Music Matching Based on Rough Longest Common Subsequence

HWEI-JEN LIN, HUNG-HSUAN WU AND CHUN-WEI WANG
Department of Computer Science and Information Engineering
Tamkang University
Tamsui, 251 Taiwan

In this paper we proposed a music matching method, called the RLCS (rough longest common subsequence) method. It is an improved version of the LCS to avoid some problems occurring in global alignment matching. First a rough equality for two notes is defined for constructing the RLCS of two music fragments. The length of the RLCS of two music sequences defined in this work is a real number, called a weighted length. It is evaluated according to the degree of similarity of every pair of matched notes from the two sequences. This method takes into account both the width-across-query (WAQ) and the width-across-reference (WAR) and combines them with the weighted length of the corresponding RLCS to define a score measurement for the RLCS. The measurement associated with WAQ and WAR enables the proposed method to tolerate dense errors. A dynamic programming algorithm is presented for simultaneously calculating the weighted length of RLCS, the WAQ, the WAR, and the score to determine the RLCS. As a result, the proposed method can perform the matching in a better and simpler manner. In order to speed up the matching process, we use the filtering algorithm proposed by Tarhio and Ukkonen [22] to filter the reference and discard most of the reference areas that do not match. We applied the proposed algorithm to content-based music retrieval. The experimental results showed that with our proposed algorithm the retrieval system provides a higher retrieval rate than that with the local alignment method proposed by Suyoto et al. [20]. The use of the filtering algorithm has been shown to greatly reduce the computation time for exact matching and for approximate matching with a low error tolerance.

Keywords: content-based music retrieval, rough longest common subsequence (RLCS), local alignment, information retrieval, musical similarity, filtering algorithm

1. INTRODUCTION

The music similarity measurement is important for the automated analysis of melodic fragments. Some factors must be considered when developing algorithms for this measurement, including the feature selection, the music data format, and the tolerance to noise. Since melodic fragments may be transformed or distorted in various ways, music matching must be able to tolerate such variation. Music is made up of musical notes, each of which has many attributes such as pitch, duration, loudness, timbre, and so on. Byrd et al. [5] indicated that pitch alone is not sufficient for searching in a large database. Smith et al. [18] concluded that pitch and pitch duration are the most important attributes, and their findings will be followed in this research. Symbolic music representation and the audio wave format [8] are the most commonly used methods in practice. While symbolic
music represents the musical attributes of music notes including pitch, duration, and loudness, the audio wave format specifies the amplitude of sound over a time series.

For comparing two melodic fragments, techniques based on geometric matching [1, 6, 12, 13] have been explored over the past few years, including point set matching and sweeping-line matching. The advantage of geometric matching is its high accuracy rate of search; while its drawback is its high time complexity, usually exceeding a polynomial of degree 2. Greg Aloupis et al. proposed a novel mechanism using a balanced binary search tree to efficiently measure the distance between two melodies by the minimum area between their corresponding polygonal chains. With the use of BST the time complexity was improved from $O(n^2m)$ to $O(mn \log n)$.

String matching techniques are more efficient than geometric matching and have also been extensively explored in the literature [2, 4, 5, 7, 9, 10, 14, 16, 18], including dynamic time warping (DTW), longest common subsequence (LCS)-based methods, and alignment [10, 20, 21]. The DTW algorithm “warsps” two sequences non-linearly in the time dimension to evaluate the similarity value, and is suitable for audio or speech comparison [11, 23]. The LCS method evaluates the similarity between two sequences by counting the number of matched symbols without considering the local correlation between the matched subsequence (common subsequence) and each of the two sequences. As a result, the LCS method does not characterize the local information such as gaps or mismatched symbols of the common sequence, and it belongs to sort of global alignment [10], which is not suitable for matching a shorter music segment against a much longer one. Most LCS-based techniques are associated with a penalty function to overcome the problem of global alignment, to become so called local alignment matching.

Mongeau et al. [16] used a type of edit distance for music dissimilarity measure, which was calculated by a dynamic programming algorithm. To make their measure transposition invariant, the pitch of each note is encoded as the relative position from the tonic in semitone-units. Lemström et al. [14] presented a framework for sequence comparison based on string matching. The distance between two monophonic musical sequences was expressed by the minimum number of required edit operations to convert the one sequence into the other. Robine et al. [17] investigated some algorithmic improvements to allow edit-based systems to take into account important musical elements: tonality, passing notes, strong and weak beats. Guo et al. [9] proposed the Time-Warped Longest Common Subsequence (T-WLCS) which combines the DTW and the LCS to calculate the similarity scores between two songs, while allowing for variations in speed and inaccuracies in the rhythm between two melodies. They modified the recurrence function for LCS to allow the mapping of one note to many repeated notes, which is similar to the consolidation and fragmentation method proposed by Mongeau et al. [16]. Suyoto et al. [20] represented a piece of music by the sequence of its pitch intervals and the sequence of its duration ratios, and employed the Smith-Waterman local alignment [19] for melodic similarity measure. The similarity values were then stored in a score matrix that allows the music retrieval to be performed rapidly and accurately. However, the Smith-Waterman local alignment has low tolerance to continuous errors in the query that might be introduced by some ornament notes.

