattern of X' and $Freq_M(X)$ is equal to $Freq_M(X')$.

[Example 2.1]

Table 1. Music example for example 2.1

Music ID	Music Melody
M1	DoMiMiMiSoSo
M2	SoMiMiFaReRe
M3	SoSoSoMiFaFaFaRe
M4	SoDoDoReSoDoDo

Consider the music objects as shown in Table 1. In music object M1, $Freq_{M1}$("Mi") = 3, $Freq_{M1}$("So") = 2, and $Freq_{M1}$("MiMi") = 2. Because the repeating frequencies of these three patterns are larger than or equal to *min-freq*, "So", "Mi", and "MiMi" are repeating patterns in M1. In addition, these three patterns are also maximum repeating patterns in M1. In music object M4, $Freq_{M4}$ ("So") = 2, $Freq_{M4}$ ("Do") = 4, $Freq_{M4}$ ("SoDo") = 2, $Freq_{M4}$ ("DoDo") = 2, and $Freq_{M4}$ ("SoDoDo") = 2. These five patterns are repeating patterns in music object M4. However, only "Do" and "SoDoDo" are maximum repeating patterns.

[*Def. 2.6*] Let MS denote a set of music objects. The note sequence Y is a *frequent note sequence* in MS, if $Sup_{MS}(Y) \geq$ *min-sup*,

where $Sup_{MS}(Y) = \sum_{M \in MS} Contains_M(Y)$,

$Contains_M(Y) = 1$, if $Freq_M(Y) \geq 1$

$Contains_M(Y) = 0$, otherwise.

In the definition, *min-sup* denotes a constant value, which is used to require that a frequent note sequence must appear in at least *min-sup* music objects in MS. Besides, the notation $Sup_{MS}(Y)$ is named the *support* of Y in MS.

[*Def. 2.7*] The note sequence Y is a *maximum frequent note sequence* in MS if Y is a frequent note sequence in MS, and there does not exist another frequent note sequence Y' in MS such that Y is a sub-pattern of Y'.

[*Def. 2.8*] Let P and Q denote a note in music object M, respectively. Q is named an adjacent note of P if note sequence PQ appears in M. We also say Q is *adjacent to* P. Moreover, PQ is an *adjacent note pair* in M.

3. Repeating Patterns Mining

3.1 Bit Index Table

The bit index table is designed based on bit stream representation. A *bit index sequence* is constructed for each note in the note sequence of a music object. The length of the bit index sequence equals the length of the note sequence. Assume that the least significant bit is numbered as bit 1 and the numbering increases to the most significant bit. If the *i*th note in the note sequence is note N, bit *i* in the bit index sequence of N is set to be 1; otherwise, the bit is set to be 0. That is, the bits in the bit index sequence of note N represent the appearing locations of N in the music object. Consider the note sequence "DoDoDoMiMi" as an example. The bit index sequence of "Do", denoted by BIS_{Do}, is 00111. Then, the entire note sequence of a music object is represented by a bit index table that consists of bit index sequences for various notes in the music object. Consider the music object with note sequence "SoSoSoMFaFaFa Re". The length of the sequence is 8, which contains four various notes. Table 2 presents the corresponding bit index table.

Table 2. Example of bit index table

Note	Bit index sequence
Re	10000000
Mi	00001000
Fa	01110000
So	00000111

For each note P, the number of bits with value 1 in its bit index sequence implies the repeating frequency $Freq_M(P)$. Suppose a note sequence consists of two notes P and Q. The repeating frequency of the note sequence PQ also can be counted efficiently by performing bit operations on bit index sequences as described in the following. Initially, the bit index table provides the corresponding bit index sequences of note P and Q, BIS_P and BIS_Q. The left shift operation on BIS_P is performed. An **and** operation on the previous result and BIS_Q is then performed to get the bit index sequence of PQ. The bits with value 1 in the sequence correspond to the positions in music object M where PQ appears. In addition, the number of bits with value 1 represents $Freq_M(PQ)$. Similarly, this strategy can be applied to verify whether a note sequence ST, consisting of a note sequence S with length(S) ≥ 1 and a note T, is a repeating pattern. Hereinafter, Frequent_Count(M, P) represents the function for counting the repeating frequency of note sequence P in music object M.

