Melody Extraction from Polyphonic MIDI Files Based on Melody Similarity

Liu Li, Cai Junwei
School of Information Science and Engineering
Lanzhou University
Lanzhou, China
liulnd@lzu.edu.cn, caijw05@st.lzu.edu.cn

Wang Lei
Institute of Automation
Chinese Academy of Sciences
Beijing, China
leiwang@hitic.ia.ac.cn

Ma Yan
School of Information Engineering
Lanzhou University of Finance and Economics
Lanzhou, China
mayan@lzcc.edu.cn

Abstract—**The aim of melody extraction is to identify the melody from polyphonic musical files. Although researchers have made great progresses in this field, the need for more efficient methods is still not satisfied. This paper proposes an algorithm for extracting the main melody from polyphonic MIDI files based on the melody similarity computation. Experiments show that this approach is effective. We have applied this method in a hamming retrieval system. Compared with the manually labeled method, our method can identify the melody from MIDI files efficiently and automatically.**

Keywords-melody extraction; melody similarity; MIDI; track

I. INTRODUCTION

With the increasing applications of multimedia data on the Internet and multimedia databases, there are more and more researches on automatic classifying, indexing and retrieving of multimedia data. This is particularly true for music files, since much more online music storing and searching applications are available. Text-based retrieval is only effective for music files with metadata. Content-based music information retrieval is to access the desired music files by melody, rhythm, timbre and other features. How to extract these content features is an important and difficult task in content-based music information retrieval [1].

For melody extraction, lots of researches have focused on monophonic MIDI (Musical Instrument Data Interface) [2]. In recent years, much of the work focuses on polyphonic MIDI [3,4,7]. Most existing melody extraction methods can be summarized into four categories: (1) Based on rules: judge the main track based on some experiences and the characteristics of the tracks [4]. (2) Based on statistics: use traditional methods of pattern recognition. (3) Based on the largest repeat pattern: identify the largest repeat pattern as the main track [2]. (4) Artificial mark: mark the main track by artificial ways.

Melody similarity [5] is one of the key technologies in content retrieval, mainly focusing on how to calculate a measure that reflects the degree of similarity or dissimilarity between a pair of melodies or melodic segments. This paper proposes an algorithm for extracting the melody from polyphonic MIDI by calculating the similarity between different MIDI files of a song to identify the main track.

The paper is organized as follows. In section II, we introduce the melody extraction algorithm based on melody similarity. Experiments and results are given in section III. In section IV, we conclude the paper.

II. MELODY EXTRACTION ALGORITHM

Usually, in different MIDI files of a song, every file should contain some main tracks that can be seen as the melody. The melody track should appear of high probability in these files. Based on this fact, we use WebCrawler to download different MIDI files of a song from the Internet, and then use melody similarity theory to calculate the similarity between two MIDI files. Each file can be regarded as a vector that has at most 16 tracks, so we can get their similarity matrices and determine the melody track of every MIDI file by analyzing these similarity matrices. The algorithm MelodyExtraction is described in the following:

```
Algorithm MelodyExtraction (F, D)
{// F: song files list;
 // D: every song have D pieces of different MIDI files.
1  i=0; j=0; k=0;
2  For each song S in the song files list F {
3    For each MIDI file M of S {
4     Step1: Representing MIDI file;}
5     while (j++< D) {
6        k=j+1;
7        while (k< D) {
8     Step2: Computing melody similarity and obtaining
             similarity matrix;
9     Step3: Dealing with special circumstance;
10    Step4: Identifying main track as melody if similarity
             matrix is not symmetrical; Else Continue ;}
11      }
12   }
}
```

A. Representing MIDI File

To compute the similarity between each track, we must find a way to represent every track of a MIDI file that preserves essential features without becoming overly

978-0-7695-3494-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ISISE.2008.228

complex to store and process. There are many representing ways, such as Parsons Code, melody chains and so on [5,6].

We use a string composed by relative pitch of each note to represent a track. This method is simple but can preserve essential features to a large extent. There are six steps to generate the string automatically.