In this paper, we adopt the symbolic music representation, where notes are characterized by pitches and duration ratios. To achieve transposition invariance we also tested the use of pitch intervals instead of pitches in the experiments. In order to retain the local
similarity and to achieve approximate matching, we proposed a modified method of LCS, called the rough LCS (RLCS). Herein the Manhattan distance (city block distance) is adopted to measure the distance of two notes. If this distance is smaller than a given threshold, then these two notes are considered to be roughly matched (or roughly equal, or similar). The length of a rough common subsequence is evaluated according to this similarity measure of notes. In addition to the length of the RLCS, we simultaneously evaluate the matrices of the width-across-query (WAQ) and the width-across-reference (WAR), which describes how densely each rough common subsequence is distributed over the query and the reference, respectively. Meanwhile, the corresponding scoring matrix can be constructed according to the length of the RLCS, WAR, and WAQ matrices. Furthermore, we adopt the filtering algorithm proposed by Tarhio and Ukkonen [22] to filter the reference, quickly discarding reference areas that do not match. However, the filtering algorithm benefits only exact matching and approximate matching with small error.

The remainder of this paper is organized as follows. Section 2 provides a detailed description of our proposed method. The use of the filtering algorithm is described in section 3. Section 4 shows the experimental results of the proposed method and compares them with a method of local alignment. Finally, we draw our conclusions and provide suggestions for future work in section 5.

2. LCS AND RLCS

For string-based methods, a melody (music segment) is represented by a string (or sequence) of entries, where each entry specifies the features of one music note. In this section, we present a modified version of the longest common subsequence (LCS) matching method, called the rough longest common subsequence (RLCS) matching method. Although, for simple explanation, the music notes are characterized by pitches and durations in this section, they are characterized by pitch intervals and duration ratios in our proposed system to achieve key invariance and tempo invariance. In section 2.1, we briefly introduce the LCS and define WAR and WAQ for measuring the local similarity between two sequences. We then define the RLCS and describe how to apply it in music matching in section 2.2. In section 2.3, the evaluation of RLCS is described.

2.1 LCS

Let $X = <x_1, x_2, \ldots, x_m>$ be a sequence (or a string) of $m$ characters. Then a subsequence of $X$ must be of the form $X' = <x_{i_1}, x_{i_2}, \ldots, x_{i_k}>$, $1 \leq i_1 < i_2 < \ldots < i_k \leq m$; while a substring of $X$ must be of the form $X'' = <x_j, x_{j+1}, \ldots, x_{j+d}>$, $1 \leq j \leq j + d \leq m$. Therefore, any substring of $X$ must be also a subsequence of $X$.

In the longest-common-subsequence (LCS) problem, we are given two sequences and wish to find a maximum-length common subsequence of them. This problem can be solved efficiently using dynamic programming. LCS solution has been widely adopted for music matching. However, its value alone is not enough to reflect the degree of similarity or matching of two sequences. For the sequence $X = <x_1, x_2, \ldots, x_m>$, we define its subsequence (or substring) as $X[i..j] = <x_i, x_{i+1}, \ldots, x_j>$ and $i$th prefix as $X'_i = <x_1, x_2, \ldots, x_i>$ for $0 \leq i \leq j \leq m$. It is obviously that $X'_i = X[1..i]$. 
Given two sequences \( X = <x_1, x_2, \ldots, x_m> \) and \( Y = <y_1, y_2, \ldots, y_n> \), called reference and query, respectively. The length of the longest-common-subsequence (LCS) of the prefixes \( X_i \) and \( Y_j \) can be evaluated by the following recursive formula:

\[
c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
\max(c[i-1, j-1], c[i-1, j]) & \text{if } i > 0 \text{ and } x_i = y_j, \\
c[i-1, j-1] + 1 & \text{if } i > 0 \text{ and } x_i \neq y_j.
\end{cases}
\]

(1)

\( c[m, n] \) contains the length of an LCS of \( X \) and \( Y \). However this value alone is not enough for reflecting how good the matching of two sequences is. As shown in Fig. 1, the matches of the query \( Y \) against the references \( X \) and \( X' \), of length 4. Although the LCSs produced by the two matches have the same length (and even the same content in this example), this value alone is not fair enough for evaluating and comparing the matches. One should take into consideration how densely the LCS is distributed over both sequences \( X \) and \( Y \) as well. To do this we evaluate the widths of the shortest substrings of the reference and query containing the LCS, called the width-across-reference (WAR)/width-across-query (WAQ), respectively. In Fig. 1 (a), the substrings \( X[1..4] \) and \( Y[1..4] \) are the shortest substrings of \( X \) and \( Y \), respectively, containing the LCS, \( <M, A, T, H> \), and thus the lengths of these substrings are \( \text{WAR}(X, Y) = 4 \) and \( \text{WAQ}(X, Y) = 4 \). Similarly, for the match shown in Fig. 1 (b), \( \text{WAR}(X', Y) = 9 \) and \( \text{WAQ}(X', Y) = 4 \). A smaller value of WAR/WAQ indicates a denser distribution of the LCS over the reference/query, and small values of WAR and WAQ together indicate a good match. Since the values of WAR and WAQ for the first match shown in Fig. 1 (a) are smaller than those for the second match in Fig. 1 (b), respectively, the first match is considered better than the second one.

(a) The LCS of \( X \) and \( Y \) is \( <M, A, T, H> \), the shortest substrings of \( X \) and \( Y \) containing the LCS are \( X[1..4] = <M, A, T, H> \) and \( Y[1..4] = <M, A, T, H> \), respectively.

(b) The LCS of \( X' \) and \( Y \) is \( <M, A, T, H> \), the shortest substrings of \( X \) and \( Y \) containing the LCS are \( X'[1..9] = <M, A, A, S, T, R, I, C, H> \) and \( Y[1..4] = <M, A, T, H> \), respectively.

