行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告 □期中進度報告

計畫類別：■ 個別型計畫　　□ 整合型計畫

計畫編號：NSC 96-2221-E-032-049

執行期間： 96 年 8 月 1 日至 97 年 7 月 31 日

計畫主持人： 林慧珍

共同主持人： 顏淑惠

計畫參與人員： 吳宏宣, 王駿瑋

成果報告類型(依經費核定清單規定繳交)：■精簡報告　□完整報告

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□赴大陸地區出差或研習心得報告一份

■出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢

□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：淡大資工系

中 華 民 國 97 年 8 月 24 日

Efficient Geometric Measure of Music Similarity

1. Introduction

With the rapid growth of digital audio data, contend-based music retrieval has become a popular research topic in the past few years. When searching for digital music one must do so by text. This means that some information such as the name of the song or the name of the artist must be remembered. In many cases, a user may not remember such information, even though they might be able to hum or sing a fragment of the song. In that case query by humming or singing would be a more reasonable search method. Many different music retrieval systems have been created for academic research or business in recent years. Some of these systems allow users to search for music by inputting a fragment of the query melody by singing or humming.

Similarity measurement in music matching is all important for a music retrieval system, whose effectiveness is directly affected by feature selection and distance metric. Although pitch is the most commonly used feature, it alone is not sufficient for search in a large database [3]. Many researchers [10][11] indicated that pitch and duration are the most important attributes, and their findings will be followed in this research.

Techniques based on string matching have been extensively explored in the literature, including edit distance [9] and n–gram [2][5]. However, their use of pitch sequences as feature has the drawback of retrieving too many irrelevant music documents with similar pitch sequences but rather different rhythm.

Clifford et al. [4] considered the music notes in a melody as a set of points in a two-dimensional Euclidean space. The music matching problem is equivalent to finding the partial subset match of two point sets. However, its discrete point representation has the drawback of low tolerance to slight displacement of points.

D. Ó. Maidín [8] proposed a geometric matching technique for music similarity measurement, considering both the pitch and rhythm information Each note is represented as a horizontal line segment so that a sequence of notes can be described as a rectangular contour in a 2-D coordinate system, in which the horizontal and vertical coordinates correspond to time and pitch value, respectively. This approach juxtaposed two melodies of equal duration and transposed one melody in the vertical direction (in pitch values) to evaluate the minimum area between them. Francu and Nevill–Manning [6] followed the geometric matching technique, except that both the query and the reference are sampled and quantized into a sequence of notes of equal small period (of 20 milliseconds). They rescaled and transposed the query in vertical and horizontal directions to find the minimum difference between the query and the reference. Aloupis et al. [1] improved the above two geometric matching techniques by using a binary search tree to achieve efficient search for the best match of two melodies. Our previous work [7] improved the efficiency of geometric matching by using pitch intervals instead of pitches to achieve key invariance and to avoid searching in the vertical direction, and utilizing a branch–and–prune mechanism to quickly discard most of the incorrect horizontal positions of the query. In this work, we further improve our previous work by constructing a balanced binary search tree to provide both quick determination of the step size of each move of the query melody and quick update of the area after each move.

The rest of this paper is organized as follows. Section 2 provides a detailed description of our proposed method including the definition of geometric measure of similarity between two melodies, the structure of the binary search tree used for improvement, and how the step size of a move of the query and the minimum area are efficiently determined using the tree. In Section 3 we draw our conclusions and provide suggestions for future work.

2. Geometrical Matching

The purpose of a music retrieval system is to match a short query melody against a larger reference melody and establish and rank the relevant references according to the similarity measurement. In this section, we briefly introduce our previously proposed geometric matching technique and describe how to utilize a balanced binary search tree to speed up the evaluation of the minimum area between two melodies.