[Example 3.1]

Given the bit index table of music object M as shown in Table 2, where the represented note sequence is "SoSoSoMiFaFaFaRe". The process for evaluating whether the note sequences "SoSo" and "SoSoMi" are repeating patterns is as follows.

<1> Frequent_Count(M, SoSo)

1) Obtain the bit index sequence of the second note "So", BIS_{So} = 00000111.
2) Perform the left shift operation on BIS_{So}, which is 00000111. The resultant sequence is then assigned to temporal variable t, t =shl(BIS_{So}, 1) =00001110.

3) Perform $r = t \wedge BIS_{So}$, and the resultant bit sequence is 00000110.
4) Count the number of 1s in the bit sequence r and get $Freq_M(SoSo) = 2$. This implies that "SoSo" appears in M two times and, thus, is a repeating pattern in M.

<2> Frequent_Count(M, SoSoMi)

1) Obtain $BIS_{Mi} = 00001000$.
2) BIS_{SoSo} , 00000110, is obtained from the previous step. Perform the left shift operation and assign it to variable t. t = 00001100.
3) Perform $r = t \wedge BIS_{Mi}$, and, in doing so, bit sequence 00001000 is obtained.
4) The number of 1s in the bit sequence r is 1. This indicates that $Freq_M(SoSoMi) = 1$ and, thus, "SoSoMi" is not a repeating pattern in M.

3.2 Algorithm for Mining Repeating Patterns

This subsection presents an algorithm, named MRP algorithm, for mining repeating patterns by applying the bit index table. Redundancy may occur among the repeating patterns and, therefore, only the maximum repeating patterns are extracted. The algorithm consists of two phases: mining repeating patterns and extracting maximum repeating patterns.

The mining phase applies the depth first search approach. First, the candidate pattern consists of a single note. If the pattern is a repeating pattern, the algorithm constructs a new candidate pattern by adding an adjacent note to the old one and verifies the repeating frequency. If the new candidate is a repeating pattern, the process will continue recursively. Otherwise, the process returns to the old candidate pattern before adding a note and attempts to add another note to the sequence.

Consider the music object with note sequence "ABCABC". The candidate pattern with length 1, "A", is initially chosen and its repeating frequency is then counted. Since the repeating frequency of "A" equals 2, "A" is a repeating pattern. Next, an adjacent note of "A" is added, and the candidate pattern "AB" is constructed as well. After verifying that "AB" satisfies the definition of a repeating pattern, the candidate "ABC" is then constructed. Although "ABC" is verified to be a repeating pattern, a repeating pattern with length 4 can not be obtained by adding any note to "ABC". Thus, the recursive process is terminated and the status returns to the sequence "AB" and then back to the sequence "A". When the process returns the status back to the sequence "A", no other adjacent notes of "A" can be added. Next, another candidate pattern with length 1, "B", is chosen and the above process repeats. After that, the repeating patterns "B" and "BC" are found. In addition, the final candidate pattern with length 1, "C", is also a repeating pattern.

Among the extracted repeating patterns, not all repeating patterns are maximum repeating patterns. Therefore, the second phase is required to investigate the sub-pattern relationship and repeating frequencies for every pair of repeating patterns to remove the non-maximum repeating patterns. In order to reduce processing cost of the second phase, the following property is applied in the mining phase for removing a part of non-maximum repeating patterns.

If repeating pattern T is constructed from repeating pattern S by adding a note, S is a sub-pattern of T. Therefore, we can make sure that S is not a maximum repeating pattern if S and T have the same repeating frequency and S can be removed in the mining phase. In the example illustrated above, the repeating frequency of sequence "AB" is 2, and so is sequence "ABC". Therefore, "AB" is not a maximum repeating pattern. Similarly, "A" is not a maximum repeating pattern. Therefore, only sequences "ABC","BC", and "C" remain in the result because they are possibly maximum repeating pattern and require the processing of second phase. This strategy can filter out many non-maximum repeating patterns, such that it is more efficient when extracting the maximum repeating patterns during the second phase.