![Fig. 1. Two matches.](image)

The values of WAR and WAQ can be evaluated by the recurrences given in Eqs. (2) and (3), respectively, where \( w^R[i, j] = \text{WAR}(X, Y) \) and \( w^Q[i, j] = \text{WAR}(X', Y) \).

\[
w^R[i, j] = \begin{cases} 
0 & \text{if } i = 0, \\
w^R[i-1, j-1] + 1 & \text{if } i > 0 \text{ and } x_i = y_j, \\
w^R[i-1, j-1] + 1 & \text{if } i > 0 \text{ and } x_i \neq y_j, c[i-1, j] = c[i, j-1], w^R[i-1, j] > 0, \\
0 & \text{if } i > 0 \text{ and } x_i \neq y_j, c[i-1, j] = c[i, j-1], w^R[i-1, j] = 0,
\end{cases}
\]

(2)

\[
w^Q[i, j] = \begin{cases} 
0 & \text{if } i = 0, \\
w^Q[i-1, j-1] + 1 & \text{if } i > 0 \text{ and } x_i = y_j, \\
w^Q[i-1, j-1] & \text{if } i > 0 \text{ and } x_i \neq y_j, c[i-1, j] \geq c[i, j-1], w^Q[i-1, j] > 0,
\end{cases}
\]

(3)
MUSIC MATCHING BASED ON RLCS

With the three tables $c$, $w^R$ and $w^Q$, we may fairly search for the best match of the query prefix $Y_j$ against the reference prefix $X_i$ as follows. The terms $c[i, j]/w^R[i, j]$ and $c[i, j]/w^Q[i, j]$ represent how densely the LCS of $X_i$ and $Y_j$ is distributed over these two prefix sequences and can be used to fairly evaluate the match. In this work we define the score of the match as the product of the weighted average of these two measures and the ratio of the length of the LCS to that of the query, as given in Eq. (4), where $\beta$ is a weighting parameter and $\rho$ is a required matched rate on the entire query sequence. It is obvious that $0 \leq s[i, j] \leq 1$, and it indicates an exact match when $s[i, j] = 1$.

$$s[i, j] = \begin{cases} 
\beta c[i, j]/w^R[i, j] + (1 - \beta)c[i, j]/w^Q[i, j] \cdot (c[i, j]/n) & \text{if } c[i, j] \geq \rho n \\
0 & \text{otherwise}
\end{cases}$$  \hspace{1cm} (4)

In the example of music sequence matching given in Fig. 2, for the query $Y$, $X$ is a relevant reference but $X'$ is not. The lengths of LCSs of the pair $(X, Y)$ and the pair $(X', Y)$ are both 5, and thus the similarity measure for $Y$ and $X$ would be equal to that for $Y$ and $X'$ while using LCS. The score defined in Eq. (4) can be used to provide a fairer similarity measure as follows: Since $WAR(X, Y) = 5$, $WAQ(X, Y) = 5$, $WAR(X', Y) = 15$, $WAQ(X', Y) = 5$, and $LCS(X, Y) = LCS(X', Y) = 5$, with the setting $\beta = 0.5$ and $\rho = 0.8$, we have the scores for the matches of $Y$ against $X$ and $X'$, $(0.5)(5/5) + (0.5)(5/5) = 1$ and $(0.5)(5/15) + (0.5)(5/5) \approx 0.667$, respectively. As a result, the use of the score function defined in Eq. (4) can fairly distinguish the relevant reference from the database.

Fig. 2. Query $Y$ matched against two references $X$ and $X'$. (a) The length of the LCS, WAQ and WAR of $Y$ and $X$ are all 5. (b) The length of the LCS, WAQ and WAR of $Y$ and $X'$ are 5, 5, and 15.
2.2 RLCS for Music Matching

Given two music fragments, a reference \( R = \langle r_1, r_2, \ldots, r_m \rangle \) and a query \( Q = \langle q_1, q_2, \ldots, q_n \rangle \), where \( r_i = (p_i, d_i) \), \( i = 1, \ldots, m \), \( q_j = (p_j, d_j) \), \( j = 1, \ldots, n \). \( p_i \) and \( p_j \) denote the pitch value of the \( i \)th and the \( j \)th note in the reference and the query; \( d_i \) and \( d_j \) represent the duration value of \( i \)th and \( j \)th note in the reference and the query, respectively. Usually, the query \( Q \) is shorter than the reference \( R \); that is, \( m > n \).

To allow slight variance a rough match between two symbols is preferred. In this paper, we define the LCS based on the rough match of symbols for two given music sequences, called the rough-longest-common-subsequence (RLCS), and denote the rough match of two notes from the query \( Q \) and the reference \( R \) as \( r_i \approx q_j \), said to be “roughly equal” if their distance \( d(r_i, q_j) \leq T_d \). If we set the threshold \( T_d = 0 \), then the match becomes exact. The definition of distance between two notes is given in Eq. (5), where \( \alpha \) is a weight value.

\[
d(r_i, q_j) = \alpha |p_i - p_j| + (1 - \alpha) |d_i - d_j| \quad (5)
\]

Upon the definition of rough equality for two notes, we define and evaluate the length of the rough longest common subsequence (RLCS), width-across-reference (WAR), and width-across-query (WAQ) of sequences \( R \) and \( Q \) by the recursive formulae given in Eqs. (6)-(8). Note that, \( c[i, j] \) representing the “length” of the RLCS of \( X_i \) and \( Y_j \) is a real number. For each match of a pair of notes, the length of the RLCS is increased by \((1 - d(r_i, q_j)/T_d)\) rather than 1 as LCS usually does. The increment \((1 - d(r_i, q_j)/T_d)\) is between 0 and 1 since \( d(r_i, q_j) \leq T_d \). The smaller the distance is the greater the increment is. We called this length “weighted length”. The weighted length is always not greater than the true length.

