An Efficient Approach to Extracting Approximate Repeating Patterns in Music Databases

Ning-Han Liu[1], Yi-Hung Wu[1], and Arbee L.P. Chen[2,*]

[1] Department of Computer Science, National Tsing Hua University, Hsinchu,Taiwan
[2] Department of Computer Science, National Chengchi University, Taipei, Taiwan
alpchen@cs.nccu.edu.tw

Abstract. Pattern extraction from music strings is an important problem. The patterns extracted from music strings can be used as features for music retrieval or analysis. Previous works on music pattern extraction only focus on exact repeating patterns. However, music segments with minor differences may sound similar. The concept of the prototypical melody has therefore been proposed to represent these similar music segments. In musicology, the number of music segments that are similar to a prototypical melody implies the importance degree of the prototypical melody to the music work. In this paper, a novel approach is developed to extract all the prototypical melodies in a music work. Our approach considers each music segment as a candidate for the prototypical melody and uses the edit distance to determine the set of music segments that are similar to this candidate. A lower bounding mechanism, which estimates the number of similar music segments for each candidate and prunes the impossible candidates is designed to speed up the process. Experiments are performed on a real data set and the results show a significant improvement of our approach over the existing approaches in the average response time.

1 Introduction

For content-based music retrieval and music style analysis, a fundamental requirement is to extract music features from the raw data of music works. One significant feature of the music work is the *structural feature*, which is described as follows. Consider the classical music works. Most of them are composed according to a particular structure named *musical form* in which there is a basic rule: *repetition rule* [5]. The repetition rule says that there exist specific sequences of notes, known as *motives*, repeating in a movement. For example, the well-known *motive* "G-G-G-E" repeatedly appears in Beethoven's Symphony No. 5. In the previous work [4], a sequence of notes appearing more than once in the music work is regarded as the structural feature and called the *repeating pattern*. Most of the researchers in the musicology agree that repetition is a universal characteristic in music structure and style analysis [5]. Moreover, the length

* Corresponding author

L. Zhou, B.C. Ooi, and X. Meng (Eds.): DASFAA 2005, LNCS 3453, pp. 240 – 252, 2005.

of a repeating pattern is much shorter than that of a music work. Therefore, using repeating patterns as music features meets both efficiency and effectiveness requirements for content-based music retrieval.

The problem of finding all the repeating patterns from a music work has been discussed in [2] with suffix-tree based solutions. Each of these approaches first builds a suffix-tree, where each path represents a pattern and each leaf node keeps all the positions of the corresponding pattern located in the music work. After traversing the suffix-tree, all the repeating patterns can be extracted. These approaches consider the patterns represented by different paths to be different. As a result, they only find exact repeating patterns instead of the repeating patterns composed of strings with minor differences. In [4], a repeating pattern that is not contained in any other repeating pattern with the same count is called *non-trivial.* Two approaches based on *correlative-matrix* and *string-join,* are proposed to extract non-trivial repeating patterns. The former approach lines up the notes of a music piece along the x-axis and y-axis respectively to form a correlative matrix and uses it to find all the non-trivial repeating patterns in the music piece. The latter approach joins shorter repeating patterns into longer ones and prunes the impossible candidates in between. Similarly, both of them only focus on finding exact repeating patterns. Shih et al. [11] also propose an algorithm for extracting repeating patterns from music databases. They segment a music score into bars, which are further encoded for efficiency. As a result, the computation cost for segment matching is reduced. Except for the encoding mechanism, this approach adopts the same concept as string-join.

One the other hand, a pattern may repeatedly appear in a music work with some variations. One popular concept to coordinate such variations is the *prototypical melody*, which is a kind of abstraction of the music work to which the corresponding music segments are similar [10]. The prototypical melody has a great impact on the way the actual melody is memorized by human. The main goal of this paper is to extract all the prototypical melodies called *approximate repeating patterns* from a music work. Pienimäki [8] considers the music transposition and adopts the algorithm on text mining to extract all the longest repeating patterns, i.e., the ones that are not contained in any others. This approach allows the extracted patterns to be discontinuous in the music piece. In this approach, shorter candidates are first generated with unqualified ones removed and then combined into longer ones. Experiments show that the execution time of this approach is considerable due to the huge number of candidates to be examined. Rolland [9] proposes a flexible similarity measure of music segments and a dynamic-programming method for extracting approximate repeating patterns. First, a music segment is regarded as a point in a graph and then the similarity between every two points in the graph is computed. After that, all the prototypical melodies are found by counting the number of similar music segments for each point in the graph. This approach costs a lot on computing the similarities among music segments. For example, given a music work with 200 notes, if the user restricts the length of a repeating pattern to the range from 10 to 100, the number of music segments involved will be (101+191)*91/2=13286. In this case, the number of similarity computations will be C_2^{13286}, which is close to 10^8. Moreover, the similarity computation for every two music segments is also time consuming since its time complexity is $O(|m|*|n|)$, where $|m|$ and $|n|$ denote the lengths of music segments m and n, respectively.

