A Position-Join Method for Mining Maximum-Length Repeating Patterns in Music Databases

Ye-In Chang, Chia-En Li and Tien-Hsiu Chen
Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
changyi@cse.nsysu.edu.tw

Abstract—**In recent years, the music has become popular due to the evolution of the technology. Many researches consider the music object as a continuously discrete note in time order. Repeating patterns are some subsequences which appear frequently in the music sequence. It can be utilized in music classification. Many methods have been proposed for mining the repeating patterns in music objects, for example, the M2P (Mining Maximum-length Patterns) method. It constructs a directed graph and uses the depth-first search to traverse the graph. It calculates the pathes by the string matching algorithm to decide whether they are repeating patterns, and finds out the maximum-length repeating pattern in a music sequence. It consumes time in creating too many candidate patterns and performing the string matching algorithm. Therefore, in this paper, we propose the PJ (Position-Join) method to efficiently find out the maximum-length repeating pattern. From our performance study based on the synthetic data and real music data, we show that our proposed PJ method is more efficient than the M2P method.**

Index Terms —**Data mining, Depth first search, Music databases, Music sequence, Repeating pattern**

I. INTRODUCTION

In recent years, as the number of music databases grows rapidly, the searching and indexing techniques for content-based audio data are getting more attention in the area of music databases. Development of representation types that can satisfy both semantic as well as efficiency requirements for retrieval has became more important. The repeating patterns can constitute a useful representation for the music data.

A. Music Representation

Recently, issues regarding content-based audio data retrieval have been studied [13]. The most popularity content-based audio data is the MIDI (Musical Instrument Digital Interface) format. According to the MIDI standard, each note is composed by two events: Onset events and Offset events. Onset events means the event of pronounced note, and Offset events means the event of unpronounced note. They both contain several parameters: start time, pitch, velocity, *etc.*. The pitch values are non-negative integers smaller than 128. It is based on the standard keyboard which has 128 keys, and each keystroke corresponds to a value. For example, if the note name C is middle C, the corresponding pitch values are shown in Figure 1. The difference of the start time of Onset event and the Offset events means the beat of this note. If each two start time of an event is nearness, it means that the music object plays on a fast tempo. In [4, 5, 6], they say that the repeating pattern is a characteristic representation type, which can represent the theme of a music. Many researches in musicology and music psychology consent that the repeating pattern is one of general features in music structure modeling [1].

B. The Repeating Patterns

The repeating patterns mean that segments of the music object that appear repeatedly. In [5], the repeating pattern is defined as follows: for a substring X of a sequence of notes S, if X appears more than once in S, they call X a *repeating pattern* of S. As shown in Figure 2, the main melody of

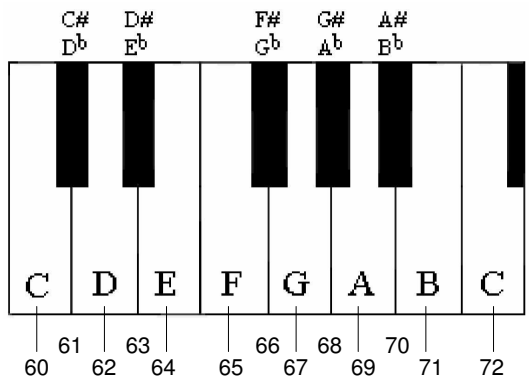


Fig. 1. MIDI format pitch value and the related position on piano keyboards

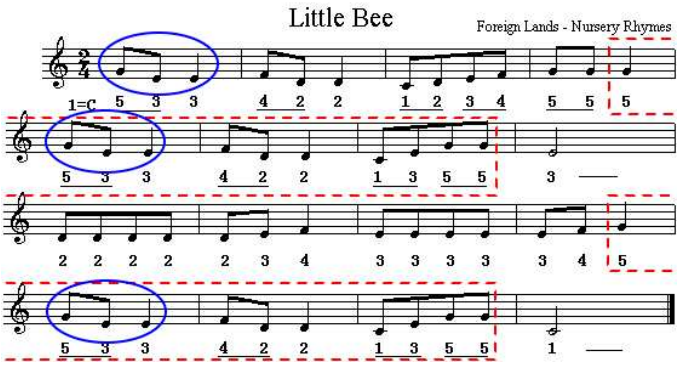


Fig. 2. An example of MLRP

"*Little Bee*", the sequences framed with the solid line oval is an instance of repeating patterns. In pop music, a refrain is a typical case of a repeating pattern. There are many different types of repeating patterns, such as non-trivial repeating patterns, polyphonic repeating patterns [2], approximate repeating patterns, maximum-length repeating patterns [6], vertical patterns [3] and geometrical patterns [12].

C. Motivation

Many methods have been proposed to find the repeating patterns in the music sequence. For finding maximum-length repeating patterns *i.e.*, MLRPs. Karydis *et al.* proposed the M2P method to find out the MLRPs specially [6]. The M2P method uses the music sequence to construct a directed graph. Then, it uses the depth-first search method to traverse the graph, and find out the MLRPs. However, the M2P method has four problems in finding MLRPs. First, it ignores the characteristics of the graph. If there are many candidate patterns which need to be calculated, it costs long time to decide whether the pattern can match the music sequence or not by using the string matching algorithm. Second, the M2P method will generate a huge number of candidate patterns which need to be calculated their frequency. It is because the M2P method does not use the filter method while it traverses the graph. For example, as shown in Figure 3, if the current path is "A→B→C", the M2P method may still down traverse the next vertex D, E, when the length of current path is shorter than the length of the current MLRP. Even if the path "A→B→C" is not a repeating pattern. Third, it costs time to calculate some similar patterns. Fourth, after the end of traversing the graph, the M2P method will have many candidate patterns with their length equal to the length of MLRP. They store these candidate patterns in a queue, and each candidate pattern in the queue needs to be decided whether it is repeating pattern. If the number of patterns in the queue is huge, it takes long time in executing the string matching algorithm.

To avoid these problems, in this paper, we propose the PJ method to find out the MLRPs in the music sequence efficiently. We modify the matrix and the graph by recording some information. Based on the modified matrix and graph, we use our proposed *p-join* method, instead of using the string matching algorithm, to efficiently calculate the frequency of a pattern. We also use the characteristics of the *p-join* method to avoid traversing some path repeatedly by dynamically modifying the graph. Moreover, we avoid to add candidate patterns into a queue. From our performance study based on the synthetic data and real data, we show that our proposed PJ method is more efficient than the M2P method.

The rest of the paper is organized as follows. Section 2 gives a survey of some methods of mining repeating patterns. Section 3 presents the proposed method, the Position-Join method. Section 4 gives the performance of the proposed method and makes a comparison between our method and the M2P method. Finally, we give a conclusion and point out some future research directions in Section 5.