\[
c[i, j] = \begin{cases} 0 & i \cdot j = 0, \\
c[i-1, j-1] + (1 - d(r_i, q_j)/T_d) & i \cdot j > 0 \text{ and } r_i \approx q_j, \\
\max(c[i-1, j-1], c[i-1, j]) & i \cdot j > 0 \text{ and } r_i \neq q_j,
\end{cases} \quad (6)
\]

\[
w^R[i, j] = \begin{cases} 0 & i \cdot j = 0, \\
w^R[i-1, j-1] + 1 & i \cdot j > 0, r_i \approx q_j, \\
\min(w^R[i-1, j-1], c[i-1, j], c[i, j-1], w^R[i-1, j]) & i \cdot j > 0, r_i \neq q_j
\end{cases} \quad (7)
\]

\[
w^Q[i, j] = \begin{cases} 0 & i \cdot j = 0, \\
w^Q[i-1, j-1] + 1 & i \cdot j > 0, r_i \approx q_j, \\
\min(w^Q[i-1, j-1], c[i-1, j], c[i, j-1], w^Q[i-1, j]) & i \cdot j > 0, r_i \neq q_j
\end{cases} \quad (8)
\]
2.3 Evaluation of RLCS

Let \( Z = <z_1, z_2, ..., z_n> \) be the LCS of \( R \) and \( Q \), and assume it corresponds to the sub-sequences \(<r_1, r_2, ..., r_m>\) and \(<q_1, q_2, ..., q_n>\) of \( R \) and \( Q \), respectively, obtained in the matching process. That is, \( <z_1, z_2, ..., z_n> = <r_1, r_2, ..., r_m> = <q_1, q_2, ..., q_n> \). Note that \( k = c[i,j], w^c[i,j] = i - i + 1, \) and \( w^q[i,j] = j - j + 1 \). The terms \( c[i,j]/w^c[i,j] \) and \( c[i,j]/w^q[i,j] \) denote the matched rates for the range across the reference prefix \( R \) and the range across the query prefix \( Q \), respectively, which can be used to evaluate a match. For this purpose, we define the matching score \( s \), as the product of the weighted matched rate and the weighted length of the corresponding common sequence in Eq. (9), where \( \beta \) is a weighting parameter and \( \rho \) is a required matched rate on the entire query sequence.

The tables \( c, w^c, w^q \), and \( s \), can be computed simultaneously using dynamic programming by Algorithm RLCS given below. For later tracking the path of entries corresponding to an RLCS, another table \( b \) is maintained. Intuitively, \( b[i,j] \) points to the table entry corresponding to the optimal subproblem solution chosen when computing \( c[i,j] \).

Algorithm RLCS \((R = <r_1, r_2, ..., r_m>, Q = <q_1, q_2, ..., q_n>)\)

1. for \( i \leftarrow 1 \) to \( m \);
2. \( \text{do } c[i,0] \leftarrow 0; \) \( w^c[i,0] \leftarrow 0; \) \( w^q[i,0] \leftarrow 0 \);
3. for \( j \leftarrow 0 \) to \( n \);
4. \( \text{do } c[0,j] \leftarrow 0; \) \( w^c[0,j] \leftarrow 0; \) \( w^q[0,j] \leftarrow 0 \);
5. for \( i \leftarrow 1 \) to \( m \);
6. \( \text{do for } j \leftarrow 1 \) to \( n \);
7. \( \text{do if } r_i \approx q_j; \)
8. \( \text{then } c[i,j] \leftarrow c[i-1,j-1] + (1 - d(r_i, q_j)/T_d) \)
9. \( b[i,j] \leftarrow "\downarrow"; \)
10. \( w^c[i,j] \leftarrow w^c[i-1,j-1] + 1; \)
11. \( w^q[i,j] \leftarrow w^q[i-1,j-1] + 1; \)
12. \( \text{else if } c[i-1,j] \geq c[i,j-1] \)
13. \( \text{then } c[i,j] \leftarrow c[i-1,j]; \)
14. \( b[i,j] \leftarrow "\uparrow"; \)
15. \( w^c[i,j] \leftarrow w^c[i-1,j]; \)
16. \( \text{if } w^q[i-1,j] > 0 \)
17. \( \text{then } w^q[i,j] \leftarrow w^q[i-1,j] + 1 \)
18. \( \text{else } w^q[i,j] \leftarrow 0 \)
19. \( \text{else } c[i,j] \leftarrow c[i,j-1]; \)
20. \( b[i,j] \leftarrow "\leftarrow"; \)
21. \( w^c[i,j] \leftarrow w^c[i,j-1]; \)
if \( w_r^Q[i,j-1] > 0 \)
then \( w_r^Q[i,j] \leftarrow w_r^Q[i,j-1] + 1 \)
else \( w_r^Q[i,j] \leftarrow 0; \)
if \( c_r[i,j] \geq \rho \cdot n \)
then \( s_r[i,j] \leftarrow \beta c_r[i,j]^2/nw_r^R[i,j] + (1 - \beta)c(i,j)^2/nw_r^Q[i,j] \)
else \( s_r[i,j] \leftarrow 0; \)
\( \text{score} \leftarrow \max_{i,j} s_r[i,j]; \)
return \( \text{score} \)

With the score table \( s_r \), one may request the system to output all the matches with high scores, or all the matches with scores higher than a given threshold \( \rho \). As shown in Fig. 3 (e), all the entries in the table \( s_r \) greater than \( \rho = 0.8 \) are highlighted. Alternatively, one may request the system to simply output the match with the highest score without providing any threshold.

