A Robust Music Retrieval Method for Query-by-Humming

Yongwei Zhu
Institute for Inforcomm Research
Singapore
ywzhu@i2r.a-star.edu.sg

Mohan S Kankanhalli
School of Computing
National University of Singapore
Singapore
mohan@comp.nus.edu.sg

Abstract **The increasing availability of digital music has created a need for effective music retrieval methods. In this paper, we present a novel content-based music retrieval method that is robust against pitch errors and tempo variations in the queries, which is especially advantageous for query-by-humming. The melody of a music item and that of a hummed query are represented by point sequences, and the skeletons of the *melody* are used to do melody searching and alignment. A novel point skipping dynamic programming algorithm is proposed for robust and efficient melody skeleton matching. The melody similarity measure is then computed based on the alignment of the point sequences. Our experiment, including a comparison to our previous methods, has demonstrated the performance of the method.**

Index Terms— **Music retrieval, Query-by-Humming**

I. INTRODUCTION

Due to the increasing availability of digital music content, effective retrieval of such data is becoming very important. Music information retrieval has therefore become an area of active research in the recent years. Several music retrieval techniques for query-by-humming have been proposed, since for most people humming is the easiest way of producing a music query.

The inevitable note articulation error, pitch inaccuracy and tempo variation in the humming, however, pose great challenges for effective music retrieval. Previous string matching approach [1-4] has much limitation for query-by-humming, since it relies on accurate note segmentation, although it is not much affected by tempo variation. A beat-based approach in [5] requires the user to hum by following a metronome, which is rather restrictive. A time series matching approach proposed in [6-8] has shown effectiveness for query-by-humming in terms of robustness against note errors, since accurate note segmentation is not needed. However, the time-warping distance computing used in [6,7] is time consuming and cannot handle well when there are both pitch error and tempo variation. And [8] requires the tempo variation to be linear.

In this paper, we present a novel melody representation and matching method, which is both robust against pitch errors and invariant to liner or non-linear tempo variation. The melody of a music item or a query is represented by a point sequence, which is derived from the pitch contour of the melody. This point sequence is invariant to the time or speed in the original melody contour. Important anchor points in the sequence, called melody skeleton, are used for melody searching and alignment. A specialized dynamic programming algorithm is proposed for robust melody skeleton matching and alignment. The melody similarity measurement of the whole point sequences is then computed based on the precise alignment. We achieve (1) tempo invariance by using the novel melody representation, (2) robustness to pitch error by using melody skeleton matching, and (3) high retrieval accuracy by using melody alignment. Furthermore, the melody skeleton matching can serve as a filter that rejects the large portion of the wrong candidate, which promote the retrieval efficiency.

This paper is organized as follows: section 2 briefly describes melody representation using point sequence derived from melody contour; section 3 describes a specialized dynamic programming method for melody skeleton search and a melody similarity metric; section 4 discusses experimental results and the last section presents the conclusions.

II. MELODY REPRESENTATION

Proper melody representation is very important for melody based music retrieval. It forms the basis for melody similarity measure and the search procedure.

Melody contour or pitch contour used in [6,7], which is a time series of pitch values, represents melody content without using explicit music notes. We proposed a greedy algorithm [9] to approximate the melody contour using a sequence of line segment (Fig.1.(a)). The algorithm guarantees that approximation error is within a certain range, for instance, less than half semitone. The line segment sequence is a more compact melody representation than the time series melody contour. Furthermore, we map the line segment sequence into a sequence of data points (Fig.1.(b)), where each line segment is mapped to a point (x_i,y_i). The vertical coordinate y_i is the pitch value, which is the same as the pitch value of the corresponding line segment. The horizontal coordinate x_i is

0-7803-7724-9/03/$17.00 © 2003 IEEE

the so-called *value run*, which is derived using following equation:

$$x_i = x_{i-1} + \left| y_i - y_{i-1} \right|, \text{ where } x_1 = 0 \qquad (1)$$

We call the points that are local maximum and minimum in pitch values the extreme points, and the other points the non-extreme points. It can be easily proved that non-extreme points, such as point B in Fig.1, reside on the straight lines that connecting the extreme points, such as A and C, as shown in Fig.1(c). As a result, pitch value inaccuracy of the non-extreme points do not affect the structure established by the extreme points (local maximum/minimum). We call the extreme points in the sequence the *melody skeleton*.