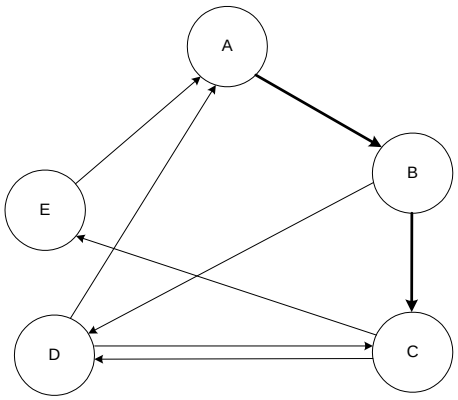


Fig. 3. Current path "A→B→C" may need to traverse the next vertex D or E

II. RELATED WORKS

A repeating pattern is defined as a sequence of notes which appears more than once in a music object. Mining repeating patterns is the task of discovering all these kinds of patterns. Several methods have been proposed to discover repeating patterns [10]. Recent research has employed data mining techniques [5, 7] to efficient discovery of repeating patterns. In this section, we introduce some methods of mining repeating patterns, including Correlative Matrix and String-Join [4, 7] and TRP [11] method for mining non-trivial repeating patterns, A-PRPD and T-PRPD with Bit-String [2] method for mining polyphonic repeating patterns, Ning-Han Liu *et al.* [8] method for mining approximate repeating patterns, and the M2P [6] method for mining maximum-length repeating patterns. In [4], Hsu *et al.* have proposed a method called correlative matrix to discover the non-trivial repeating pattern. The method is done by using an upper-triangular matrix to compute the repeating parts. In Hsu *et al.*'s method, first, it constructs the correlative matrix row by row. Let $T_{i,j}$ indicate the value of the cells located in this matrix at the conjunction of the i-th row and the j-th column. If the symbol in the first place of row i is the same as the symbol in the first place of column j, the value of the $T_{i,j}$ can be computed by the equation of $T_{i,j} = T_{(i-1),(j-1)} + 1$. In [7], Liu *et al.* proposed a string-join method and *RP-tree* to discover the non-trivial repeating pattern. The method is based on repeatedly joining two shorter repeating patterns to form a longer one. They use the form {X, $freq(X)$, $(position1, position2, ...)$} to represent the repeating patterns. In [11], Lo *et al.* proposed a true suffix tree method to discover the non-trivial repeating pattern. The method is based on converting the music sequence into the suffix tree. In [2], Chiu *et al.* proposed two methods, *A-PRPD* and *T-PRPD* with bit-string, to discover polyphonic repeating patterns. A polyphonic repeating pattern means that the pattern is consisted of more than one voice or notes in the stave. In [8], Liu *et al.* proposed a method to discover the approximate repeating pattern (abbreviated as *ARP*), which is defined in [9]. The maximum-length repeating pattern [6] is defined as follows: For all repeating patterns in the music sequence S, there does not exist another repeating pattern X' for which length of $X' >$ length of X, then X is a maximum-length repeating pattern. For example, in a sequence $S = ADBCEFADBEFADCDBCEADBC$, $ADBC$, $EFAD$, and $DBCE$ are the maximum-length repeating patterns. I. Karydis *et al.* [6] proposed a Mining Maximum-length Patterns (M2P for short) method to find the maximum length repeating pattern. It has two parts as follows:

Step 1: Construct the graph $G(V, E)$**.** For input sequence $S = \{S_1, ..., S_n\}$ with length *n*, the first step is to convert the sequence into repeating patterns of length two, and use these repeating patterns to construct directed graph.

Step 2: Traverse the graph $G(V, E)$**.** By using the depth-first manner, the M2P method traverses each vertex in Graph G to find out the satisfied longest path. The method traverses G by searching for the paths that originate from any of its vertices. While encountering paths, the M2P method is concerned in identifying only these which are candidates to become a MLRP. During the traversal, it keeps track of the path C that has already been visited and: (1) has, so far, the maximum length, and (2) corresponds to a repeating pattern.

III. THE POSITION-JOIN METHOD

In this section, we present the proposed PJ (Position-Join) method which improves the M2P method [6].

A. Notations and Definitions

In this subsection, we define some basic notations and definitions in the PJ method. As in [6], we consider a music object or a music sequence to be a sequence of symbols from alphabets containing discrete elements. In musicology, a music object is characterized by several features, such as pitch, chord. We focus on the pitch because it carries the high relative weight of information. Note that in the MIDI format, the size of alphabets is 128.

Notation 1: Repeating Patterns In the PJ method, we consider a music sequence as consecutive symbols and the position starting from 0, as shown in Figure 4, the main melody and the position. We use the positions to represent a pattern as $\{p_1, p_2, ..., p_n : plen\}$, where p_1 to p_n mean the start positions of the pattern in the music sequence, and the frequency is n. $plen$ is the length of the pattern. If the number of frequency is greater than or equal to two, it is a repeating pattern. For example, the representation of repeating pattern $\langle$GEE$\rangle$ is $\{0, 13, 38 : 3\}$ in Figure 4.

Notation 2: (Edge) The edges are represented the same way as patterns because each edge is a repeating pattern with length 2. If an edge is formed by vertex A linking to vertex B, we represent it as $EG(AB) : \{p_1, p_2, ..., p_m : 2\}$, where p_1 to p_m are the start positions of the substring $\langle AB\rangle$ in the music sequence, and the frequency is m. The length is always equal to 2.

Notation 3: (Path) The path is composed by many edges. We use the format $RP(path) : \{p_1, p_2, ..., p_n : plen\}$ to represent the current path, where p_1 to p_n are the start positions of the substring in the music sequence, and the frequency is n. $plen$ is the length of the current path.

Notation 4: (Terminal Edge) In the PJ method, for each original vertex, if it has been traversed, we create terminal edges which store the information of the current path. The terminal edge is used to avoid that when we traverse the same vertex next time. We need not to traverse the same path again. The terminal edge is represented as $TR(path) : \{p_1, p_2, ..., p_t : plen\}$, where p_1 to p_t are the start positions of the substring in the music sequence, and the frequency is t. $plen$ is the length of the terminal edge.

```
G E E F D D C D E F G G G G E E F D D C E G G E D
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

D D D D E F E E E E E F G G E E F D D C E G G C
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
```

Fig. 4. The music sequence of song "Little Bee"

Definition 1: (MLRPs) The Maximum-Length Repeating Patterns in the music sequence.

Definition 2: (MLQ) Maximum Length Queue which stores the current MLRPs while traversing.

Definition 3: (CML) Current Maximum Length which stores the value of the length in MLQ.

B. Constructing the Graph

In this subsection, we describe the PJ method to solve the mining maximum-length repeating patterns problem. The PJ method calculates the frequency of substring by using the *p-join method*, instead of the string matching algorithm. The main concept of the PJ method has two steps: first, we construct a directed graph. Second, we traverse the graph to find out the MLRPs.