![Fig. 3. A matching example: \( R = <M, A, T, H, E, M, A, T, I, C> \) and \( Q = <M, A, T, H> \).](image)

### 3. Filtering Algorithm

Although the proposed RLCS matching algorithm has been shown to be effective, it requires a large amount of computational time. This is due to the fact that in the matching process, the query is being matched against the reference by moving along the generally quite long reference, one step at a time. To speed up this process, we adopted the filtering algorithm proposed by Tarhio and Ukkonen [22] to rapidly filter the reference and discarding most of the reference areas that do not match. As a result, the computational time can be reduced. Our main interest in filtering algorithms is their potential of not needing to inspect all notes in the reference. The idea is to align the query with a reference window and scan the reference backwards. The scanning ends when more than \( k \) bad reference notes are found. A bad note is one that not only does not match the query position but also does not match any query note at a distance of \( k \) notes or less. Let’s assume that the window is placed such that \( r_j \) is aligned with \( q_i \). Then \( r_j \) is bad for \( i \) when \( r_j \not\in \{ q_{i-k}, q_{i-k+1}, \ldots, q_{i+k} \} \). The badness \( \text{Badchar}(i, c) \) can be defined in Eq. (10).

<table>
<thead>
<tr>
<th>( c_r[m,n] )</th>
<th>( h[m,n] )</th>
<th>( w_r^R[m,n] )</th>
<th>( w_r^Q[m,n] )</th>
<th>( s_r[m,n] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 1 1 1 1</td>
<td>M * * * *</td>
<td>M 1 1 1 1</td>
<td>M 1 2 3 4</td>
<td>M 0 0 0 0</td>
</tr>
<tr>
<td>A 1 2 3 4</td>
<td>A * * * *</td>
<td>A 2 2 2 2</td>
<td>A 1 2 3 4</td>
<td>A 0 0 0 0</td>
</tr>
<tr>
<td>T 1 2 3 4</td>
<td>T * * * *</td>
<td>T 1 2 3 4</td>
<td>T 1 2 3 4</td>
<td>T 0 0 0 0</td>
</tr>
<tr>
<td>H 1 2 3 4</td>
<td>H * * * *</td>
<td>H 4 4 4 4</td>
<td>H 1 2 3 4</td>
<td>H 0 0 0 1.00</td>
</tr>
<tr>
<td>E 1 2 3 4</td>
<td>E * * * *</td>
<td>E 3 3 3 3</td>
<td>E 2 3 4</td>
<td>E 0 0 0 0.90</td>
</tr>
<tr>
<td>A 1 2 3 4</td>
<td>A * * * *</td>
<td>A 2 2 2 7</td>
<td>A 1 2 3 4</td>
<td>A 0 0 0 0.79</td>
</tr>
<tr>
<td>T 1 2 3 4</td>
<td>T * * * *</td>
<td>T 3 3 3 8</td>
<td>T 1 2 3 4</td>
<td>T 0 0 0 0.75</td>
</tr>
<tr>
<td>I 1 2 3 4</td>
<td>I * * * *</td>
<td>I 4 4 4 9</td>
<td>I 1 2 3 4</td>
<td>I 0 0 0 0.72</td>
</tr>
<tr>
<td>C 1 2 3 4</td>
<td>C * * * *</td>
<td>C 5 5 5 10</td>
<td>C 1 2 3 4</td>
<td>C 0 0 0 0.70</td>
</tr>
</tbody>
</table>

(a) (b) (c) (d) (e)
**Badchar**<sub>k</sub>(i, c) = \[
\begin{cases}
\text{true} & \text{if } c \not\in \{q_{i-k}, q_{i-k+1}, \ldots, q_i, \ldots, q_{i+k}\} \\
\text{false} & \text{otherwise}
\end{cases}
\]

for \( c \in \Sigma, k + 1 \leq i \leq n - k \) (10)

When more than \( k \) bad notes are found, the current reference window is abandoned and shifted forwards; otherwise, the area must be checked with the RLCS algorithm. The shift of the window is computed as \( \min_{n-k \leq i \leq n} \text{Shift}_k(i, r_j) \), where \( \text{Shift}_k(i, r_j) \) is defined in Eq. (11).

\[
\text{Shift}_k(i, c) = \begin{cases} 
\min_{s > 0} \{q_{i-s} = c\} & \text{if } s \text{ exists} \\
\frac{n}{n} & \text{otherwise}
\end{cases}
\]

for \( c \in \Sigma, n - k \leq i \leq n \) (11)

Both the \( \text{Badchar}_k \) and the \( \text{Shift}_k \) tables can be precomputed. Some of the potential reference areas that have to be checked with the RLCS algorithm might be overlapped, and so are then the matching processes. To address this problem, instead of individually checking each potential reference area with the RLCS algorithm, we check the union of all the potential reference areas with the RLCS algorithm. The RLCS matching algorithm combined with the filtering algorithm, called FRLCS, is summarized below. The preprocessing of the filtering algorithm moves the query, which is generally much smaller than the reference, along the reference multiple steps at a time. It searches for possible fragments of the reference that are about the same length as the query to match against. As a result the computation time for matching is greatly reduced. If we let \( \tau \) denote the error tolerance rate, then \( k = \tau n \) (\( n \) is the size of the query).