In this paper, we consider each music segment as a candidate *ARP* (namely, an approximate repeating pattern or a prototypical melody). Two constraints, the maximum and minimum pattern lengths, are set to filter out the candidates that are not interesting to the user. After that, for each candidate, we use the edit distance and a threshold to identify all the music segments that are similar to it. Finally, based on the number of similar music segments and how they overlap each other, we determine whether a candidate ARP is qualified to be an ARP. For efficiency, we design a modified R*-tree to prune impossible candidates before the computations of edit distances. We propose a novel distance measure to approximate the edit distance, by which we can reduce the number of similar segments for each candidate ARP. In addition, since it is difficult to set the above constraints and thresholds perfectly at the first time, enabling the user to tune them without rerunning the entire process is necessary. We call it the *interactive environment*. Our modified R*-tree can work in the interactive environment and avoid rerunning the entire algorithm. According to the experiment results, especially on the average response time, our approach outperforms Rolland's approach [9] in both the normal and interactive environments.

The remainder of this paper is organized as follows. In Section 2, we define the approximate repeating pattern and formulate the ARP extraction problem. Section 3 presents our approach to the ARP extraction problem. Section 4 shows the experiment results with discussions. Finally, Section 5 concludes this paper with future research directions exposed.

2 Problem Formulation

The problem of prototypical melody extraction has been defined in Rolland's work [9], where the pattern composed of music segments is called the *star-type pattern*. In a star-type pattern, one music segment is its origin called the *pivot* and the others are music segments similar to the pivot over a predefined threshold. In this paper, we regard a pivot as the prototypical melody if it is the origin of a star-type pattern. In the following, we formulate our problem, where more constraints are specified.

2.1 Data Representation

There are several symbolic representations in digital music. We choose the MIDI [7] representation because of its popularity. The melody of a music work includes two kinds of basic information, i.e., pitch and duration. Each note in a MIDI file can be represented as a triple (p,s,e) where p is the pitch value, s means the starting time of playing (i.e., *note on*) and e is the ending time of playing (i.e., *note off*). As a result, a MIDI file is an ordered list of triples sorted by the note on time, e.g., (p_1,s_1,e_1), (p_2,s_2,e_2), ……,(p_n,s_n,e_n) where $s_1 \leq s_2 \leq \ldots \leq s_n$. Two music pieces whose notes have the same pitches is often considered the same even though their notes have different durations. Therefore, in our approach, the order instead of the exact time is retained. Moreover, since two melodies with the same contour are considered the same, we use intervals as our representation, which is defined as follows.

Definition 2.1. Pitch String:
A *pitch string* $P=(p_1,p_2,\ldots p_m)$ is the ordered list of pitch values p_i retrieved from a MIDI file, where m is the string length denoted as $|P|=m$.

Definition 2.2. Interval String:
An *interval string* of a pitch string $P=(p_1,p_2,\ldots p_m)$ is defined as $D=(d_1,d_2,\ldots d_{m-1})$, where $d_i=p_{i+1}-p_i$, $1\leq i<m$ and d_i is called an *interval*.

The set of all the distinct interval values in D is denoted as $\sum_D$, whose size is denoted as $|\sum_D|$. Fig. 1 shows the examples of a pitch string and an interval string.

Definition 2.3. Interval Segment:
An *interval segment* $S[i:j]$ is a substring of an interval string $D=(d_1,d_2,\ldots d_n)$ from i to j, i.e., $S[i:j]=(d_i,d_{i+1},\ldots,d_j)$.

For the simplicity of presentation, in the remainder of this paper, we use *string* and *segment* to mean *interval string* and *interval segment*, respectively.