Our melody retrieval method is based on the point sequence representation. The point sequence for a hummed query is matched with the point sequences in the database, and the most similar melody is returned as retrieval result. This melody representation has the property of tempo invariance, since the time information is eliminated. And the melody skeleton structure is robust to pitch errors in the query. Due to space constraint, readers please refer to [9] for the details of this melody representation.

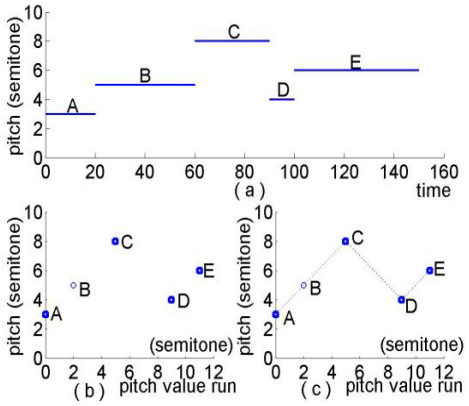

Fig.1. Melody representation using point sequence

III. MELODY MATCHING

Our melody matching method has 2 steps: (1) melody localization and alignment using melody skeleton; (2) melody similarity measurement based on the alignment.

3.1. Melody Skeleton Matching

The melody skeleton matching serves two roles: (1) locates only the likely candidates who have a skeleton similar to that of the query melody; (2) provides a proper alignment between the query data sequence and the candidate data subsequence.

In melody skeleton matching, only the extreme points are utilized, and it is assumed that for a valid matching the maximum point matches only maximum point and minimum point only matches minimum point. The assumption is based on the robustness and invariance property of the melody skeleton. However, to accommodate errors in melody skeleton, we allow some points to be skipped, i.e. not matched to any points. Note that, under the above assumption, points are skipped in pairs (2, 4, …).

We propose a specialized dynamic programming algorithm for melody skeleton matching. A query data sequence is denoted as *q[i],* where $1 \le i \le m$, i is the index of the sequence, m is the number of points in the sequence. The pitch value and value run of *q[i]* are denoted as *qv[i]* and *qr[i]*.

A target data sequence is denoted as *t[i]*, where $1 \le i \le n$, i is the index of the sequence, n is the number of points in the sequence. The pitch value and value run of *t[i]* are denoted as *tv[i]* and *tr[i]*.

For the simplicity of presentation, we assume $n > m$, and *q[1]* and *t[1]* are both peak (maximum) points or both valley (minimum) points.

A table for calculating the distance between two sequences starting from *q[1]* and *t[1]* is illustrated in Fig.2.

A value of a cell in the table $D_{i,j}$ stands for the minimum accumulated distance of *(q[1],…,q[i])* to *(t[1],…,t[j])*. Since a peak point does not match with a valley point, the distance values of the shaded cells in the table are not computed.

In this dynamic programming formulation, there are two issues of concern: (1) computing the distance value in a cell; (2) tracing the path of an alignment that has the minimum distance.

In our method, we use accumulated distance for each cell *(i,j),* which means $D_{i,j}$ equals a local distance added by the distance value $D_{x,y}$ of a "previous" cell *(x,y)*. Depending on the possible cases of point skipping, the possible "previous" cells of *(i,j)* are illustrated in Fig.3.

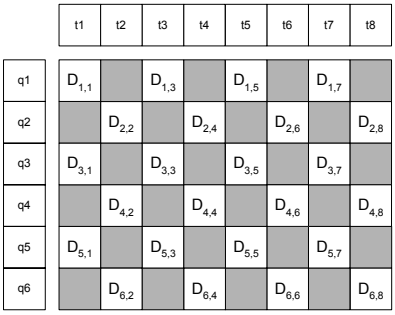

Fig.2. The table for computing the distance of t[i] and q[i]

If the cell *(i-1,j-1)* is the previous cell, then it means there is no point skipping for $D_{i,j}$. If *(i-1,j-3)* or *(i-3,j-1)* is the previous cell, then there is a 2-point-skipping in either point sequence. If *(i-1,j-5)* or *(i-5,j-1)* is the previous cell, then there is a 4-point-skipping in either point sequence. If *(i-3,j-3)* is the previous cell, then there is a 2-point-skipping in both of the two point sequences. Other possibilities of previous point for *(i,j)* are not considered in our algorithm, since they are very unlikely to be present.