In the constructing graph phase, as shown in Figure 5 line 5 and in Figure 6, we use the music sequence to construct a directed graph, and traverse the directed graph to find out the longest satisfied paths which stand for maximum-length repeating patters. Let music sequence $S = \langle s_0, s_1, s_2, ..., s_n\rangle$ with length n, and a two dimensional $m \times m$ modified adjacent matrix $AdjT$, where m is the size of alphabets. The modified $AdjT$ is used to store the start position of each repeating pattern with length two.

First, for every two continuous symbols s_i and s_{i+1} ($0 \leq i \leq n-1$) of music sequence S, we add its start position i into $AdjT[s_i][s_{i+1}]$. In detail, we scan S and get the foremost substring $\langle s_0, s_1\rangle$, and then we add its start position 0 into $AdjT[s_0][s_1]$. Next time, we get the substring $\langle s_1, s_2\rangle$ and add its start position 1 into $AdjT[s_1][s_2]$, and so do substring $\langle s_2, s_3\rangle$, $\langle s_3, s_4\rangle$, ..., $\langle s_{n-1}, s_n\rangle$. For each element in $AdjT$, the numbers are in order. As shown in Figure 7, the position sequence " $4, 17, 24, 25, 26, 27, 42$ " is in order.

Second, after all substrings have been processed and inserted into the $AdjT$, we use the matrix $AdjT$ to construct the graph as follows. For each element

1: **Procedure** $PositionJoin$ (S);
2: /* Input: A music sequence S. */
3: /* Output: The set of $MLRP$. */
4: **begin**
5: For every two continuous symbols x, y of S, insert the $position$ into Adjacent matrix $AdjT[x][y]$; If Count($AdjT[x][y]$) ≥ 2, create a vertex x linking to y, and the graph G could be constructed;
6: $MLQ := \phi$;
7: $MLQ.max\_length$:= 2;
8: CurrentPath $CP := \phi$;
9: **for each** Vertex $VStart \in G$ **do**
/* Start to Traversing the Graph */
10: **begin**
11: $Vstart.checked$:= false;
12: $CP.length$:= 2;
13: $TraverseG(Vstart, CP)$;
14: $Vstart.checked$:= true;
15: **end**;
16: **for each** $MLRP \in MLQ$ **do**
17: output $MLRP$;
18: **end**;

Fig. 5. Procedure $PositionJoin$

$AdjT[x][y]$, where $1 \leq x \leq m$ and $1 \leq y \leq m$, we count its numbers to decide its frequency. If it is greater than or equal to two, which means that it is a repeating pattern, we create two vertexes and an edge, where the vertex with value x links to the vertex with value y. Here, we need to notice that each edge in the graph is a directed edge. An example of the input Figure 4 is shown in Figure 7. Each edge in the graph is a directed edge. The dotted line represents that the edge in G corresponds to its position in the music sequence. For example, the edge $\langle F \rightarrow D \rangle$ appears in positions 3, 6, 41 in the music sequence.

The conceptual graph is shown in Figure 8. In implementation, we do not create the edges which store the positions because it will have duplicated information corresponding to the $AdjT$. For example, if we need the information of edge $\langle F \rightarrow D \rangle$, we can get the information by accessing $AdjT[F][D]$.

C. The P-Join Method

In the *p-join* phase, the *p-join* method calculates whether two repeating patterns can join together to become a new longer repeating pattern by their

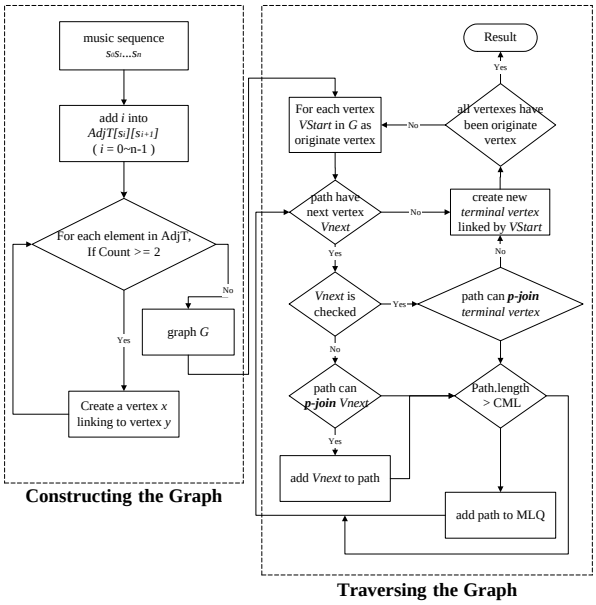
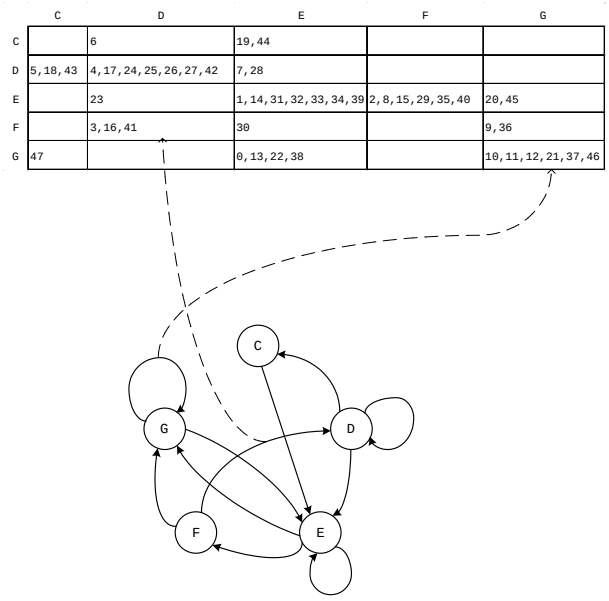


Fig. 6. The flowchart of the PJ method



	C	D	E	F	G
C		6	19,44		
D	5,18,43	4,17,24,25,26,27,42	7,28		
E		23	1,14,31,32,33,34,39	2,8,15,29,35,40	20,45
F		3,16,41	30		9,36
G	47		0,13,22,38		10,11,12,21,37,46

Fig. 7. The matrix $AdjT$ and the graph for song "Little Bee"

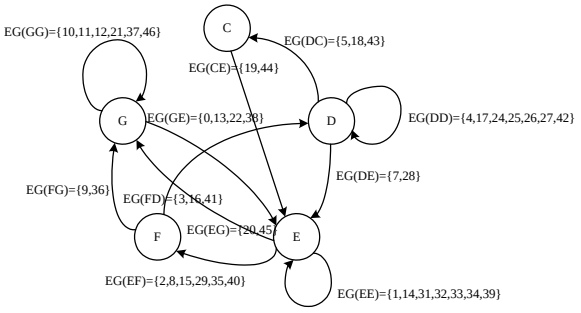


Fig. 8. The conceptual graph for song "Little Bee"

positions. The concept of the *p-join* method is that a substring Y can append at the tail of another substring X if the first symbol of Y and the last symbol of X have the same position. For example, given substring "ABCD" starting at position 0 and substring "DC" starting at position 3, then the tail position of substring "ABCD" is 3 and it is equals to the start position of "DC". These two substrings can join together.