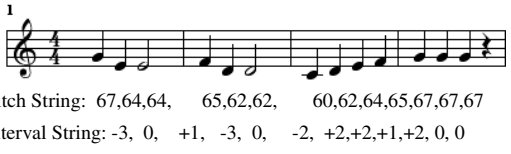

Fig. 1. A pitch string and an interval string

2.2 Approximate Repeating Patterns

If there is no constraint on the music patterns, too many patterns will be extracted and some of them can be uninteresting to user, e.g., too short and too long patterns. Therefore, we define five constraints that can filter out unimportant music patterns as follows.

In a music work, too long segments tend to contain duplicate information, while too short segments often have little information about the music semantics. Therefore, allowing users to specify constraints on the pattern length will reduce the unnecessary costs on duplicate information and a large amount of very short segments. In this paper, we use two constraints on the pattern length, called the *maximum length* (*max_len*) and the *minimum length* (*min_len*), respectively. As a result, segments are generated from the given string by using sliding widows whose sizes are from *min_len* to *max_len*. For example, given a string (a, b, c, d), the qualified segments are (a, b), (b, c), (c, d), (a, b, c), (b, c, d) when *min_len*=2 and *max_len*=3. Furthermore, we adopt the *edit distance* to measure the similarity degree between two segments.

Definition 2.4. Edit Distance:
Based on the definition in [2], three types of *edit operations* that transform segment P (denoted as $p_1\ldots p_m$) into segment Q (denoted as $q_1\ldots q_n$) are insertion, deletion and replacement. The edit distance between segments P and Q denoted as $edit(P,Q)$, is the minimum number of edit operations required to transform P into Q.

To determine whether one segment is similar to another segment, a distance threshold (denoted as δ) is needed. Considering the prototypical melody, the difference

between the pivot and a segment S can be compensated by changing a number of notes on the pivot into those on S. Owing to the definition of edit distance, for a given segment, the pivot with a long length has more chances to satisfy the distance threshold than a shorter one even when the numbers of note changes are the same. Therefore, the similarity measure should take the segment length into consideration. Instead of a constant value, we use a variable value depending on the segment length to be the distance threshold.

Definition 2.5. Distance Threshold:
A *distance threshold* for a pivot P is $\delta_P = |P| * \gamma$, where $|P|$ is the segment length of the pivot and γ is the *distance threshold ratio*, $0 \leq \gamma < 1$.

Definition 2.6. Similar Segment:
Given two segments P and Q, satisfying *max_len* and *min_len*, Q is a *similar segment* of P if $edit(P,Q) \leq \delta_P$. In this case, the segment length of Q must be at least $(|P|-\delta_P)$. Note that P is always a similar segment of P since $edit(P,P)=0$.

For example, if the distance threshold ratio is 50% and the segment length of P is 6, the distance threshold for P is 6* 0.5=3. In this case, segment Q is similar to P only if $edit(P,Q)$ is not larger than 3.

When two similar segments overlap to a high degree, they are treated as one segment. We define a measure called *overlapping degree* as follows.

Definition 2.7. Overlapping Degree:
Given two similar segments $S[a:b]$ and $S[c:d]$ where $a \leq c \leq b$, the overlapping degree of them is $(b-c+1)/min(b-a+1,d-c+1)$ if $b<d$. Otherwise it equals 1.

Since the overlapping degree also depends on the segment length, we use a variable value to restrict the maximum overlapping degree among the similar segments.

Definition 2.8. Overlapping Threshold:
An *overlapping threshold* for two similar segments I and J of a pivot is $O_{IJ} = min(|I|,|J|) * \rho$, where $|I|$ and $|J|$ are the segment lengths and ρ is the *overlapping threshold ratio*, $0 \leq \rho \leq 1$.

When ρ is zero, all the similar segments of P should not overlap any others. Another way to estimate the overlapping degree is to ignore the segment length. For instance, in Definition 2.7, the overlapping degree can be simplified to $(b-c+1)$. In this paper, we adopt the measure as Definition 2.7 states.

Definition 2.9. Extension:
Given a pivot P and the set of all its similar segments **S**, an *extension* of P (denoted as *Ext(P)*) is a subset of **S**, where every two segments in it satisfy the overlapping threshold. The number of segments in an extension is called the *support* and denoted as $|Ext(P)|$.