To better understand the filtering portion of the algorithm, let’s consider the example illustrated in Fig. 4, in which \( m = 500, n = 15, \tau = 2/15 \), and so \( k = 2 \), and the sequence of moves is as listed in the first row of the table. This allows us to determine the sequence of positions pointer \( j \) travels (starting from \( j = n = 15 \)) as listed in the second row of the table. For example, since the first move is 3, pointer \( j \) moves 4 steps from the initial point to the second position \( 15 + 3 = 18 \). The third row of the table shows if the reference window pointed by pointer \( j \) is bad; that is, it is bad if \( \text{bad} \geq k \), otherwise it is not bad. As shown in the fourth row of the table, the 6 reference windows with pointer values \( j = 18, 23, 245, 249, 252, \) and 376, are potential, the first two that overlap, and the next three that overlap, respectively. Merging the overlapping areas yields three \( (\lambda = 3) \) extended areas \( [\text{Lt}(1), \text{Rt}(1)] = [11, 27], [\text{Lt}(2), \text{Rt}(2)] = [234, 254], \) and \( [\text{Lt}(3), \text{Rt}(3)] = [365, 378] \). The concatenation of these extended reference areas, called a filtered reference, is \( R' = [\text{Lt}(1), \text{Rt}(1), \text{Lt}(2), \text{Rt}(2), \text{Lt}(3), \text{Rt}(3)] = R[11 \ldots 27]R[234 \ldots 254]R[365 \ldots 378] \), which has a length \( m' = |R'| = 52 \). Therefore, the query only needs to be matched against a smaller portion of the reference, namely \( R' \), and the time cost is reduced from \( O(mn) \) to \( O(m'n) \). However, the use of the filtering algorithm can reduce the computation time for exact matching and for

<table>
<thead>
<tr>
<th>move</th>
<th>*</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>...</th>
<th>12</th>
<th>4</th>
<th>3</th>
<th>...</th>
<th>10</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j )</td>
<td>15</td>
<td>18</td>
<td>23</td>
<td>32</td>
<td>...</td>
<td>245</td>
<td>249</td>
<td>252</td>
<td>...</td>
<td>376</td>
<td>...</td>
</tr>
<tr>
<td>( \text{bad} \geq k )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>area</td>
<td>*</td>
<td>[7, 20]</td>
<td>[12, 25]</td>
<td>*</td>
<td>*</td>
<td>[234, 247]</td>
<td>[238, 251]</td>
<td>[241, 254]</td>
<td>*</td>
<td>[365, 378]</td>
<td>...</td>
</tr>
<tr>
<td>ext. area</td>
<td>*</td>
<td>[11, 27]</td>
<td>*</td>
<td>*</td>
<td>[234, 254]</td>
<td>*</td>
<td>[365, 378]</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( R' )</td>
<td>[11, 27]</td>
<td>[234, 254]</td>
<td>[365, 378]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4. An example of filtering: \( m = 500, n = 15, \tau = 2/15, k = 2, \lambda = 3, m' = 50 \).
approximate matching with a low error tolerance, but not for approximate matching with a high error tolerance for which the reference won’t be filtered to any extent. To maintain the retrieval rate and the ranking, the error tolerance rate must be greater than the noise rate of the query; i.e., the value of $k$ must be at least the noise rate of the query set times the size of the query.

Algorithm FRLCS ($Q = \langle q_1, q_2, \ldots, q_n \rangle$, $R = \langle r_1, r_2, \ldots, r_m \rangle$)

1. $j \leftarrow n$; exact $\leftarrow$ false; $\lambda \leftarrow 0$; end $\leftarrow$ true;
2. while (not exact) and ($j \leq m + k$) do
3.   bad $\leftarrow 0$; $i \leftarrow n$; $h \leftarrow j$; step $\leftarrow n$;
4.   while bad $\leq k$ and $i > k$ do // Check if the reference window is bad or potential
5.     if badchar($i, r_h$) then
6.       bad $\leftarrow$ bad + 1;
7.     if $i \geq n - k$ then step $\leftarrow$ min(step, shiftk($i, r_h$));
8.     $i \leftarrow i - 1$; $h \leftarrow h - 1$;
9.   if bad $\leq k$ then if end
10.      $\lambda \leftarrow \lambda + 1$; end $\leftarrow$ false; // Record the potential area
11.     if $j - n + 1 - k < 1$
12.        then $Lt(\lambda) \leftarrow 1$ else $Lt(\lambda) \leftarrow j - n + 1 - k$;
13.     if $j + k > m$
14.        then $Rt(\lambda) \leftarrow m$ else $Rt(\lambda) \leftarrow j + k$
15.     else if $j - n - k \geq Rt(\lambda)$
16.        then if $j + k > m$ // Extend the area
17.           then $Rt(\lambda) \leftarrow m$ else $Rt(\lambda) \leftarrow j + k$
18.        else end $\leftarrow$ true; $\lambda \leftarrow \lambda + 1$;
19.     if $j - n + 1 - k < 1$
20.        then $Lt(\lambda) \leftarrow 1$ else $Lt(\lambda) \leftarrow j - n + 1 - k$;
21.     if $j + k > m$
22.        then $Rt(\lambda) \leftarrow m$ else $Rt(\lambda) \leftarrow j + k$;
23.     end $\leftarrow$ false;
24. move $\leftarrow$ max($k + 1$, step); $j \leftarrow j + move$;
25. $R' \leftarrow$ Con $\{Lt(r) . Rt(r)\}$
26. score $\leftarrow$ RLCS($Q, R'$)

(a) It is found that the best value for $\alpha$ exists in the range $[0.7, 0.9]$ when testing with a set of 807 references over interval $[0.5, 1]$.
(b) The best setting for $\alpha$ is 0.76 when testing on the whole corpus of 3497 references over interval $[0.72, 0.88]$.

Fig. 5. The average retrieval ranks for query set C over different settings for $\alpha$. 