Assume that a substring (or pattern) X is represented as $\{x_{p1}, x_{p2}, x_{p3}, ..., x_{pn} : xlen\}$. A substring (or pattern) Y is represented as $\{y_{p1}, y_{p2}, y_{p3}, ..., y_{pm} : ylen\}$. We define that substring X can *p-join* substring Y as follows. There are two steps.

Step 1: Create the candidate repeating pattern. For each position in X, in order to get the tail position, we add $xlen$-1 to it and get X'=$\{x_{p1}+xlen-1$, $x_{p2}+xlen-1$, $x_{p3}+xlen-1$, ... , $x_{pn}+xlen-1\}$. We examine whether there is the same value between X' and Y, and store its corresponding value of X to a candidate pattern (abbreviated as $candRP$). In other words, if y_i=$x_j + xlen - 1$ ($i \leq m$ and $j \leq n$), we add its position x_j into $candRP$.

Step 2: Check the candidate repeating pattern. The *p-join* method needs to check whether the candidate repeating pattern is a repeating pattern or not. As previous discussion, a repeating pattern needs appearing in the music sequence at least twice. We calculate the frequency of the $candRP$ by counting the number of positions in $candRP$. Between the count and two, there are two cases needed to be considered. For Case 1: if the count is less than 2, the $candRP$ is not a repeating pattern. For Case 2: if the count is greater or equal to 2, we assume that the first start position of the $candRP$ is x_{p1} and the last start position is x_{pn}. (1) If x_{pn}-x_{p1} is greater or equal to the sum of the length of X and the length of Y (*i.e.*, $xlen$+$ylen$-1), then the $candRP$ is a repeating pattern with length equals to $xlen$+$ylen$-1. (2) If x_{pn}-x_{p1} is less than $xlen$+$ylen$-1, it means that there is an overlap in $candRP$, we modify the length of $candRP$ to x_{pn}-x_{p1}. If length of $candRP$ equals to length of X, the $candRP$ is not a repeating pattern. If length of $candRP$ is greater than the length of X, the $candRP$ is a repeating pattern.

D. Traversing the Graph

In the graph G, each path P can be considered as a possible repeating pattern, since all its sub-paths of length two (*i.e.*, the directed edges) are repeating patterns. The set of all possible paths of G forms the search space of the examined problem. We traverse G by searching for the paths that originate from any of its vertices, and use the depth-first manner to traverse the graph in order to find the path as long as possible.

As discussed before, we use the format $RP(path) : \{p_1, p_2, ..., p_n : plen\}$ to represent the current path. Procedure is shown in Figure 5, lines 6 to 15, Figure 6 and Figure 9. Initially, CML is set to 2, MLQ is set to empty, and all vertexes are setting unchecked. While the PJ method traverses the graph G, the current path visits the next vertex and calculates whether the current path can continue to visit the next vertex or not. We use the *p-join* method. If the current path can *p-join* the edge, it means that it can be a repeating pattern, then we add the vertex to the current path and compare its length to CML. There are three cases: (1) The length of the path $> CML$. (2) The length of the path $= CML$. (3) The length of the path $< CML$. For Case 1, it means that the current path has longer length than MLRPs in MLQ. Therefore, we clean MLQ, add the path to MLQ, and set CML equals to the length of the path. For Case 2, the current path has length equals to MLRPs in MLQ, so we add the path to MLQ and continue to visit the next vertex. For Case 3, we just continue to visit the next vertex.

Each time, when the path can not visit the next vertex anymore, we set the original vertex checked, and create a terminal edge which stores the path and the length of the path. The terminal edge is linked by the original vertex. The terminal edge is used to avoid the case that we traverse the same path. In other words, in the traversing step, when traversing the vertex which is the formerly original vertex, as shown in Figure 9 line 16, we only calculate its terminal edges whether the current path can *p-join* them. Otherwise, we still use the depth first search to traverse the graph.

When the PJ method has ended all the traversal, MLRPs in MLQ are all answers, as shown in Figure

```
1: Procedure TraverseG (V, CP);
        /* V : Vertex, CP : CurrentPath */
2: begin
3:    for each Vertex Vnext ∈(V → Vnext) do
4:    begin
5:       if Vnext.checked = false then
          /* Case 1: Vertex is not checked */
6:          if CP can p-join Enext then
7:          begin
8:             store satisfied position to CP.position;
9:             CP.length = CP.length + 1;
10:            TraverseG(Vnext, CP);
11:         end
12:         else
13:            create a new terminal edge Enew
               Constructed by CP;
14:         end
15:      else
          /* Case 2: Vertex is checked */
16:         for each Vnext → Vnew do
17:            if CP can p-join Enew then
18:            begin
19:               store satisfied position to
                  CP.position;
20:               CP.length = CP.length +
                  Vnew.length - 1;
21:               create a new terminal edge Enew
                  Constructed by CP;
22:            end;
23:         end;
24:      end;
25:   end;
26:   if CP.length > MLQ.max_length then
27:   begin
28:      clean MLQ;
29:      add CP to MLQ;
30:      MLQ.max_length := CP.length;
31:   end
32:   else if CP.length = MLQ.max_length then
33:         add CP to MLQ;
34:   end;
35: end;
```

Fig. 9. Procedure $TraverseG$

5, lines 16 to 19. From Figure 10 to Figure 14 are the process of traversing the graph of Figure 8. For each figure, (a) is the process of traversal from the original vertex; (b) is its terminal edges after being ended the traversal from the original vertex, and the double circled means that the vertex set checked; (c) is the current MLQ and CML; (d) is the conceptual graph which after the original vertex set checked.