In the application of music classification [6], a constraint on the minimum number of occurrences for a repeating pattern in a music work makes the discovered patterns significant. In this paper, the constraint on the support of an extension is called the *support threshold* (*min_sup*).

Definition 2.10. Approximate Repeating Pattern:
A pivot P is called an *approximate repeating pattern* (abbreviated as ARP) if there exists at least one Ext(P) satisfying the support threshold, i.e., $|\text{Ext}(P)| \geq min\_sup$.

Definition 2.11. Problem of ARP Extraction:
Given a string S and *min_len*, *max_len*, γ, ρ, and *min_sup*, extract all the ARPs in S.

3 Our Approach

3.1 Lower-Bounding Distance

Using the dynamic-programming based approach to compute the edit distance between two strings often costs a lot of time. To reduce it, we define a novel distance measure that can be efficiently computed. The rationale of the proposed measure is as follows. From Definition 2.4, we observed that the order of values in segments has great influence on the edit distance and its computation. Therefore, we ignore the order but count the number of occurrences of each distinct value in a segment instead. The differences between such counts computed from two segments can be combined to approximate the edit distance. Moreover, the distance estimated by our measure is proved to be lower than the edit distance. In this way, we can build a lower-bounding mechanism on the index tree to prune the segments with too large distances. At first, we represent each segment as follows:

Definition 3.1. Histogram Vector:
Let D be a string with $\Sigma_D = \{a_1, a_2, \ldots a_n\}$, S be a segment of D, and h_k^S be the count of a_k in S. The *histogram vector* (abbreviated as *Hvector*) is defined as follows:

$$HV(S) = <h_1^S, h_2^S, \ldots, h_n^S>$$

All the segments are represented as their Hvectors and the Hvectors form a multidimensional space called the *histogram space*, where each dimension refers to a distinct value in the string and the total number of dimensions is $|\sum_D|$. Fig. 2 shows an example, where each bin in the histogram indicates the count of a distinct value in the segment. Note that different segments may be represented as the same Hvector. Moreover, the segments represented by the same Hvector must have the same length. From this property, given a Hvector V_p, we can compute the length of the corresponding segments which is denoted as $|V_p|$.

String: D=(0,1,1,-2,0,1,1,-2,2,1,1,-1)
$\sum_D$ = {-2,-1,0,1,2}
S=(1,1,-2,0,1,1,-2,2,1,1)
$HV(S)$=<2,0,1,6,1>
$|HV(S)|$=10

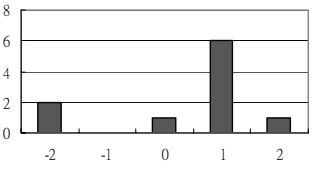

Fig. 2. Representing a segment as the Hvector

Definition 3.2. Histogram Distance:
We define *an insertion* to a dimension in the Hvector as increasing that dimension by one. For two segments S_1 and S_2 of a string D, the minimum number of insertions required to make each dimension in $HV(S_1)$ not smaller than the corresponding one in $HV(S_2)$, is calculated as follows:

$$ins(HV(S_1), HV(S_2)) = \sum_{i=1}^{|\Sigma_D|} d_i, where \quad d_i = \begin{cases} h_i^{S_2} - h_i^{S_1} & if \quad h_i^{S_2} > h_i^{S_1} \\ o & otherwise \end{cases} \quad (1)$$

The distance between the two Hvectors of segments S_1 and S_2, called the *histogram distance* (abbreviated as *Hdistance*) is formulated as follows:

$$HD(S_1,S_2)=max(ins(HV(S_1), HV(S_2)), ins(HV(S_2),HV(S_1))) \quad (2)$$

The Hdistance is guaranteed to be lower than edit distance. The time complexity of edit distance computation is O($m*n$), where m and n denote the two segment lengths. By contrast, the time complexity of Hdistance computation is O($|\sum_D|$), which is independent of the segment lengths. Even if the transformation cost is included, the time complexity is only O($max(|\sum_D|,m,n)$). In general, $m*n$ is larger than $|\sum_D|$. As a result, the Hdistance computation is more efficient than the edit distance computation.