4. EXPERIMENTAL RESULTS

In this work, each music fragment is considered a sequence of musical notes, whose pitches and duration ratios form a pitch sequence and a duration ratio sequence, respectively. As stated by Moles [15] and Suyoto et al. [20], the duration ratio is less important than the pitch for matching, thus the duration ratio \(d[i]\) for note \(i\) is quantized into \(d'[i]\), an integer between 0 and 4, as shown in Eq. (12).

\[
d'[i] = \begin{cases} 
0 & \log_2 d[i] < -2 \\
1 & -2 \leq \log_2 d[i] < -1 \\
2 & -1 \leq \log_2 d[i] < 1 \\
3 & 1 \leq \log_2 d[i] < 2 \\
4 & 2 \leq \log_2 d[i] 
\end{cases}
\]  

(12)

We used a corpus of 3497 pieces of music obtained from the internet, consisting of a total of 6,524,476 music notes. A subset of 50 pieces of music were randomly selected, from each of which a theme, as suggested by Barlow et al. [3], was extracted to form a query consisting of 10 to 36 music notes. As a result, a set of 50 queries were introduced, called query set \(A\). We varied query set \(A\) to form two query sets \(B\) and \(C\) with different ranges of noise rates by randomly changing the pitch of some notes within an octave, deleting some notes, or inserting some notes in each query. Query set \(B\) has noise rates between 5% and 10%, query set \(C\) has noise rates between 10% and 30%. In addition, a query set of the same size, named \(D\), was generated by the users. The performance will be evaluated by retrieval rank and retrieval rate. The retrieval rank is the rank of the correct reference melody in the list of the melodies based on the magnitude of matching scores. The top-\(n\) retrieval rate is defined for a set of queries as the percentage of queries appearing at the top-\(n\) retrieval lists.

The threshold \(T_d\) is set to 1.0, the parameter \(p\) is set to 0.7 since the noise rate for each query set is under 30%, the parameter \(\beta\) is set to 0.5 since the WAR and WAQ are considered to be equally important. To determine the best value for the parameter \(\alpha\) in the interval \([0.5, 1]\) as stated above, we performed test in two stages since testing with a large set of references over a wide range of \(\alpha\) takes time. First, the testing for \(\alpha\) over the interval \([0.5, 1]\) was done with a smaller set of 807 references randomly selected from the corpus of 3497 references, and the value \(\alpha = 0.8\) was found to provide the best average rank, as shown in Fig. 5 (a). Next, the testing was done with the whole corpus of 3497 references over the small range \([0.7 + 0.02, 0.9 - 0.02] = [0.72, 0.88]\) of \(\alpha\) and the best setting \(\alpha = 0.76\) was found, as shown in Fig. 5 (b).

4.1 Comparison for Ranking and Retrieval Rates

We compared the retrieval rate for our proposed matching method \(RLCS\) with that for the method proposed by Suyoto et al. [20], called \(Local\-\text{Alignment}\). Both algorithms were used to calculate the similarity between two music fragments to perform content-based music retrieval. The retrieval rates based on top-\(n\) lists using each of the two algorithms over two query sets are illustrated in Figs. 6 (a) and (b). The \(RLCS\) and the \(Local\-
Fig. 6. Retrieval rates on top-$n$ lists using RLCS and Local-Alignment based on pitch and duration ratio and tested on (a) query set $C$ and (b) query set $D$.

Table 1. Average retrieval rank (based on pitch and duration ratio).

<table>
<thead>
<tr>
<th>query set</th>
<th>algorithm</th>
<th>RLCS</th>
<th>Local-Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td></td>
<td>3.28</td>
<td>53.36</td>
</tr>
<tr>
<td>$D$</td>
<td></td>
<td>1.40</td>
<td>31.00</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>2.34</td>
<td>42.18</td>
</tr>
</tbody>
</table>

Fig. 7. Retrieval rates on top-$n$ lists using RLCS and Local-Alignment based on pitch interval and duration ratio and tested on (a) query set $C$ and (b) query set $D$.

Table 2. Average retrieval rank (based on pitch interval and duration ratio).

<table>
<thead>
<tr>
<th>query set</th>
<th>algorithm</th>
<th>RLCS</th>
<th>Local-Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td></td>
<td>16.14</td>
<td>117.36</td>
</tr>
<tr>
<td>$D$</td>
<td></td>
<td>2.16</td>
<td>54.74</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>9.15</td>
<td>86.05</td>
</tr>
</tbody>
</table>

Alignment achieve retrieval rates of 100% and 94%, respectively, for top-128 retrieval testing on query set $C$; and achieve 100% and 94%, respectively, for top-16 retrieval testing on query set $D$. 
The average retrieval ranks of the two matching algorithms over two query sets are shown in Table 1. When testing on query set $C$, the average ranks yielded by the RLCS and the Local-Alignment are 3.28 and 53.36, respectively. The performance of the former is better than the latter. Testing on query set $D$, the average ranks for the RLCS and the Local-Alignment are 1.40 and 31.00, respectively.

For invariance to transposition, we replace the feature of pitch with pitch interval and compare the performance of RLCS and the Local-Alignment using the new feature. As demonstrated in Fig. 7, the RLCS is superior over Local-Alignment tested on both query sets. Table 2 shows that the average ranks yielded by the RLCS and the Local-Alignment are 9.15 and 86.05, respectively.