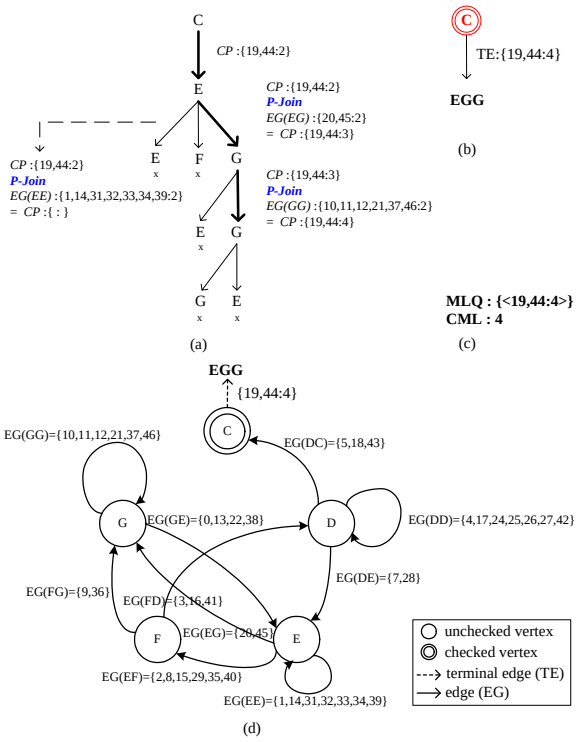


Fig. 10. The process of traversing the graph: (a) traversal of original vertex C; (b) the terminal edge of C; (c) MLQ and CML; (d) the conceptual graph.

Finally, the information in the MLQ is the answer. In Figure 14, the answer is $\{< 12, 37 : 11 >\}$. The pattern appears in positions 12 and 37 with length 11 in Figure 4 is ”GGEEFDDCEGG”, and it is the maximum-length repeating pattern.

E. The Terminal Edge

In the PJ method, in order to avoid repeatedly traversing some paths, we propose the concept of creating terminal edge. Initially, all vertexes in the graph are setting unchecked. We start to traverse the graph from any of its originate vertices. Each time, when the current path CP visits the next vertex and can *p-join* the edge, then it means that CP can still down traverse, because it is not the terminal. If all next edges of CP can not be *p-joined*, then we create a terminal edge and store the information of CP. After the original vertex has been traversed, the original vertex is set checked. An illustration is shown in Figure 15.

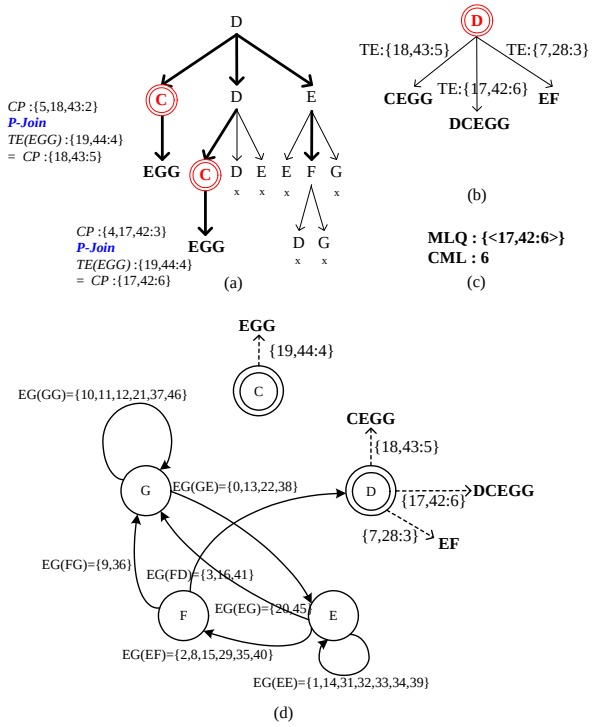


Fig. 11. The process of traversing the graph: (a) traversal of original vertex D; (b) the terminal edges of D; (c) MLQ and CML; (d) the conceptual graph.

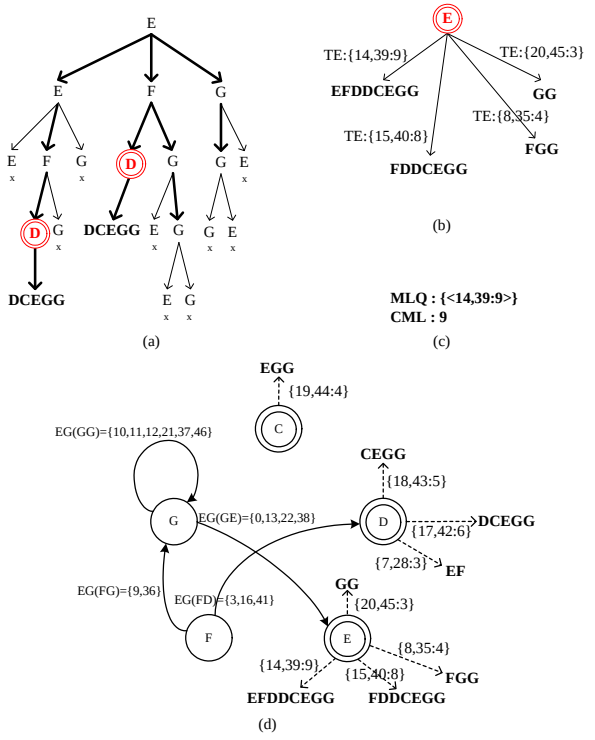


Fig. 12. The process of traversing the graph: (a) traversal of original vertex E; (b) the terminal edges of E; (c) MLQ and CML; (d) the conceptual graph.

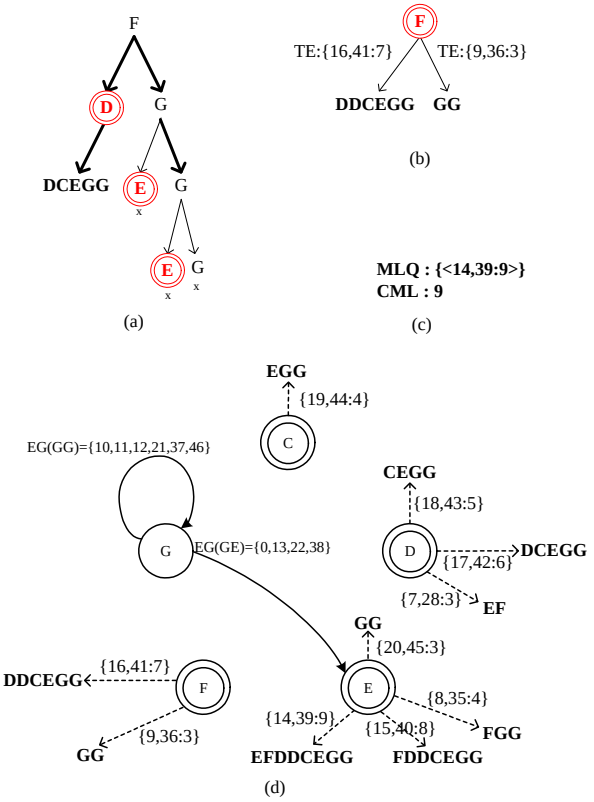


Fig. 13. The process of traversing the graph: (a) traversal of original vertex F; (b) the terminal edges of F; (c) MLQ and CML; (d) the conceptual graph.