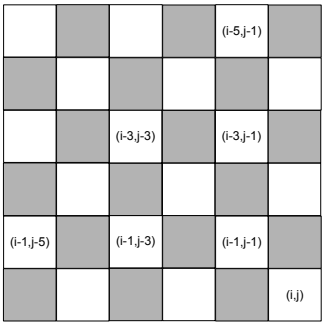

Fig.3. The possible previous cells for (i,j)

With the possible previous cells for *(i,j)* given, the distance value for $D_{i,j}$ can then be computed as follows:

$$D_{i,j} = d_{base}(i,j) + \min \begin{cases} D_{i-1,j-1} \\ D_{i-1,j-3} + P(i,-1,j,-3) \\ D_{i-3,j-1} + P(i,-3,j,-1) \\ D_{i-1,j-5} + P(i,-1,j,-5) \\ D_{i-5,j-1} + P(i,-5,j,-1) \\ D_{i-3,j-3} + P(i,-3,j,-3) \end{cases} \quad (2)$$

where i>3 or i>5 or j>3 or j>5 are required for the respective case to be considered.

$$d_{base}(i,j) = |qv(i) - tv(j) - \lambda| \quad (3)$$

$$\lambda = qv(1) - tv(1) \quad (4)$$

$$P(i,-k,j,-l) = P_Q(i,k) + P_T(j,l) \quad (5)$$

$$P_Q(i,k) = 0, \text{ if } k = 1 \quad (6)$$

$$P_Q(i,k) = \eta \sum_{x=1}^{(k-1)/2} |qv(i-2x+1) - qv(i-2x)| \quad (7)$$

$$P_T(j,l) = 0, \text{ if } l = 1 \quad (8)$$

$$P_T(j,l) = \eta \sum_{x=1}^{(l-1)/2} |tv(j-2x+1) - tv(j-2x)| \quad (9)$$

$d_{base}(i,j)$ is the local distance between *q[i]* and *t[j]*, and λ is the shifting between *q[1]* and *t[1]*. *P(i,-k,j,-l)* is the penalty imposed for point skipping, in which $P_Q(i,k)$ is the penalty for skipping points in query, and P_T is the penalty for skipping points in target. The penalty is based on the sum of the value differences of the pairs of points that are skipped. η is a weight for the penalties, which takes a value of 1 in our algorithm.

The previous cell, which gives *(i,j)* the minimum distance value, is chosen and recorded using a path table, which stores the pointers to (or the index of) the respective chosen previous cells.

The border cells are initialized as:

$D_{1,1} = 0$;

$D_{1,j} = \infty$; *for* $j > 1$

$D_{i,1} = \infty$; *for* $i > 1$

since the alignment starts with *q[1]* and *t[1]*.

The order of computation of distance values for other cells is from top to bottom and from left to right. Since the possible previous cells and the border initialization are known, not all the cells in the table need to be computed. This is because distance values of some cells are determined to be ∞. Furthermore, the value-run *qr* and *tr* can also be used to constrain the number of cells to be computed. Because for an valid alignment, the mapped points from query sequence and target sequence should not have large difference in their value run after shifting the run difference between *q[1]* and *t[1]*.

After the computation of distance value of the cells, the best alignment is obtained by locating the $D_{m,x} = \min_j D_{m,j}$, which means *(q[1],...,q[m])* has the minimum accumulated distance with *(t[1],...,t[x]),* and $D_{m,x}$ is the distance value.

The mapped path is obtained by tracing back from the cell *(m,x)* in the path table. The tracing is stopped when the pointer points to cell *(1,1)*.

The above mentioned dynamic programming technique will find the best subsequence of target sequence starting from *t[1],* which can be aligned with the query sequence *(q[1],...,q[m]).* For the other subsequence in the targeting sequence starting from *t[1+2x] (x>0),* the dynamic programming computation can be done in a same way by replacing *t[1]* by *t[1+2x]*.