### 4.2 Time Efficiency Improvement with Filtering Algorithm

As discussed in the previous section, the time complexity of both algorithms are $O(mn)$ with different leading constants, where $m$ and $n$ are the length of the reference and the query, respectively. The experimental results show that the RLCS requires more retrieval time than the Local-Alignment since the former needs to evaluate more tables than the latter. The average retrieval times (based on pitch interval and duration ratio) for RLCS and Local-Alignment for the query set $C$ are 14.893 and 9.324 seconds, respectively, and for the query set $D$ are 14.762 and 9.198 seconds, respectively. A filtering algorithm can be used to reduce the matching time. As shown in Table 3, testing on query set $A$ without noise, the use of the filtering algorithm with $k = 0$ reduces the retrieval time from 14.315 seconds to 3.868 seconds, or a time saving of about 73%, while maintaining the same effectiveness (ranking). When testing on query set $B$ with a noise rate of between 5% and 10%, the use of a filtering algorithm with $k = 0.1n$ and $0.05n$ reduces the retrieval time from 14.516 seconds to 3.935 and 3.874 seconds, respectively. With $k = 0.1n$ the average rank is maintained; but with $k = 0.05n$ the average rank drops down to 560.0 since some of the queries have a noise rate greater than the error tolerance rate of 5% and their matches were abandoned in the filtering process. When testing on query set $C$ with a noise rate between 10% and 30%, the use of a filtering algorithm with $k = 0.3n$, $0.2n$, and $0.1n$ reduces the retrieval time from 14.893 seconds to 13.157, 7.591, and 4.295, respectively; but with $k = 0.4n$ the retrieval time increases to 16.845 seconds. This is because with a high error tolerance very little to nothing in the reference is filtered, and the extra time taken by the filtering simply increases the time required. As a result, the filtering is of no benefit for approximate matching with an error tolerance rate greater than about 20%.

### Table 3. Performance with or without filtering.

<table>
<thead>
<tr>
<th>Query set</th>
<th>Noise rate</th>
<th>$k$</th>
<th>RLCS with Filtering</th>
<th>RLCS without Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Retriv time</td>
<td>rank</td>
</tr>
<tr>
<td>$A$</td>
<td>0%</td>
<td>0</td>
<td>3.868</td>
<td>1.0</td>
</tr>
<tr>
<td>$B$</td>
<td>5%-10%</td>
<td>0.1n</td>
<td>3.935</td>
<td>3.02</td>
</tr>
<tr>
<td>$B$</td>
<td>5%-10%</td>
<td>0.05n</td>
<td>3.874</td>
<td>560.0*</td>
</tr>
<tr>
<td>$C$</td>
<td>10%-30%</td>
<td>0.4n</td>
<td>16.845</td>
<td>16.14</td>
</tr>
<tr>
<td>$C$</td>
<td>10%-30%</td>
<td>0.3n</td>
<td>13.157</td>
<td>102.52</td>
</tr>
<tr>
<td>$C$</td>
<td>10%-30%</td>
<td>0.2n</td>
<td>7.591</td>
<td>322.12</td>
</tr>
<tr>
<td>$C$</td>
<td>10%-30%</td>
<td>0.1n</td>
<td>4.295</td>
<td>2697.88</td>
</tr>
</tbody>
</table>
4. CONCLUSIONS AND FUTURE WORK

In this work, we presented a music similarity measurement, which calculates the similarity score between a query and a reference using the length of their RLCS, the WAR and the WAQ, to address the problem introduced by local alignment. This modified version of the LCS algorithm has two benefits: (1) The definition of rough equality for two notes provides a fairer approximation match than the conventional LCS, and (2) The use of width-across-query and width-across-reference for the score measurement reflects the local comparison. The use of pitch interval and duration ratio makes the proposed method RLCS invariant to both transposition and tempo, but causes the anti-symmetry effect [14] while altering the pitch or the duration of a single note in a musical sequence. That is, altering the pitch/duration of a single note would alter two consecutive pitch intervals and two consecutive duration ratios and thus the difference for both features would be counted twice. Solving this problem is part of the focus of our current work. We improved the time efficiency of the RLCS by a filtering algorithm. However, the use of the filtering algorithm only substantially reduce the computation time for exact matching and for approximate matching with a low error tolerance, but not for approximate matching with a high error tolerance for which the reference won’t be filtered to any extent.

We have applied our proposed method to content-based music retrieval tasks and are currently investigating the applicability of this method to main melody extraction. For main melody extraction the given melody is matched against itself to extract the approximate repetitions of the main melodies.

REFERENCES

Hwei-Jen Lin (林慧珍) received the B.S. degree in Applied Mathematics from National Chiao Tung University, Hsinchu, Taiwan and the M.S. and the Ph.D. degrees in Mathematics from Northeastern University, Boston, U.S.A. She worked for Northeastern University, Boston and Rhode College, Providence, Rhode Island in U.S.A., as a lecturer and an Assistant Professor, respectively. She is currently an Associate Professor at the Department of Computer Science and Information Engineering of Tamkang University, Taipei, Taiwan, R.O.C. Her current research interests include pattern recognition, image processing, and intelligent computation.

Hung-Hsuan Wu (吳宏宣) received the B.S., M.S., and Ph.D. degrees in Computer Science and Information Engineering from Tamkang University, Taipei, Taiwan in 1996, 2002 and 2009, respectively. He is currently an industrial teacher at the Department of Information Management of Tamkang University, Taipei, Taiwan, R.O.C. His research interests include music information retrieval, pattern recognition and image processing. He worked for K. C. Trade Co. as a sales assistant during 1997-1999, and as a software engineer in Kinpo Electronics, Inc. in 2002-2003.

Chun-Wei Wang (王駿瑋) received the B.S. and M.S. degrees in Computer Science and Information Engineering from Tamkang University, Taipei, Taiwan in 2004 and 2006, respectively. He is currently a Ph.D. candidate at the Department of Computer Science and Information Engineering of Tamkang University, Taipei, Taiwan, R.O.C. His research interests include image forgery analysis, pattern recognition and soft computing.