2.1. Pitch–Interval Duration Sequences and Similarity Measure

Aloupis et al. [1] represented a melody as a sequence of music notes. Each note was described by a horizontal line segment, of which the height and the width denote the pitch value and the duration of the note, respectively. They shifted the incoming query in both the horizontal and vertical directions to find the minimum area between it and the reference melody, and this minimum area is defined as the distance the two melodies. Aloupis et al. indicated that the minimum area must occur in a case when two vertical edges coincide, one from the query and the other from the reference. There are at most mn such cases, since there might be a case with more than one coincident pairs, and thus it needs to evaluate the area for only these cases and the query needs to be shifted at most mn times in the horizontal direction. Fixing the two melodies in the horizontal direction and fixing the reference melody in the vertical direction, moving the query melody in the vertical direction to find the minimum area requires $O(n)$ time. Thus the minimum area between two melodies can be computed in $O(mn^2)$ time. They also constructed a binary search tree to speed up the search of the best vertical position for the query and ease area reevaluation at each position. To efficiently reevaluate the area after each horizontal shift of the query, the step size of a shift is set to the minimum of all possible steps that would cause another pair of vertical lines coinciding, which is just the minimum of the widths of the rectangles enclosed by the two melodies. Determination of the step size for each shift requires at least $\Theta(m)$ time, the search of the best vertical position requires $O(\log n)$ time, and tree modification (deletion and insertion) and reevaluation together for each move requires $O(\log n)$ time. Thus the total time required should be $O(mn(m + \log n))$ rather than $O(mn\log n)$ as they claimed. As discussed in [7], the average run time is $O(mn(m + \log m)) = O(m^2n)$.

Similar to our previous work, pitch interval–duration (PID) sequence is used in this work to represent a melody for the benefit of its key–invariance. Each note in a PID sequence can be also described by a horizontal line segment, of which the height and the width denote the pitch interval and the duration of the note, respectively. With the key–invariance property of this representation, the query needs to move in only the horizontal direction.

For a query $Q = (Q_1, Q_2,\ldots, Q_m)$ and a reference $R = (R_1, R_2,\ldots, R_n)$ represented by sequences of notes, where Q_i and R_j are the i-th note of the query and the j-th note of the reference, respectively. To form a PID sequence for a melody, we denote the pitch value of a note N by $P[N]$, let $q_i = P[Q_{i+1}] - P[Q_i]$ and $r_j = P[R_{j+1}] - P[R_j]$ to denote the i-th pitch interval of the query Q and the j-th pitch interval of the query R, respectively. The j-th pitch interval–duration of R and the i-th pitch interval–duration of Q are then represented as $<t_{r_j}, r_j>$ and $<t_{q_i}, q_i>$, respectively, where $t_{r_j} = start[R_{j+1}]$ is the starting time of note R_{j+1} but $t_{q_i} = start[Q_{i+1}] + \theta$ is the starting time of note Q_{i+1} plus a the time replacement θ, which is the total step size at the moment whenever it is referred, since the reference melody is fixed whereas the query melody is dynamical. For easy implementation, we define a dummy item $<t_{q_m}, q_m>$, where t_{q_m} = the ending time of Q_m plus θ and q_m = 0, with assumption that the ending time of one note is the same as the starting time of the next note. Then melody Q can be represented as the PID sequence $(<t_{q_i}, q_i>)_{i=1,2,\cdots,m}$. The PID sequence for the reference R is defined in the same fashion by $(<t_{r_j}, r_j>)_{j=1,2,\cdots,n}$.

2.2. Geometric Matching using A Binary Search Tree

After a move of the query, at least one pair of vertical edges coincide, some rectangles grow in width, some become narrower or even varnishing, and some are unaffected, and thus the area must be reevaluated. In this work, we utilize a balanced binary search tree to efficiently indicate all the changes and reevaluate the area.

2.2.1. Binary Search Tree Construction

As depicted in Figure 1, the query and the reference are modeled as monotonic pitch interval rectilinear functions of time. The region between two melodies is partitioned into rectangles C_α, $\alpha = 1, 2, \cdots, k$. Each rectangle can be specified by its four edges, each of the left side and the right side is formed by a vertical edge of the query or the reference; the top and the bottom edges are formed by a pitch interval of the query and a pitch interval of the reference. The distance between the two melodies is defined as the total area of these rectangles $A = \sum_{\alpha=1}^{k} |C_\alpha| = \sum_{\alpha=1}^{k} (w_\alpha \cdot h_\alpha)$, where w_α and $h_\alpha = |q_{i_\alpha} - r_{j_\alpha}|$ are the width and the height of the α-th rectangle C_α formed by pitch intervals q_{i_α} and r_{j_α}. The type of a rectangle can be determined by its vertical edges; that is, it is growing if its left edge and right edge are from the reference and the query, respectively; it is shrinking if its left edge and right edge are from the query and the reference, respectively; it is unaffected if the two edges are both from the query or both from the reference. As depicted in Figure 1, the rectangles between the query and the reference are categorized into 3 types by the sets $W_T^- = \{1, 4\}$, $W_T^0 = \{3, 5, 7\}$, and $W_T^+ = \{2, 6\}$.