TABLE I
PARAMETERS USED IN THE EXPERIMENT

Parameters	Meaning
Ls	The length of the music sequence
Nc	The note counts in the music sequence
Lmlrp	The length of MLRP in the music sequence
Fmlrp	The frequency of MLRP in the music sequence

IV. PERFORMANCE

In this section, we study the performance of the proposed Position-Join method. We also make a comparison with the M2P method. The simulation was performed on an Intel Pentium Core2 Duo 2.66G Hz CPU computer with 1.99GB of RAM, running Windows XP, and compiled by JDK 1.5.0.

A. Generation of Experimental Data

In order to evaluate the performance of the proposed algorithm, we generate synthetic data sets by different parameters and extract the music sequence from real music objects in the web site.

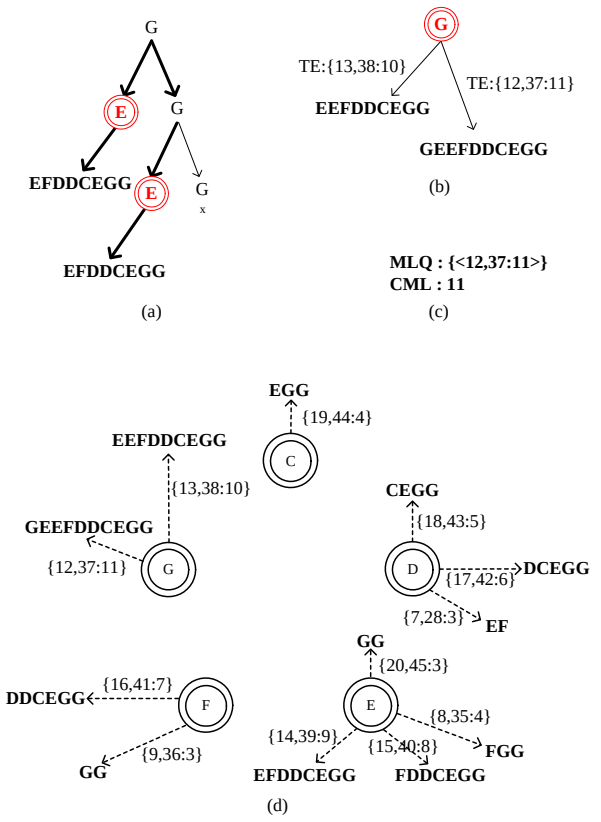


Fig. 14. The process of traversing the graph: (a) traversal of original vertex G; (b) the terminal edges of G; (c) MLQ and CML; (d) the conceptual graph.

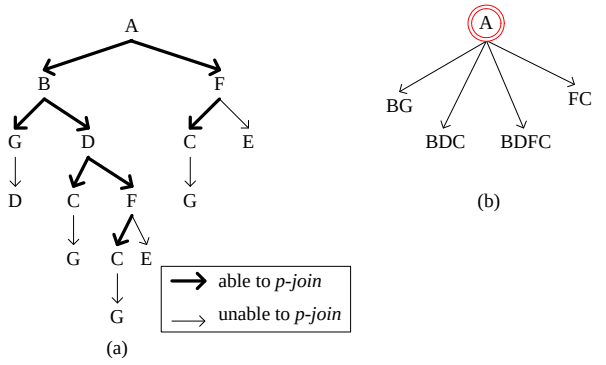


Fig. 15. An example of the terminal edge: (a) traversal of original vertex A; (b) the terminal edges of A.

The first set of experiments was conducted on the synthetic data sets, which a sequence with repeating patterns. The parameters used in the generation of the synthetic data are shown in Table I. There is a sequence which has length equals to *Ls* and the note count is *Nc*. The second set of experiments was conducted on the real music sequence extracted by the MIDI file music object in the web site (http://content.edu.tw/senior/music/tn_nn/%A5j%A8%E5midi/midi.html). We use these data sets to run our method, and compare the execution time with the M2P method.

B. Synthetic Data

In this subsection, we create synthetic data to compare the experiment results between the PJ method and the M2P method. For the synthetic data, as shown in Table I, the parameter (*Ls*) means that the length of the music sequence which we used as input. For example, as shown in Figure 4, the (*Ls*) of ”Little Bee” is 49. The second parameter *Nc* means that the number of different notes in the music sequence. For example, if a song is ”Fa-Fa-Mi-Do-Re-Do”, we have *Nc* = 4. In the MIDI file format music object, the range of the note is between 1 to 128 as discussed previously. The parameter *Lmlrp* means that the length of the maximum-length repeating pattern in the music sequence. For example as shown in Figure 4, the *Lmlrp* is 11 as discussed before. The last parameter *Fmlrp* means that the times of MLRP appear in the music sequence. For the same example, the MLRP appears only in positions 12 and 37, so we have *Fmlrp* = 2.

There are five cases: (1) keeping the values of *Ls*, *Nc*, *Fmlrp*, and changing the value of *Lmlrp*; (2) keeping the values of *Ls*, *Lmlrp*, *Fmlrp*, and changing the value of *Nc*; (3) keeping the values of *Nc*, *Lmlrp*, *Fmlrp*, and changing the value of *Ls*; (4) keeping the values of *Ls*, *Nc*, *Lmlrp*, and changing the value of *Fmlrp*; (5) comparing the PJ method with using terminal edges and the PJ method without using terminal edges.

For Case 1, we set *Ls* = 1000, *Nc* = 30, *Fmlrp* = 2 and changing the value of *Lmlrp* in the music sequence from 10 to 100. The *Nc* is set to 30, because most of music objects is in the bound of three Perfect Octaves (3×12). The experiment results is shown in Figure 16. We find out that the PJ method is better than the M2P method in each *Lmlrp*. It is due to that the M2P method needs to use the string matching algorithm (the KMP algorithm) to calculate the frequency of patterns.

For Case 2, we set *Ls* = 1000, *Fmlrp* = 2, and changing the value of *Nc* from 30 to 50 to see the

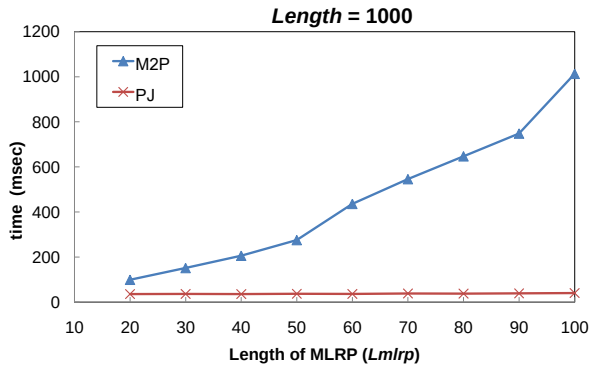


Fig. 16. A comparison of the processing time under different values of *Lmlrp*

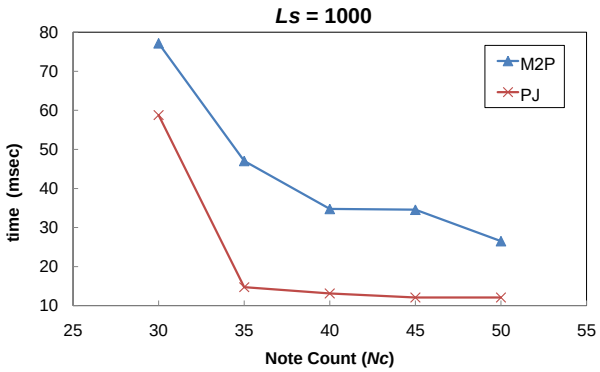


Fig. 17. A comparison of the processing time under different values of *Nc*

impact of different values of *Nc* between the two methods. The experiment result is shown in Figure 17. We find out that the PJ method is better than the M2P method in each *Lmlrp*. It is due to that the M2P method needs to use the string matching algorithm (the KMP algorithm) to calculate the frequency of patterns.