3.2 Indexing Tree

To speed up the retrieval of similar segments for each pivot, we built an R*-tree in the same way as proposed in [1] to index all the Hvectors. Each leaf node in the R*-tree is in the form of (*I*, *p-id*), where *I* denotes a minimal bounding rectangle (MBR) and *p-id* refers to the Hvectors contained in *I*. Moreover, each non-leaf node in the R*-tree is in the form of (*I*, *child-p*), where *child-p* are pointers of all the child nodes and *I* is an MBR that covers all the MBRs of the child nodes. Furthermore, we add entries to each node in the R*-tree such that more nodes can be pruned during the tree traversal for ARP extraction. The modified R*- tree is called the *parametric R*-tree*, where the entries added are as follows:

Definition 3.3. RM Pairs:
A range in string D is denoted as a:b, where a and b are two positions in D and $a < b$. Two segments with ranges a:b and c:d are called non-overlapping if $b < c$ or $d < a$; otherwise, overlapping. A set of overlapping segments can then be represented as (*R*,*M*), called *the RM pair*, where R is the union of all their ranges and M is the minimum of their lengths.

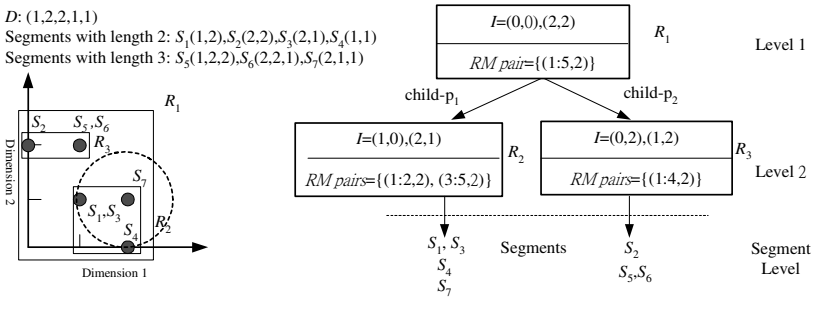

Fig. 3. An example of the histogram space and a parametric R*-tree

For example, S_2 (D[2,3]), S_5 (D[1,3]), and S_6 (D[2,4]) in Fig. 3 are represented as the RM pair (1:4,2). In the parametric R*-tree, for each node, the segments corresponding

to the Hvectors contained by its MBR are distributed into RM pairs such that the overlapping ones fall in the same RM pair. Fig. 3 shows a parametric R* tree with two leaf nodes and only one non-leaf node. $|\sum_D|$ is 2, i.e., the number of dimensions in the histogram space. We construct the parametric R*-tree by sliding windows on D, where *min_len* and *max_len* are set to 2 and 3 respectively. As a result, only two leaf nodes are built to keep all the segments at the bottom level, called the *segment level*. For instance, in node R_2, the RM pair is computed as follows. Since S_7 overlaps S_3 and S_4, they form the RM pair (3:5, 2). On the other hand, S_1 does not overlap any other segment in R_2 and therefore a RM pair (1:2, 2) is generated.

3.3 Extraction Procedure

In this subsection, our approach to ARP extraction on a music work is introduced. In our algorithm, there are three main stages. The first stage constructs the parametric R*-tree as the index tree for subsequent processing. Second, we regard each Hvector in the index tree as a range query and execute them to generate the candidate ARPs. The candidate ARPs are recorded as a linked list named CandidateList, which is put into the last stage. As a result, ARPs satisfying all the constraints are outputted. The last two stages are repeated until the outputted ARPs fulfill user's information need.

3.3.1 Index Construction

An interval string is cut into segments by sliding windows according to the two constraints on segment lengths. After that, each segment is mapped to a Hvector and then inserted into the parametric R*-tree. The mapping is recorded in a *mapping table*. Note that the parametric R*-tree is constructed at the beginning and then updated when new segments are inserted due to a smaller *min_len* or a larger *max_len*.

3.3.2 Candidate Generation

After index construction, we regard each segment in it as a pivot and use its Hvector as a range query on the parameter R*-tree. The segments that are possible to be the similar segments of the pivot are returned and called the *candidate segments*. During the query processing, some pivots that cannot be ARP are pruned. A pivot that survives after query processing is called a *candidate ARP*. For each candidate ARP, we will further check its candidate segments to determine whether it is an ARP or not.