Among the possible maximum repeating patterns remained in set RP, no two repeating patterns exist that have the same repeating frequency and one is the prefix of the other. These patterns are stored in a table as shown in Table 3. In addition to the note sequence of the repeating pattern, the other three columns are used to store the length, repeating frequency, and the final starting position where the pattern appears in the music object. Applying the table allows us to extract the maximum repeating patterns without performing string matching.

The function MMRP() is used to extract the maximum repeating patterns. The pseudo codes for function MMRP() are shown below. The first two predicates in the "if clause" are used to verify whether if note sequence T is a sub-pattern of sequence S. If T is a sub-pattern of S and has the same repeating frequency with S, T is not a maximum repeating pattern and is removed from the results.

```
Function MMRP()
{for each note sequence S in RP
     for each note sequence T in RP and T ≠ S
       i := the final starting position where S appears
       j := the final starting position where T appears
       if((i < j)and(length(S)+i≥length(T)+j)and(Freq_M(S)=Freq_M(T))
         RP = RP - {T}
     end for }
```

Table 3. Example of maximum repeating patterns

Repeating pattern	Length	Repeating frequency	Final starting position
ABC	**3**	**2**	**4**
BC	**2**	**2**	**5**
C	**1**	**2**	**6**

Consider the note sequence "ABCABC". Table 3 displays the possible maximum repeating patterns extracted in the mining phase. Both the final starting positions of "BC" and "C" are larger than the one of "ABC". These patterns have the same result by adding the final starting positions and the lengths of the patterns. Moreover, their repeating frequencies are all the same. Therefore, only "ABC" is a maximum repeating pattern among these patterns.

4. Frequent Note Sequences Mining

This section describes a novel algorithm for mining frequent note sequences in a set of music objects. The note combination structure is designed for storing the adjacent note pairs appearing in a set of music objects. Applying the structure allows us to produce the candidate frequent note sequences efficiently in the mining process.

4.1 Note Combination Structure

The note combination structure is a two-dimensional array named *NC table*. Notably, the index values of the array correspond to the distinct notes. The data stored in array NC[M][N] is a bit stream representing the music objects in which the note sequence MN appears and is named the *appearing bit sequence* of MN. The length of the sequence equals the number of music objects. If MN appears in the *i*th music object, bit *i* in the appearing bit sequence is set to be 1; otherwise, the bit is set to be 0. In the following, $Appear_S$ denotes the appearing bit sequence of a note sequence S.

Table 4. Example of NC table

	Do	Re	Mi	Fa
Do	000	011	100	000
Re	000	000	011	000
Mi	000	000	000	111
Fa	110	000	000	000

[Example 4.1] Consider the note sequences of three music objects: "DoReMiFa", "ReMiFaDoRe", and "MiFaDoMi". All the adjacent note pairs appearing in the first music object are "DoRe", "ReMi", and "MiFa". Because "DoRe" appears in music object 1, the first bit of the appearing bit sequence in NC[Do][Re] is set to be 1. Table 4 presents the NC table for these three music objects.

The bit sequence stored in NC[Do][Re], which is 011, represents that the note sequence "DoRe" appears in music object 1 and 2. Similarly, $Appear_{MiFa}$ is stored in NC[Mi][Fa]. The sequence "111" implies that "MiFa" appears in all the music objects.

For a single note P, the appearing bit sequence of P, denoted as $Appear_P$, represents the music objects in which note P appears. This sequence can be obtained via performing **or** operations on the sequences stored in the row and column indexed by P. Then the number of bits with value 1 in $Appear_P$ represents the support of P.

For a note sequence consisting of two notes, P and Q, $Appear_{PQ}$ is obtained from the NC table directly. Therefore, the support of PQ also can be counted efficiently according to the number of 1s in $Appear_{PQ}$.