For Case 3, we set *Nc* = 30, *Lmlrp* = 100, *Fmlrp* = 2, and changing the value of *Ls* from 1000 to 5000 to see the impact of different length of the music sequence between the two methods. We find out that the PJ method is better than the M2P method in each *Lmlrp*. It is due to that the M2P method needs to use the string matching algorithm (the KMP algorithm) to calculate the frequency of patterns. Moreover, the execution time of both methods increases when the length of the sequence increases.

For Case 4, we set *Ls* = 1000, *Nc* = 30, *Lmlrp* = 10, and changing the value of *Fmlrp* from 4 to 6 to see the impact of different frequency of maximum-length repeating pattern between the two methods. As shown in Figure 18, we find out that the PJ method is better than the M2P method in each *Lmlrp*. It is due to that the M2P method needs to use the string matching algorithm (the KMP algorithm) to calculate the frequency of patterns.

Finally, we consider the impact of terminal edges in the PJ method. We compare two methods, the first method is the PJ method with using terminal edges, and second method is the PJ method without creating terminal edges and each vertex is considered unchecked (*abbreved* as PJ-woTE method). We set *nc* = 30, *Lmlrp* = 100, *Fmlrp* = 2, and changing

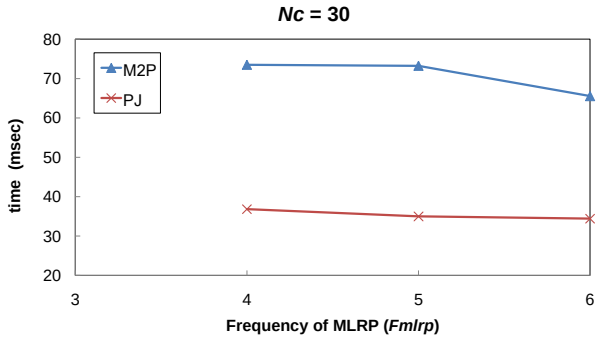


Fig. 18. A comparison of the processing time under different values of *Fmlrp*

the value of *Lmlrp*. The results are shown in Figure 19 and Figure 20. The PJ-woTE method needs to execute more number of the p-join method than the PJ method, because the PJ-woTE method just depth-first searches the graph and has many duplicated paths. Hence, the PJ-woTE method needs longer execution time than the PJ method.

C. Real Data

In this subsection, we use real music objects to compare the two methods. We employ the MIDI file format. First, because the MIDI format have 16 channels, each channel may represent different voices or different musical instruments, we choose the main melody channel. Second, the main melody channel is composed by many events, which includes information such as pitch, start time, end time, *etc*. We keep only the pitch information of the MIDI file format and get the pitch string. It means that we keep only the main melody string.

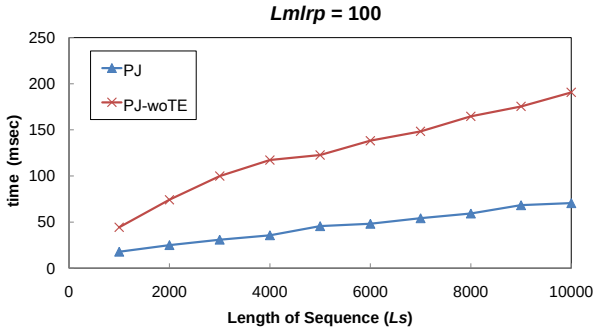


Fig. 19. A comparison of the processing time under different values of *Fmlrp*

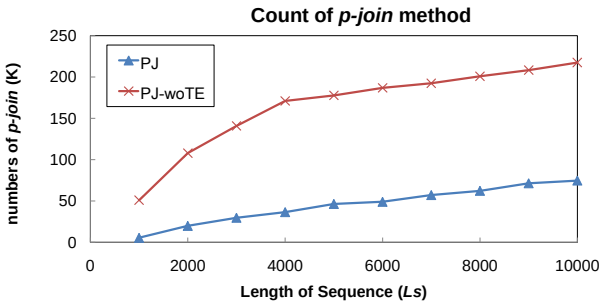


Fig. 20. A comparison of the number of counting RP

Different kinds of music objects contain distinct characteristics and have distinct MLRPs. We use different kinds of music objects as shown in Talbe II. There are three types of music objects, classical, modern, and nursery rhyme.

For the first real case, classical type, La Primavera (Spring), the music object has length = 949, and note count *Nc* = 18. Figure 21 demonstrates the execution time for varying the music object size, which *Nc* is always 18 in each object size. Moreover, Table III depicts the length of the discovered MLRPs with respect to the size of the music object.

TABLE II
REAL MUSIC OBJECTS USED IN THE EXPERIMENT

Type	Song	Composer
classical	La Primavera(Spring)	Antonio Vivaldi
classical	Canon	Pachelbel
modern	Tears in heaven	Eric Clapton
nursery rhyme	Twinkle, Twinkle, Little Star	Ann Taylor

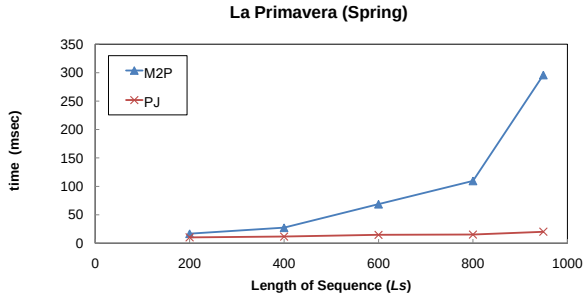


Fig. 21. Case 1: A comparison of the processing time for music "La Primavera"

TABLE III
CASE 1: LENGTH OF MLRPS *vs.* OBJECT LENGTH FOR MUSIC "LA PRIMAVERA"

Music object length	200	400	600	800	949
Length of MLRPs	29	30	30	51	56

For the second real case, classical type, Canon, the music object has length = 1746, and note count *Nc* = 32. We compare the execution time under length 200 to 1746, as shown in Figure 22. The reason that the PJ method has better performance than the M2P method is discussed in the previous section "Synthetic Data". As shown in Figure 22, the longer the *Ls* and *Lmlrp* are, the more time consuming for both methods are.