Given a Hvector of pivot p (denoted as V_p), we retrieval its candidate segments from the index tree in four steps:

Step 1: Range Query Formulation

V_p triggers a range query in the form of (V_p,δ_p), where V_p is the center and δ_p is the radius of a sphere in the histogram space.

Step 2: MBR Retrieval

When traversing a level of the index tree, all the MBRs overlapping with (V_p,δ_p) are retrieved and denoted as *overlapping MBR*s. Referring to the histogram space in Fig. 3 as an example, the overlapping MBRs of (<2,1>,1) are R_1 and R_2, which are located at level 1 and level 2, respectively.

Step 3: Estimation for the Maximal Number of Similar Segments
The number of similar segments in an overlapping MBR is estimated in three steps as follows. First, for each RM pair (R_X,M_X) in the MBR, the minimal length of similar segments covered by the range R_X is denoted as ML_X and computed as follows. From Definition 2.6, the length of a similar segment of p must be at least $|p|-\delta_p$. Since M_X records the actual minimal length of segments covered by R_X, we set ML_X to be the maximum of these two values, i.e., $max(|p|-\delta_p, M_X)$.

Second, for each RM pair (R_X,M_X) in the MBR, our goal is to estimate the maximal number of similar segments that can be fitted in R_X, such that any two of them satisfy their overlapping thresholds. This is similar to the following problem.

Refer to the period from a to b on the axis in Fig. 4, we draw a line with the fixed length L starting from position a. Next, we draw a line with the same length starting from the position on the right of position a such that the length of its overlap with the previous line is m. This process is repeating until a line covers position b. The total number of lines drawn in this period is $\lfloor (n - L) / (L - m) \rfloor+1$, where $n=b-a+1$. We denote this number as num_X.

Referring to our goal, the above formula can be used to compute the number of segments with the same length ML_X to be fitted into the range R_X for $m = \rho * ML_X$, which m indicates the overlapping threshold.

At last, all the num_X estimated for RM pairs X in an MBR R are summed up to represent the maximum number of similar segments that can be retrieved from R (denoted as num_R).

We continue using the example in Fig, 3 for illustration. Suppose that the pivot is S_7, ρ is set to 0.5, and the range query (<2,1>,1) is performed on R_2. There are two RM pairs (1:2, 2) and (3:5, 2) in R_2. For the 1st RM pair, minimal length $ML_1 = max(3-1, 2)=2$, $\rho * ML_1=0.5*2=1$ and $num_1 = \lfloor (2-2)/(2-1) \rfloor +1 = 1$. Using the same formula, num_2 of second RM pair is 1. The num_{R2} computed from the example of Fig. 3 is 1+1=2.

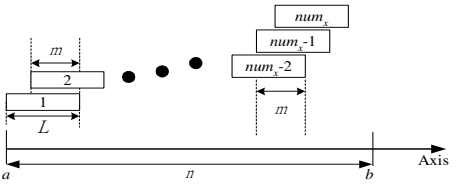

Fig. 4. Maximum number of segments fitted into a range

Step 4: Candidate Pruning Before *HD* Computations
When the range query is processed at the level above the segment level, the num_R of each overlapping MBR R is computed and their sum is denoted as *max_num*. To the pivot corresponding to the Hvector V_p, when *max_num* for the range query (V_p,δ_p) is less than *min_sup*, the computations for the edit distances between it and the other segments are unnecessary. If *max_num* is less than *min_sup*, we terminate the processing of this query and execute the next range query. Otherwise, this query is recursively propagated to the lower levels.

We continue the example in Fig. 3 and assume the *min_sup* is set to 3. Segment S_7 can be pruned because its *max_num* at level 2 is only 2 (according to the Step 3).

Step 5: Candidate Pruning After *HD* Computations
When the range query *Vp* is processed at the segment level, we compute the Hdistances between the pivot *p* and the segments covered by the overlapping MBRs. All the segments whose Hdistances satisfy δ_p are permutated to compute the *max_num* as mentioned in Step 3 and 4. Similarly, if *max_num* is less than *min_sup*, the pivot will be pruned. Otherwise, the segments are regarded as candidate segments. As a result, the CandidateList records the candidate ARP and its candidate segments.

After all the range queries have been performed, we will obtain a set of candidate ARPs and their candidate segments associated in CandidateList, which need to be processed further in the last stage.