Suppose a note sequence S with length n ($n \geq 2$) is a frequent note sequence. And a new candidate note sequence is constructed by adding a note Q to S. In order to reduce the processing cost of frequent note sequence verification, information in the NC table is used to filter out the non-frequent note sequences as early as possible. Suppose the last note in S is P. The note sequence SQ remains as a candidate if both the following two requirements are satisfied.

1. The number of bits with value 1 in $Appear_{PQ}$, stored in NC[P][Q], is larger than or equal to *min_sup*.
2. Perform $Appear_S \wedge Appear_{PQ}$. The number of 1s in the resultant sequence is larger than or equal to *min_sup*.

The result of $Appear_S \wedge Appear_{PQ}$ is an approximation of $Appear_{SQ}$, which represents the music objects in which both S and RQ appear. If the number of bits with value 1 in this sequence is larger than or equal to *min-sup*, both S and RQ are frequent note sequences. However, further verification is required by invoking function Frequent_Count() to verify if SQ is actually a note sequence in the corresponding music objects specified by the bits with value 1.

4.2 Algorithm for Mining Frequent Note Sequences

This subsection presents MFNS algorithm designed for mining frequent note sequences, which mainly consists of the following three steps.

[Step 1] Bit index tables and NC table construction.

The note sequences of the given music objects are scanned sequentially. The bit index table for each music object and the NC table for the given music objects are then constructed.

[Step 2] Frequent note sequences extraction.

Similar to the mining phase in MRP algorithm, this step also applies the depth first search approach to extract the frequent note sequences. Initially, the candidate note sequence consists of a single note. The information in the NC table and bit index tables is used to verify if a candidate is a frequent note sequence and construct a new candidate sequence recursively.

[Step 3] Maximum frequent note sequences extraction.

This step extracts maximum frequent note sequences. For any two frequent note sequences P and P', if P is the sub-pattern of P', P is not a maximum frequent note sequence and removed from the result.

[Example 4.2] Table 4 displays the NC table of a set of music objects, and *min-sup* is set to be 2. Frequent note sequence mining according to step 2 of the algorithm is performed as follows.

<1> Select note "Do" as the first note of candidate note sequences.

After performing $Appear_{DoRe} \vee Appear_{DoMi} \vee \ldots \vee Appear_{FaDo}$, the outcome is "111". The sequence represents that "Do" appears in all the music objects and is a frequent note sequence. Then note "Re" is chosen to added adjacent to "Do". $Appear_{DoRe}$="011" implies that "DoRe" appears in two music objects and is a frequent note sequence. Next, note "Mi" is chosen to construct candidate sequence "DoReMi". The outcome is "011" after performing $Appear_{DoRe}$ ("011") $\wedge$ $Appear_{ReMi}$ ("011"). This occurrence implies that "DoRe" and "ReMi" appear in the first and second music objects. It is necessary to verify whether if "DoReMi" exists in the first and second music objects.

Closely examining the bit index sequences of "DoReMi" in the first and second music objects reveals that "DoReMi" only appears in the first music. Therefore, "DoReMi" is not a frequent note sequence. Next, no other notes can be added to

the frequent note sequences "DoRe" and "Do", individually, to construct new candidates. Therefore, the mining process for the frequent note sequences begins with "Do" terminates. "Do" and "DoRe" are the extracted frequent note sequences.

<2> Select note "Re" as the first note of candidate note sequences.

Applying the same mining process allows us to extract the frequent note sequences "Re", "ReMi", and "ReMiFa" sequentially. Next, the note "Do" is chosen to construct a candidate pattern "ReMiFaDo". However, after performing the **and** operation on $Appear_{ReMiFa}$("011") and $Appear_{FaDo}$("110"), there is only one bit with value 1 in the resultant sequence, implying that only the second music object contains "ReMiFa" and "FaDo" at the same time. Therefore, "ReMiFaDo" is obviously not a frequent note sequence. The frequent note sequences beginning with "Re" are "Re", "ReMi", and "ReMiFa".

Similarly, notes "Mi" and "Fa" are chosen as the first note of candidate note sequences individually, and the above process repeats. Finally, the extracted frequent note sequences beginning with "Mi" are "Mi", "MiFa", and "MiFaDo". The frequent note sequences beginning with "Fa" are "Fa" and "FaDo".