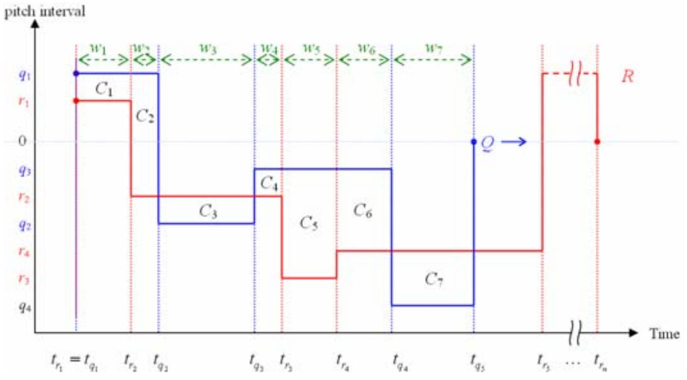


Figure 1. The region between the query and the reference was partitioned into 7 rectangles.

The rectangles are stored in the leaves of a balanced binary search tree. Each leaf stores the four edges of one rectangle, thus the width w_α and the height h_α of each rectangle C_α can be computed easily. Each leaf also has a label to indicate the type of the corresponding rectangle, growing, shrinking, or unaffected. We also use three sets, W^-, W^0, and W^+, to store the indices of growing, shrinking, and unaffected rectangles, respectively. The leaves are ordered by the key value of k_α, as defined in Eq. (1), so that the shrinking rectangles are stored in front of the sequence of leaves, followed by the unaffected rectangles, and then the growing rectangles. As shown in Figure 1, rectangle C_2 is specified by the four edges (left edge, right edge, query interval, reference interval) = $(t_{r_2}, t_{q_2}, q_1, r_2)$ in a leaf with key value $k_2 = w_2 + 2L = t_{q_2} - t_{r_2} + 2L$ since it is growing. The unaffected rectangle $C_3 = (t_{q_2}, t_{q_3}, q_2, r_2)$ has key value $k_3 = w_3 + L = t_{q_3} - t_{q_2} + L$, and the shrinking rectangle $C_4 =$ $(t_{q_3}, t_{r_3}, q_3, r_2)$ has key value $k_4 = w_4 = t_{r_3} - t_{q_3}$.

At every internal node with subtree T we store H_T^-, H_T^0, and H_T^+, which are the sums of heights over shrinking, unaffected, and growing leaves, respectively, in T.

$$k_\alpha = \begin{cases} w_\alpha & \text{if } \alpha \in W^- \\ w_\alpha + L & \text{if } \alpha \in W^0 \\ w_\alpha + 2L & \text{if } \alpha \in W^+ \end{cases}, \text{ where } L = sq_m - sq_1 \qquad (1)$$

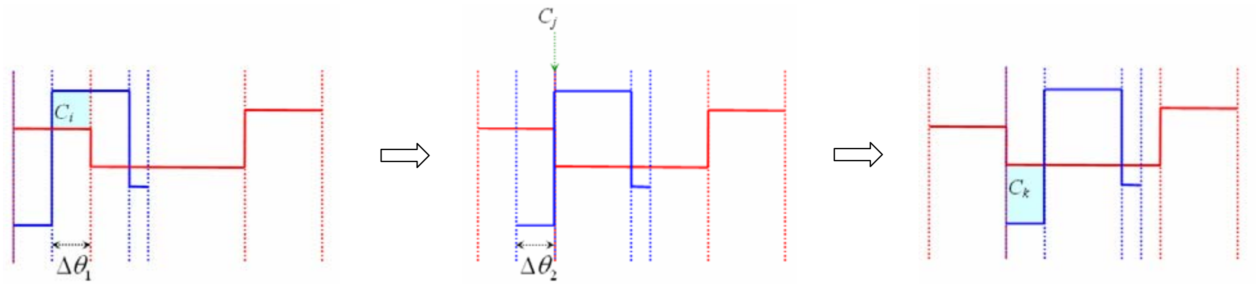
Since we only consider the horizontal movement of the query that causes two vertical lines coinciding (or one rectangle vanishing), the step size of a move is taken as the minimum of all such possible movements, which is just the minimum of the widths of all the shrinking rectangles. This way ensures that the minimum area can be found. The construction of the balanced binary search tree allows us to quickly determine the step size $\Delta\theta$ of each move of the query, by finding the width of the first shrinking leaf (or the narrowest shrinking rectangle, which has the minimum key value and is stored in the leftmost leaf of the tree), if any. That is, $\Delta\theta = \min_i \{w_i \mid i \in W^-\}$, which can be found in $O(\log k)$ time, where the k is the number of leaves (or rectangles). Usually $k = O(m+n)$, but $k = O(m)$ on the average, if the notes of the query are much fewer than the reference. Hence, the step size $\Delta\theta$ can be determined in $O(\log m)$ time.