3.3.3 ARP Extraction

The output of our approach includes each ARP and its extensions, which can be used to verify whether the ARP is a prototypical melody or not by musicians. Given a candidate ARP and its candidate segments, we first compute the edit distance between the candidate ARP and each of the candidate segments and then remove the candidate segments violating the distance threshold to obtain the set of similar segments.

After that, we generate all the extensions of the candidate ARP by considering the overlapping threshold. Then, if the support of an extension is less than the *min_sup*, the extension is not an answer. As a result, by the Definition 2.10, a candidate ARP is an answer if one of its extensions satisfies the *min_sup* threshold.

3.4 Dimensions Reduction

In a music work, the large number of distinct intervals leads to a high dimensional histogram space. Using the R*-tree to index high dimensional data can be time-consuming. Several methods have been proposed to reduce the dimensions but most of them spend a lot of time on computing the optimal number of dimensions for static data [3]. By the contrast, the parametric R*-tree in our approach is constructed dynamically and the construction time is a part of response time. Therefore, it is not allowable to spend too much time on optimization of dimension reduction. In our approach, we use a simple hashing function to reduce the dimensions of histogram space. In our approach, each interval is divided by a predefined number and the remainder is regarded as the hash value. In this way, different intervals may have the same hash value and their counts in a music work are summed up as a result. The Hdistance after the dimension reduction, denoted as the *Hdistance'*, is still guaranteed to be the lower bound of edit distance. For example, given S_1 = (1,2,3,1,1,2,3,1,3,4,5,3,4,5) and S_2 = (1,3,1,2,3,1,4,5,2,4,4), $HD(S_1,S_2)$ and $edit(S_1,S_2)$ are 4 and 5, respectively. We mod each value by 3 to transform into (1,2,0,1,1,2,0,1,0,1,2,0,1,2) and (1,0,1,2,0,1,1,2,2,1,1), respectively. The number of dimensions is reduced from five to three. Moreover, the Hdistance' between S_1 and S_2 is 3, which is smaller than Hdistance. Because the new lower bound provided by the Hdistance' is looser, more MBRs will be visited during query processing over the parametric R*-tree. Such a trade-off depends on the hashing function and data distribution.

4 Performance Evaluation

4.1 Experiment Set-Up

We compare our approach with a modified version of the dynamic-programming approach named FIExPat [9], which is a famous approach in this field. Four important factors which have great impacts on ARP extraction are investigated, i.e., maximum length, minimum length, distance threshold and support threshold. In Rolland's experiment [9], the segments are not allowed to overlap, for fair comparison, we did not consider the performance comparison on the overlapping threshold and set it to zero in all the experiments. For our approach, the number of reduced dimensions in histogram space is set to 11, which has the best performance in all the experiments. The experiment scenario is set up as follows. The user initially sets the constraints and then the system extracts the ARPs. We name one process of ARP extraction for the user-specified constraints as one *iteration*. In each iteration, one of the constraints is varied such that the influence of that constraint on the elapsed time at different iteration can be observed.

4.2 Experiment Results

Fig. 5(a) shows the result for the various values of *max_len*, where the parameters *min_len*, *min_sup* and γ are set to 4, 5 and 25%, respectively. At the first iteration, both approaches spend more time than the other iterations, which is because ARP has to build a parametric R*-tree and FIExPat has to construct a graph structure. Our approach performs better than FIExPat for all iterations. In addition, the elapsed time of our approach decreases as the *max_len* increases. The reason is because the segment with a larger length gets less chance to form a similar segment of the others and can be pruned by our approach.