For a frequent note sequence K, K is not a maximum frequent note sequence if K is a prefix of another frequent note sequence. Such sequences are not stored in MaxFNS when mining frequent note sequences. Therefore, step 3 can be performed more efficiently via only considering the frequent note sequences in MaxFNS. Among the extracted frequent note sequences, only "DoRe", "ReMiFa", "MiFaDo" and "FaDo" are stored in MaxFNS, which are possible maximum frequent note sequences. After step 3 is performed, only the maximum frequent note sequences "DoRe", "ReMiFa", and "MiFaDo" remain.

5. Performance Study

The efficiency of the proposed MRP algorithm is evaluated by comparing it with two related approaches: the correlative matrix approach [5] and the string join method [8].

The algorithms are implemented and performed on a personal computer. The data sets used in the experiments include synthetic music objects and real music objects. Herein, the *object size* of a music object is defined as the length of the note sequence and the *note count* represents the number of various notes appearing in the music object. In addition, the total repeating frequency of repeating patterns in a music object is the *RP repeating frequency* and the number of maximum repeating patterns is named *RP count.* Due to page limit, the parameter setting of synthetic music objects for each experiment refers to [7].

According to the results of the experiments shown in Fig.1, the execution time of MRP algorithm is less than 0.2 seconds for the real music objects and less than 0.01 seconds for the synthetic music objects. The proposed algorithm is more efficient than the other two approaches. Moreover, the memory requirement of MRP algorithm is less than the other ones, thus confirming that the algorithm proposed herein is highly appropriate for mining maximum repeating patterns.