For the third real case, modern type, Tears in heaven, the music object has length = 2926, and note count *Nc* = 43. We compare the execution time under length 400 to 2926, as shown in Figure 23. The reason that the PJ method has better performance

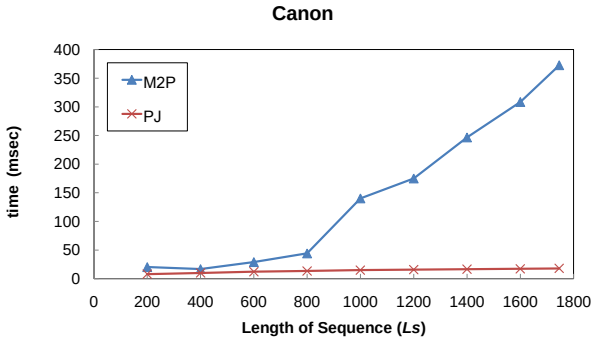


Fig. 22. Case 2: A comparison of the processing time for music "Canon"

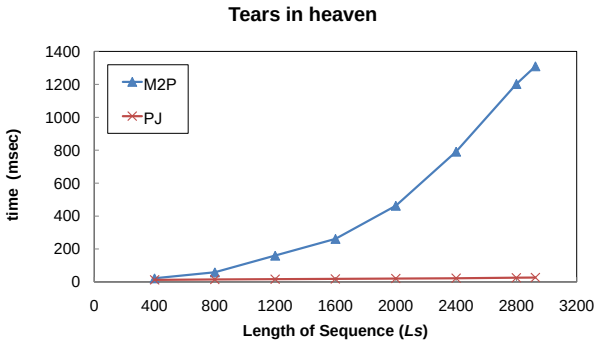


Fig. 23. Case 3: A comparison of the processing time for music "Tears in heaven"

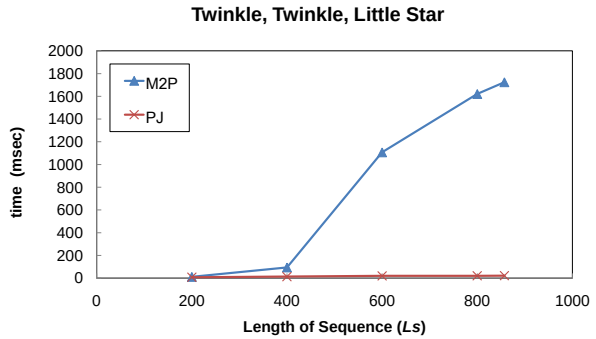


Fig. 24. Case 4: A comparison of the processing time for music "Twinkle, Twinkle, Little Star"

than the M2P method is discussed in the previous section "Synthetic Data". As shown in Figure 23, the longer the *Ls* is, the more time consuming for both methods is.

For the fourth real case, nursery rhyme type, "Twinkle, Twinkle, Little Star", the music object has length = 857, and note count *Nc* = 45. We compare the execution time under length 200 to 857, as shown in Figure 24. The reason that the PJ method has better performance than the M2P method is discussed in the previous section "Synthetic Data". As shown in Figure 24, the longer the *Ls* and *Lmlrp* are, the more time consuming for both methods are.

V. CONCLUSION

In the paper, we have developed the PJ method to mine maximum-length repeating patterns. We have avoided to traverse some paths repeatedly in traversing the graph step. From our performance study based on the synthetic data and real music data, we have shown that our proposed PJ method is more efficient than the M2P method.

VI. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-87-2213-E-110-014.

REFERENCES

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

[2] S. C. Chiu, M. K. Shan, J. L. Huang, and H. F. Li, "Mining Polyphonic Repeating Patterns from Music Data Using Bit-String Based Approaches," *Proc. of IEEE Int. Conf. on Multimedia and Expo*, pp. 1170–1173, 2009.

[3] D. Conklin, "Representation and Discovery of Vertical Patterns in Music," *Proc. of Int. Conf. on Music and Artificial Intelligence*, pp. 32–42, 2002.

[4] J. L. Hsu, C. C. Liu, and A. L. P. Chen, "Efficient Repeating Pattern Finding in Music Databases," *Proc. of the 7th Int. Conf. on Information and Knowledge Management*, pp. 281–288, 1998.

[5] J. L. Hsu, C. C. Liu, and A. L. P. Chen, "Discovering Non-trivial Repeating Patterns in Music Data," *IEEE Trans. on Multimedia*, pp. 311–325, Sept. 2001.

[6] I. Karydis, A. Nanopoulos, and Y. Manolopoulos, "Finding Maximum-Length Repeating Patterns in Music Databases," *Multimedia Tools and Applications*, Vol. 32, No. 1, pp. 49–71, Oct. 2007.

[7] C. C. Liu, J. L. Hsu, and A. L. P. Chen, "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Database," *Proc. of the 15th IEEE Int. Conf. on Data Eng.*, pp. 14–21, 1999.

[8] N. H. Liu, Y. H. Wu, and A. L. P. Chen, "An Efficient Approach to Extracting Approximate Repeating Patterns in Music Database," *Proc. of Int. Conf. on Database Systems for Advanced Applications*, pp. 240–252, 2005.

[9] N. H. Liu, Y. H. Wu, and A. L. P. Chen, "Identifying Prototypical Melodies by Extracting Approximate Repeating Patterns from Music Works," *Journal of Information Science and Eng.*, Vol. 26, No. 4, pp. 1181–1198, July 2010.

[10] Y. L. Lo and C. Y. Chen, "Fault Tolerant Non-Trivial Repeating Pattern Discovering for Music Data," *Proc. of Computer and Information Science, IEEE/ACIS Int. Workshop on Component-Based Software Eng.*, pp. 130–135, 2006.

[11] Y. Lo, W. Lee, and L. Chang, "True Suffix Tree Approach for Discovering Non-Trivial Repeating Patterns in a Music Object," *Multimedia Tools and Applications*, Vol. 37, No. 2, pp. 169–187, July 2007.

[12] D. Meredith, K. Lemstroumlm, and G. A. Wiggins, "Algorithms for Discovering Repeated Patterns in Multidimensional Representations of Polyphonic Music," *Journal of New Music Research*, Vol. 31, No. 4, pp. 321–345, 2003.

[13] E. Wold, T. Blum, D. Keislar, and J. Wheaton, "Content-Based Classification, Search, and Retrieval of Audio," *Proc. of IEEE Multimedia*, pp. 27–36, 1996.