1) Parse: According to the protocol, a MIDI file can be parsed into notes stream. Each note is denoted as a vector:

$$(Channel, Start, Duration, Pitch, Velocity). \quad (1)$$

2) Separate tracks: Separate each track by *Channel* to get an array denoted by $Track[16][n]$, each MIDI file has 16 tracks at most and each track has n pieces of notes.

3) Merge: There are more than two notes playing simultaneously, or playing nearly simultaneously in each track in the case of

$$\begin{aligned} &Track[i][j].Start = Track[i][j+1].Start \\ &Track[i][j+1].Start\text{-}Track[i][j].Start <= \delta, \end{aligned} \quad (2)$$

where, δ is a time parameter and $0 \le \delta \le 50$ ms.

In this matter, we only need extracting the highest pitch.

4) Get pitch sequence: Get the pitch sequence $Type[]$ of each track:

$$Type[] = \{a_1, a_2, ..a_j, ..a_n \mid 0 < a_j \le 127\}, \quad (3)$$

where, a_j is the value of the pitch.

5) Difference: In the former step, we get the absolute pitch sequences, but they can not represent the track because their pitches may be in difference altitude for different MIDI files. So we should compute the difference value $Dtype[]$:

$$\begin{aligned} &Dtype[i] = Type[i+1] - Type[i], 0 \le i < l-1 \\ &Dtype[n-1] = Type[n-1] - Type[0], n = l-1, \end{aligned} \quad (4)$$

where, l represents the length of array $Type[]$.

6) Get string sequence: Change each pitch value into its corresponding ASCII character, then every track of a MIDI file can be represented as a pitch string.

After these six steps, a MIDI file can formally be represented as follows:

$$M(k,s) =< Cs_1, Cs_2, Cs_3, Cs_4 ..., Cs_{16} >, \quad (5)$$

where, Cs_i represents the string of track i for file s of music k.

B. Computing Melody Similarity

Given two MIDI files, after the above six steps, they can be represented as $M(k,s)$ and $M(k,t)$. In order to compare the similarity between their tracks, we introduce the definition of similarity matrix.

Definition 1: Similarity matrix is a 16×16 matrix that $matrix[i][j]$ is the similarity value between $M(k,s).Cs_i$ and $M(k,t).Ct_j$.

Constructing the matrices is the kernel of the algorithm. While a song have D diffident MIDI files, we will get C_D^2 diffident matrices. Then by analyzing these matrices we can identify the main track. In our algorithm, we use the following three methods to measure the similarity value.

1) N-Grams method: It is a method usually used in text retrieval. N-grams are n-sized substrings of a specific string. Reference [7] introduces how to change the music information to N-grams and use it to compute melody similarity in music retrieval.

The main idea of using N-grams to compute the similarity between tracks is computing the number of n-sized substrings of each track to get two arrays, $array1$ and $array2$. Then we get similarity matrix by computing the Euclid distance of the two arrays:

$$matrix[i][j] = \sum_{m=0}^{N} (array1[m] - array2[m])^2. \quad (6)$$

The smaller the value of $matrix[i][j]$, the more similar two tracks are.

2) LCS method: LCS (The Longest Common Subsequence) is also used to measure the similarity between sequences. To computer two tracks' similarity, we can first compute the length of the LCS:

$$LCSLen = LCSF(M(k,s).Cs_i, M(k,t).Ct_j), \quad (7)$$

where, $LCSF$ is a function that computes the length of LCS between two tracks that using $LCSLen$ as return value. The function can be solved by Dynamic Programming [8].

Then, we can calculate the value of $matrix[i][j]$ by $LCSLen$:

$$matrix[i][j] = \begin{cases} \dfrac{LCSLen}{len(Ct_j)}, & len(Cs_i) > len(Ct_j) \\ \dfrac{LCSLen}{len(Cs_i)}, & len(Cs_i) < len(Ct_j) \end{cases}, \quad (8)$$

where, $len()$ is the function that computes the length of a string. The greater the value of $matrix[i][j]$, the similar they are.