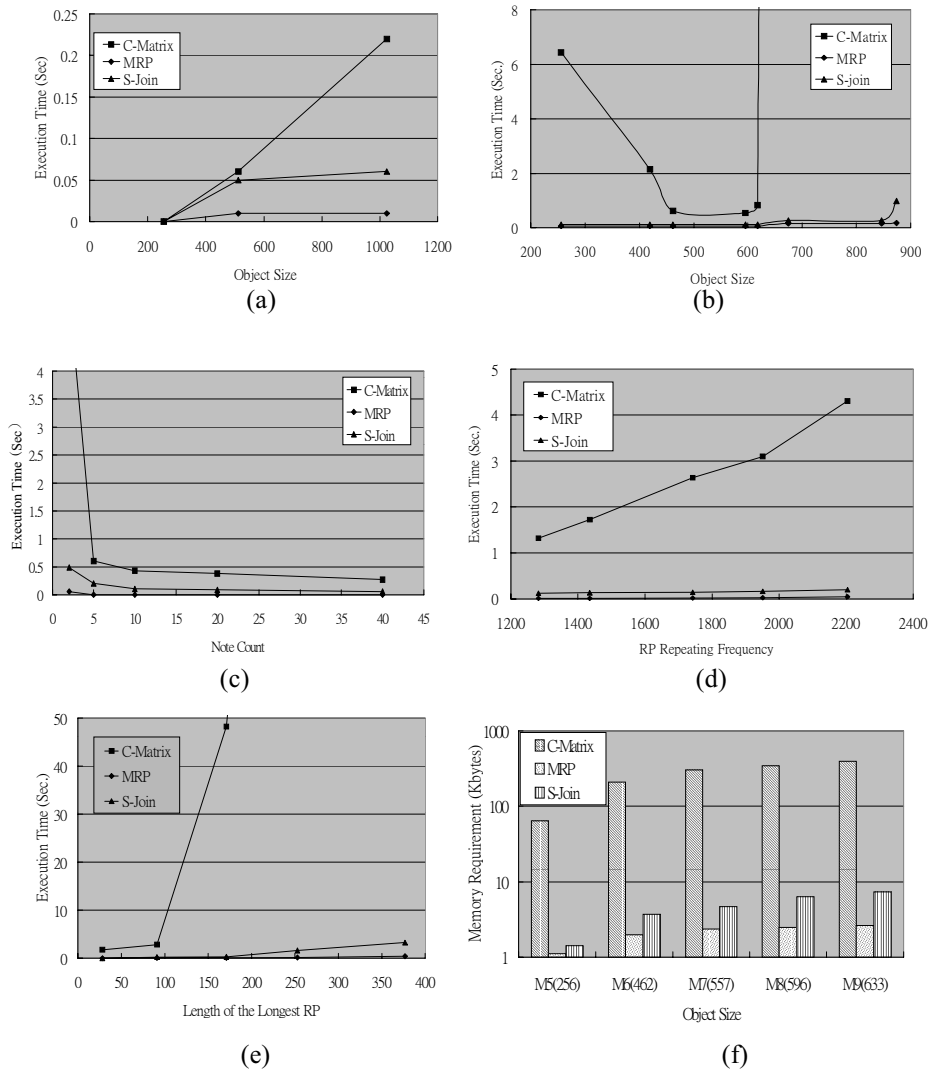

Fig. 1. (a) illustrates the execution time versus the object size of synthetic music objects. (b) shows the results of eight real music objects. (c) indicates that the execution time is inversely proportional to the note count. (d) reveals that the RP repeating frequency more significantly influence the execution time of correlative matrix approach than the MRP algorithm and the string join method do. (e) reveals that all the execution times of the three algorithms increase with an increasing length of the longest repeating pattern. (f) shows that the size of memory requirement for MRP algorithm is the least among the three algorithms.

6. Conclusion

This paper presents a novel means of mining the maximum repeating patterns in the melody of a music object. With the design of bit index sequence representation, the repeating frequency of a candidate pattern can be counted efficiently. In addition, the proposed approach is extended to extract frequent note sequences in a set of music objects. With the aid of appearing bit sequence representation and note combination structure, MFNS algorithm is proposed for generating candidate sequences and verifying the frequent note sequences efficiently.

Experimental results indicate that both execution time and memory requirement of the proposed MRP algorithm are less than the other two related works for mining maximum repeating patterns. Moreover, our approach is extended to mine the frequent note sequences in a set of music objects, which has seldom been mentioned previously.

From the extracted repeating patterns, a future work should address issues on music data clustering and classification. Furthermore, the music association rules should be analyzed for frequent note sequences to develop the intelligent agent so that similar music objects can be automatically retrieved. Finally, a future application will extend the proposed techniques to DNA and protein sequence mining.

References

1. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proc. 20th International Conference on Very Large Data Bases,* 1994.
2. R. Agrawal and R. Srikant, "Mining Sequential Patterns," in *Proc. the IEEE International Conference on Data Engineering (ICDE),* Taipei, Taiwan, 1995.
3. C. Bettini, S. Wang, S. Jajodia, and J.-L. Lin, "Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences," *IEEE Trans. on Knowledge and Data Eng.*, vol. 10, no. 2, 1998.
4. M.S. Chen, J. Han and P.S. Yu, "Data Mining: an Overview from a Database Perspective," *IEEE Trans. Knowledge and Data Eng.*, Vol. 8, No. 6, Dec.1996.
5. J.-L. Hsu, C.-C. Liu, and A.L.P Chen, "Efficient Repeating Pattern Finding in Music Databases," in *Proc. the 1998 ACM 7th International Conference on Information and Knowledge Management (CIKM'98),* 1998.
6. Roberto J. and Bayardo Jr., "Efficiently Mining Long Patterns from Databases," in *Proc. ACM SIGMOD International Conference on Management of Data,* 1998.
7. J.-L. Koh and W.D.C. Yu, "Efficient Repeating and Frequent Sequential Patterns Mining in Music Databases," Technique report in Department of information and computer education, National Taiwan Normal University.
8. C.-C. Liu, J.-L. Hsu and A.L.P. Chen, "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases," in *Proc. IEEE International Conference on Data Engineering,* 1999.
9. K. Wang, "Discovering Patterns from Large and Dynamic Sequential Data," *Journal of Intelligent Information Systems (JIIS)*, Vol. 9, No. 1, 1997.