As soon as the step size $\Delta\theta$ is determined, we increment the so–far total step size θ by $\Delta\theta$, i.e., $\theta \leftarrow \theta + \Delta\theta$.

2.2.2. Tree Update

After a move of the query, at least one pair of vertical edges coincide with each other, some rectangles are vanishing, some are created, and some change their types. If the width of a shrinking rectangle is equal to the step size of the incoming move, then it will be temporarily vanishing after the move and growing up after next move. Such a vanishing rectangle is considered as a growing rectangle of area zero and stored into a leaf of the tree. For example, if the query shown in Figure 2(a) makes a move of step size $\Delta\theta_1$, the area of rectangle C_i becomes C_j of area zero as shown in Figure 2(b). After another move of step size $\Delta\theta_2$, C_j would be growing up into a new rectangle C_k, as shown in Figure 2(c).

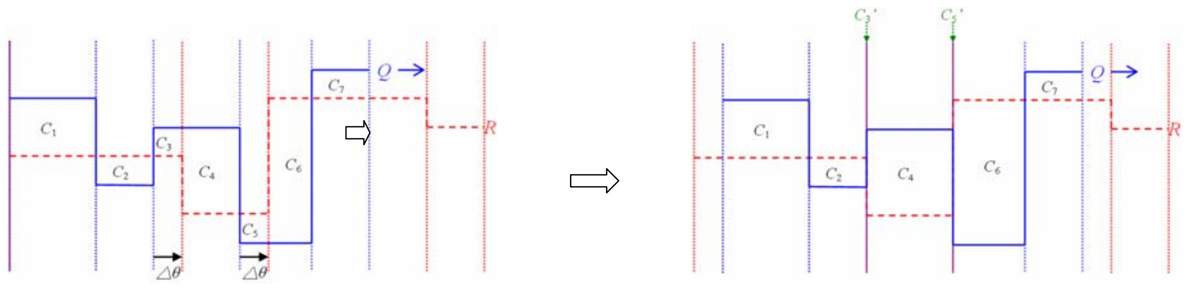
For each shrinking rectangle C_i to be vanishing after a move, the types of the rectangles to its left and right, say C_{i-1} and C_{i+1}, must be either unaffected or growing. If C_{i+1} is growing or unaffected, then the move would change its type to "unaffected" or "shrinking", respectively. The type of C_{i-1} is updated in a similar manner.



(a) (b) (c)

Figure 2. (a) A shrinking rectangle C_i. (b) The rectangle C_i in (a) is vanishing and the rectangle C_j with zero area is created after the query make a move of $\Delta\theta_1$. (c) The rectangle C_j in (b) grows up into a new rectangle C_k after another move of $\Delta\theta_2$.

Consider the example depicted in Figure 3(a), since the widths of the shrinking rectangles C_3 and C_5 are both equal to $\Delta\theta$, they would be vanishing and two rectangles C_3' and C_5' with zero area would be created, after the move of step size $\Delta\theta$, as shown in Figure 3(b). Moreover, the rectangles adjacent to C_3 or C_5, say C_2, C_4, and C_6, would change their types. While checking the neighbors of C_3, the rectangles C_2 and C_4 change their types from unaffected and growing to shrinking and unaffected, respectively. And then while checking the neighbors of C_5, the rectangles C_4 and C_6 change their types from unaffected and growing to shrinking and unaffected, respectively. Note that during this move C_4 changes types twice. To keep the order of rectangles, we use a linked list to link the leaves corresponding to the rectangles from left to right in the region enclosed by the two melodies. In such a way, neighbors of a rectangle can be easily referred to.



(a) (b)

Figure 3. (a) The region between the query and the reference is partitioned into 7 rectangles. (b) After a move, the rectangle C_3 and C_5 are vanishing, and their neighboring rectangles change types.