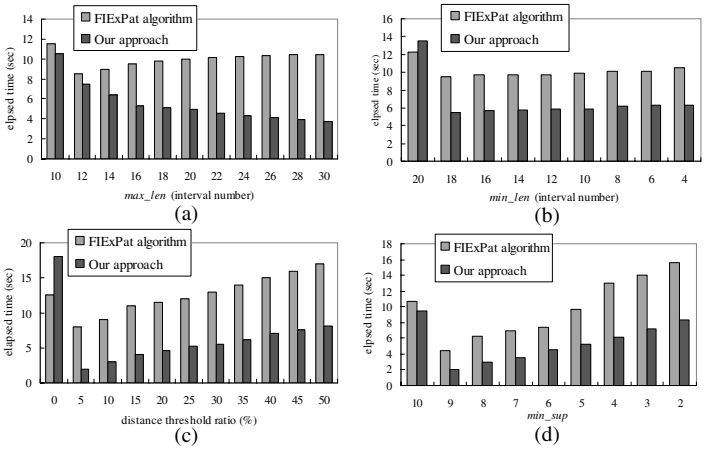

Fig. 5. Experiment results

Fig. 5(b) shows the result for various values of *min_len*, where the parameters *max_len*, *min_sup*, and γ are set to 30, 5 and 25%, respectively. Our approach also performs better than FIExPat except for the first iteration. The elapsed times of both approaches are increased as the *min_len* is decreased, because the number of smaller *min_len* produces more patterns. If we accumulate the elapsed time of first iteration and the one of second iteration for both approaches, our approach costs less than FIExPat. This means our approach is more suitable than FIExPat in the iterative environment.

Fig. 5(c) shows the result for various values of γ, where the parameters *max_len*, *min_len* and *min_sup* are set to 30, 4 and 5, respectively. This setting means that the user releases the distance threshold ratio in order to find more ARPs. Our approach spends more time at the first iteration but less time at the subsequent iterations. The reason for the observation is that our approach builds the parametric R*-tree only at the first iteration, but does not modify the index tree at subsequent iterations since the *max_len* and *min_len* are not changed.

Fig. 5(d) shows the result for various values of *min_sup*, where the parameters *max_len*, *min_len* and γ are set to 30, 4 and 25%, respectively. From the result, our approach also performs better than FIExPat.

5 Conclusion

Since the approximate repeating pattern can be found in both classical and pop music, it plays an important role in the representation of music database and the music style analysis. In this paper, we develop a novel approach to extract the approximate repeating pattern from the music work. This approach adopts the technique of the range query processing on the multidimensional data to reduce the execution time. In the performance study, the execution time of our approach is reduced dramatically when comparing with the FIExPat approach. Our approach not only can be used in the music field, but also can be applied in other fields such as patterns extraction on web click strings or DNA strings.

Some research directions can be considered further. First, improving Hdistance measure such that we can prune more impossible candidates before the computation of edit distance, it can make the ARP extraction more efficient. Second, the dimension reduction sophisticated strategy should be studied to reduce the processing time of range query. Third, the applications base on the approximate repeating patterns will be investigated in the future, e.g., the music classification, the music analysis and the music content-based retrieval.

Acknowledgement

This work was partially supported by the NSC Program for Promoting Academic Excellence of Universities (Phase II) under the grant number 93-2752-E-007-004 -PAE, and the NSC under the contract number 93-2213-E-004-012.

References

[1] Beckmann, N., H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of ACM SIGMOD Int'l. Conf. on Management of Data*, 1990.

[2] Gusfield, D., *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997.

[3] Han J., and M. Kamber, "Data Mining Concepts and Techniques," Morgan Kaufmann Publishers, 2001.

[4] Hsu, J. L., C. C. Liu, and A. L.P. Chen, "Discovering Non-trivial Repeating Patterns in Music Data," *IEEE Transactions on Multimedia*, Vol. 3, No. 3, 2001.

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

[6] Lin, C. R., N. H. Liu, Y. H. Wu and A. L.P. Chen "Music Classification Using Significant Repeating Patterns," in *Proceedings of International Conference on Database Systems for Advanced Applications* (*DASFAA'04*), 2004.

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

[8] Pienimäk, A. "Indexing Music Databases Using Automatic Extraction of Frequent Phrases," in Proceedings of the 3rd International Symposium on Music Information Retrieval (ISMIR'02), 2002.

[9] Rolland, P. Y., "FIExPat: Flexible Extraction of Sequential Patterns," in *Proceedings of the IEEE International Conference on Data Mining* (*ICDM'01*), 2001.

[10] Selfridge-Field, E., "Conceptual and Representational Issues in Melodic Comparison," in Hewlett, W. B. and E. Selfridge-Field (eds.), *Melodic Similarity: Concepts, Procedures, and Applications* (*Computing in Musicology: 11*), The MIT Press, 1998.

[11] Shih, H. H., S. S. Narayanan, and C. C. Jay Kuo, "Automatic Main Melody Extraction From MIDI Files with a Modified Lempel-Ziv Algorithm," in *Proceedings of International Symposium on Intelligent Multimedia, Video and Speech Processing*, 2001.