3) BSS method: BSS (Big Substring) is similar to LCS. The difference is that for BSS it is to find the biggest substring but not longest subsequence to measure similarity.

$$BSSLen = BSSF(M(k,s).Cs_i, M(k,t).Ct_j), \quad (9)$$

where, $BSSF$ is a function that computes the length of BSS between two tracks that using $BSSLen$ as return value. The function can be solved by KMP [8].

Then, $matrix[i][j]$ can be obtained by $BSSLen$:

$$matrix[i][j] = \begin{cases} \dfrac{BSSLen}{len(Ct_j)}, & len(Cs_i) > len(Ct_j) \\ \dfrac{BSSLen}{len(Cs_i)}, & len(Cs_i) < len(Ct_j) \end{cases}. \quad (10)$$

The greater the value of $matrix[i][j]$, the more similar they are. Because of the strict continuity, the method has a better result than LCS for the pitch timing sequence.

C. Dealing with Special Circumstances

While constructing the similarity matrix, certain special MIDI tracks may influence on the value of $matrix[i][j]$ and precision, so they must be dealt with specially.

1) Symmetric matrix: For different MIDI files of a song, their tracks' information may be exactly the same. In this situation, the similarity matrix got by Step 2 in algorithm MelodyExtraction is a symmetric matrix. Therefore, only one should be considered follow-up.

2) Empty tracks: If $M(k,s).Cs_i$ or $M(k,t).Ct_j$ is a null track, then let $matrix[i][j] = -1$.

3) Bass accompaniment: Normally, in the vector of tracks from a MIDI file, the pitches of melody track are higher; otherwise it may be masked by other tracks so that it can not be heard. Some bass accompaniment tracks are very low and duplicate certain rhythms, so they can be filtered out first. Before the step *Difference* in II.*A*, we can remove bass accompaniment tracks.

Let S denote the sum of all the notes' pitches of a track in the vector, and N denote the number of the notes, so

$$NoteAverage = S/N. \quad (11)$$

If $NoteAverage < \delta$ (δ is a parameter, $0 \le \delta \le 50$), then remove the track form the vector.

4) Harmony accompaniment: In different MIDI files of a song, the harmony accompaniment tracks are also more likely appearing, and by simply calculating the similarity, they may also be regarded as the main track. We can avoid such situation by calculating the variance V.

$$V = \frac{\sum_{i=0}^{N} (C[i] - NoteAverage)^2}{N}. \quad (12)$$

where, $C[]$ represents the string of a track. Let the variance of $M(k,s).Cs_i$ be V_1, and the variance of $M(k,t).Ct_j$ be V_2. If $V_1 > \alpha$ or $V_2 > \alpha$ (α is a parameter, $\alpha \ge 20$), then let $matrix[i][j] = -1$.

5) Too short tracks: Normally, the melody track is longer than other tracks of a MIDI file and the too short tracks must be not the melody track. But sometimes the too short tracks may exist in all MIDI files of a song. While computing the similarity, they may also be regarded as the main track.

Let the average length of all tracks of $M(k,s)$ be $aveLen1$, and the average length of all tracks of $M(k,t)$ be $aveLen2$. If $len(Cs_i) < aveLen1 \times 0.5$ or $len(Ct_j) < aveLen2 \times 0.5$, let $matrix[i][j] = -1$.

D. Identifying Main Track

After Step 2 and 3 in algorithm MelodyExtraction, given m different MIDI files of a song, we can get C_m^2 similarity matrices. Through analyzing these matrices, the main track of each MIDI will be identified.

Let $mainTracks[m][16]$ be an array, the value of each element means the score of the track.

For N-grams, to find η pieces of smallest values from every similarity matrices, for the LCS and BSS, to find η pieces of largest values from the similarity matrices, and at the same time record the number's row and column in the matrix. Finding results are represented as (x, y, v), here x is the row, y is the column and v is the value.