Finding the step size $\Delta\theta$ for next move requires $O(\log m)$ time. On average, the number of leaves to be inserted or deleted in each step is a constant. For each newly inserted or deleted leaf, it requires $O(\log m)$ time to update its ancestors. After each move, each growing rectangle increases its width by the step size and each shrinking rectangle decreases its width by the step size, and thus the area between the two melodies can be recalculated immediately. Since H_T^-, H_T^0, and H_T^+ are recorded in the root of the tree, the area can be updated easily by adding the value $(H_T^+ - H_T^-)\cdot\Delta\theta$ to the current value of area A; that is, $A \leftarrow A + (H_T^+ - H_T^-)\cdot\Delta\theta$. At most $O(mn)$ moves are required, and thus the total time required to compute the minimum area is $O(mn \log m)$. The algorithm ***Pitch–Interval–Area with BST*** for computing the minimum area between two melodies is given in the following.

```
Algorithm Pitch–Interval–Area with BST
Input: reference R and query Q
1.  Initialization: Construct the initial binary search tree (BST), evaluate initial area $A \leftarrow A_0$;
    $MinArea \leftarrow A_0$; and initialize total step of move $\theta \leftarrow 0$.
      While ($t_{q_m} \leq t_{r_n}$) and ($MinArea$ > Treshold) do      // $t_{q_i} = sq_i + \theta$
    Evaluate step of move $\Delta\theta \leftarrow$ the width of the leftmost leaf.
    For each shrinking rectangle $C_i$ of width = $\Delta\theta$ do
              //In the following, as soon as a node is deleted or inserted, all its ancestors are updated.
      For each of the rectangles adjacent to $C_i$
      if it is growing then remove it and insert a new unaffected leaf
      if it is unaffected then remove it and insert a new shrinking leaf
     Remove $C_i$ and create a new growing leaf $C_j$' of area zero.
      Update: $A \leftarrow A + (H_T^+ - H_T^-) \cdot \Delta\theta$;   $\theta \leftarrow \theta + \Delta\theta$
      If $A < MinArea$ then $MinArea \leftarrow A$
```

3. Conclusions and Future Work

The proposed algorithm improves the time efficiency for geometric measure of music similarity in two aspects: (1). matching the pitch interval instead of the pitch sequence to achieve key invariance, and (2). utilizing a balanced binary search tree to efficiently determine the step size and update the area for each move. This improvement reduces the average time complexity to $O(mn \log m)$, which is better than the work done by Aloupis et al. and our previous work, both requiring $O(mn(m + \log m)) = O(m^2n)$ on average.

Our future work will concentrate on a continuing improvement of measure of music similarity, with particular focus on the scaling problem of duration and polyphonic music matching.

References

[1] G. Aloupis, T. Fevens, S. Langerman, T. Matsui, A. Mesa, Y. Nunez, D. Rappaport, and G. Toussaint, Algorithms for computing geometric measures of melodic similarity, Computer Music Journal 30(3) (2006) 67–76.

[2] D. Bainbridge, M. Dewsnip, and I. H. Witten, Searching digital music libraries, Information Processing and Management 41(1) (2005) 41–56.

[3] D. Byrd and T. Crawford, Problems of music information retrieval in the real world, Information Processing and Management 38 (2002) 249–272.

[4] R. Clifford, M. Christodoulakis, T. Crawford, D. Meredith, and G. Wiggins, A fast, randomised, maximal subset matching algorithm for document–level music retrieval, in: Proc. of ISMIR, 2006, pp. 150–155.

[5] J. Foote, An overview of audio information retrieval, Multimedia Systems 7(1) (1999) 2–10.

[6] C. Francu and C. G. Nevill–Manning, Distance metrics and indexing strategies for a digital library of popular music, in: Proc. of the IEEE International Conference on Multimedia and EXPO, 2000, pp. 889–892.

[7] H. J. Lin, H. H. Wu, and Y. T. Kao, Geometric Measures of Distance between two Pitch Contour

Sequences, Journal of Computers (2007), to appear.
[8] D. Ó. Maidín, A geometrical algorithm for melodic difference, Computing in Musicology 11 (1998) 65–72.
[9] M. Mongeau and D. Sankoff, Comparison of musical sequences, Computers and the Humanities 24 (1990) 161–175.
[10] L. A. Smith, R. J. McNab, and I. H. Witten, Sequence–based melodic comparison: A dynamic programming approach, Computing in Musicology 11 (1998) 101–117.
[11] F. Wiering, R. Typke, and R. C. Veltkamp, Transportation distances and their application in music–notation retrieval, Computing in Musicology 13 (2004) 113–128.