Thus finally, for each starting position *(2x-1) (0 <x < n/2+1)* in the target sequence, the best alignment with the query sequence is found and the corresponding accumulated distance $D_m(x)$ is obtained. In these *n/2* alignments, the alignments at the following position are selected as matches with the query sequence based on $D_m(x)$*:*

$D_m(x)$ is a local minimum;

$D_m(x) < D_{thres}$.

The local minimum of $D_m(x)$ is selected, because the best alignment should always have a smaller distance than the alignment at adjacent positions. D_{thres} is a threshold, which is to ensure that the aligned target subsequence is close enough to the query sequence. In our algorithm, we use $D_{thres} = m-1$, which means 1 semitone error tolerance is given to every (excepting the first) extreme point in the query. The selected target subsequences are the likely candidates, on which an accurate final melody similarity will be computed.

This melody skeleton matching method requires the first extreme point in the query data to be reliable. The experiments show that extreme points are robust against variations in the humming, and are reliable for the alignment. To make it even more reliable, an extreme point that has the largest value differences with its predecessor and successor extreme points can be used.

3.2. Melody Similarity Measure

The melody similarity measure is based on the final point alignment, which consists of two steps: (1) aligning all the data points of the two sequences, including the non-extreme points; (2) computing the similarity of the two sequences.

Since the alignment of the extreme points of the two sequences is already done, we only need to align the non-extreme points and skipped extreme points (empty point in Fig.4.) between two not skipped extreme points (solid point in Fig.4.) in a sequence with the corresponding points in the other sequence. This can be easily done using traditional dynamic programming algorithm.

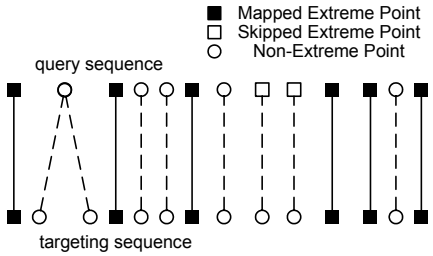

Fig.4. Alignment of the whole points sequences

The final similarity between 2 points sequence is computed based on the cumulated distance of the final alignment of the 2 sequences.

IV. EXPERIMENTS

In our experiments we choose 10 tunes including English and Chinese songs as the query melodies. 5 human subjects including 3 males and 2 females hummed each of the tune four times, once in a normal tempo, once in a faster tempo, once in a slower tempo, and once in combination of faster & slower tempo. So there are totally 200 hummed queries. The hummed part is at the beginning of the query melodies, however we search anywhere in the middle of all the target melodies in the database. We use 2000 MIDI files in our database.

Robustness of the melody skeleton against (linear and non-linear) tempo and pitch variations is the main feature of our method. Fig.6 shows 6 hummed queries of the tune "Happy Birthday To You" using different tempos by different subjects. Fig.6(d) shows the query hummed in normal speed. Fig.6(a) shows a faster tempo. Fig.6(e) and (f) shows slower tempos. Fig.6(b) and (c) shows inconsistent tempos.

Each subfigure in Fig.6 shows the original query time series, line segments approximation and the value-run domain data points. It can be seen that the melody skeleton structure formed by extreme points is very robust, as it is almost identical for all the 6 queries.

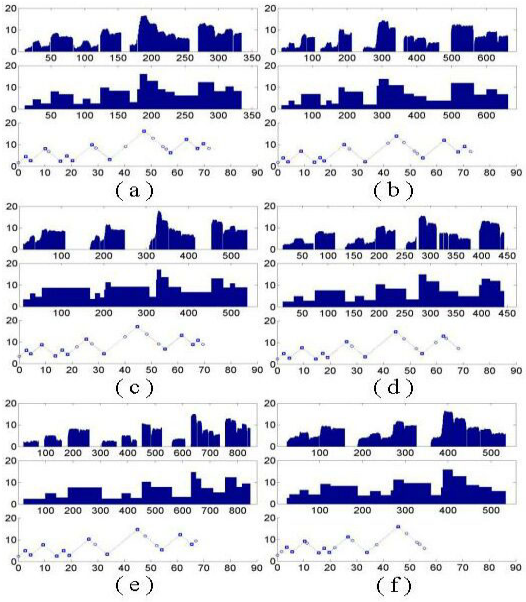

Fig.6. Six hummed queries of a same tune

4.1. Melody Skeleton Matching

To evaluation the performance of melody skeleton matching, we examine the correct hit rate and candidate elimination rate. The correct hit rate is the number of queries that have matches with the correct target divided by the total number of queries. The candidate elimination rate is average number of eliminated candidates divided by the total number of possible candidates.

The 2000 MIDI files in our database have a total of 185604 extreme points in the melody skeletons. Considering that the matching starts from only a peak point or a valley point, there are totally 185604/2=92802 possible matching candidates.

94.5% (correct hit rate) of the queries can match with the correct target melodies. And on an average, 5130 candidates are returned for each query. So the candidate elimination rate is 1-5130/92802=94.47%.

4.2. Melody Retrieval

For the performance of melody retrieval, we conducted two types of evaluation: one for matching anywhere in the middle of songs and one for matching only at the beginning of songs.

It has achieved 43% for top 1, 64% for top 2, 76% for top 5, 80% for top 10, and 88% for top 20.

The average time for each query is about 4 seconds for the anywhere matching, which is faster than [9] which takes 7.6 seconds for only matching the beginning of 2000 songs.

To examine the retrieval performance of matching the beginning of songs, we simply discard the retrieved entries that are not at the beginning of the tune. The retrieval rate is higher than that for matching in the middle of songs. It has achieved 55% for top 1, 71% for top 2, 86% for top 5, 90% for top 10, and 92% for top 20. In general it has a better

performance than [9], which achieved 56% for top 1, 68% for top 3, and 85% for top 20.

V. Conclusion

We have presented a method for content-based music retrieval which uses a novel tempo invariant melody representation. The representation is obtained by converting melody contour (in form of line segments) to a sequence of points. The extreme points in the sequence are called melody skeleton. There are two steps in doing melody matching. The first step is a matching based on melody skeleton, which filters out unlikely candidates and aligns 2 melody skeletons properly. A point skipping dynamic programming matching algorithm is proposed for melody skeleton matching. The second step is the matching of the whole point sequence, which is based on the alignment achieved in the first step. The final melody similarity is computed based on the pitch distance between the corresponding matched points.

The advantages of the method are: 1) it is tempo invariant; 2) it is robust by using the melody skeleton representation and matching; 3) it is efficient since melody skeleton matching can filter out large proportion of the candidates; 4) it is effective since the skeleton alignment improves the final similarity measurement. Our experiment for query-by-humming has demonstrated the performance of the method.

References

[1] A. Ghias, J. Logan, and D. Chamberlin. "Query By Humming". *Proceedings of ACM Multimedia 95,* November 1995, pages 231-236.

[2] R.J. McNab, L.A. Smith, I.H. Witten, C.L. Henderson and S.J. Cunningham. "Towards the digital music library: tune retrieval from acoustic input". *Proceedings of ACM Digital Libraries'96*, 1996, pages 11-18.

[3] S. Blackburn and D. DeRoure. "A Tool for Content Based Navigation of Music". *Proceedings of ACM Multimedia 98,* 1998, pages 361-368.

[4] A. Uitdenbogerd and J. Zobel. "Melodic Matching Techniques for Large Music Database". *Proceedings of ACM Multimedia 99,* November 1999, pages 57-66.

[5] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamanuro, and K. Kushima. "A Practical Query-By-Humming System for a Large Music Database". *Proceedings of ACM Multimedia 2000*, Los Angeles USA, 2000, pages 333-342.

[6] J.S. R. Jang, H.R. Lee, "Hierarchical Filtering Method for Content-based Music Retrieval via Acoustic Input", *Proc. ACM Multimedia 2001*.

[7] T. Nishimura, H. Hashiguchi, J. Takita, J. X. Zhang, M. Goto, and R. Oka, "Music Signal Spotting Retrieval by a Humming Query Using Start Frame Feature Dependent Continuous Dynamic Programming", *Proc. 3rd International Symposium on Music Information Retrieval*, Indiana, USA, October 15-17, 2001.

[8] Y.W. Zhu, M.S Kankanhalli and C.S. Xu, "Pitch Tracking and Melody Slope Matching for Song Retrieval", *Proc. Second IEEE Pacific-Rim Conference on Multimedia PCM2001*, Beijing, October 2001.

[9] Y.W. Zhu, M.S Kankanhalli, "Value-Run Domain Representation for Content-Based Music Retrieval", LIT Technical Report, 2002.