For similarity matrix $M(k,s)$ and $M(k,t)$, while finding a value, $mainTracks[m][16]$ will be updated as:

$$\begin{aligned} mainTracks[s][x] &= mainTracks[s][x] + 1 \\ mainTracks[t][y] &= mainTracks[t][y] + 1. \end{aligned} \quad (13)$$

After the whole finding process for C_m^2 similarity matrices, the main track of $M(k,s)$ is $\min(mainTracks[s])$ in N-grams method; for LCS and BSS method, the main track is $\max(mainTracks[s])$.

III. EXPERIMENTS

In our experiments, we use WebCrawler to download MIDI files from the Internet randomly. In the database, there are 500 MIDI files that every song has five different versions. The database is partitioned into two sets: the developing set and testing set respectively. The developing set contains 100 files, and the others belong to the testing set.

In order to verify the effectiveness and performance of the algorithm, we define the following two parameters:

$$R = Ocd / d, \quad (14)$$

where, R represents the recall of a song, Ocd is the number of the MIDI files that have found their melody correctly in d different MIDI files of a song.

$$P = sum(R_1, R_2, ..., R_n) / n , \quad (15)$$

where, P represents the precision, n is the number of the songs.

In the first experiment, we compute the similarity using N-grams, LCS, and BSS respectively, and the experimental result is shown in Fig.1.

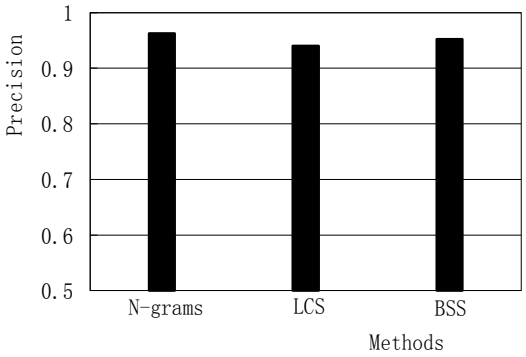
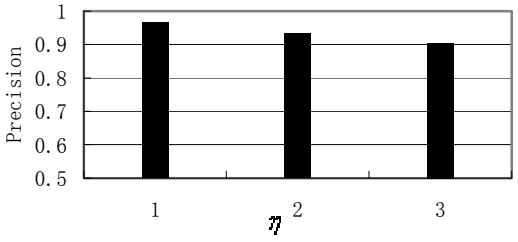


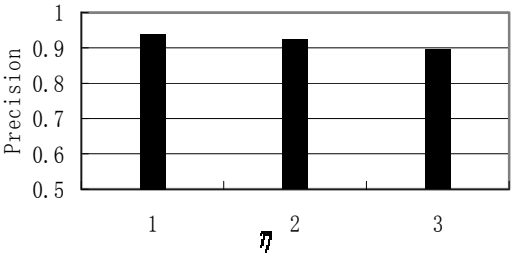
Figure 1. Precision of the algorithm.

We can find that the algorithm can receive high precision and is effective, and N-grams method has the highest precision and better than LCS and BSS.

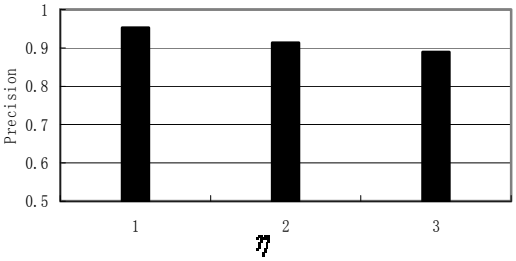
The parameter η will influence on the precision. In the second experiment, we test the influence on the precision of the algorithm by using N-grams, LCS and BSS respectively and the experimental result is shown in Fig.2.



(a) N-grams



(b) LCS



(c) BSS

Figure 2. Influence on the precision.

We find that while the value of η increases from 1 to 3, all the precision of the algorithm decreases using the three similarity computing methods. The reason is that the scores of non-main tracks will increase when η increases.

IV. CONCLUSION

In this paper, the theory of melody similarity is adopted in the automatic melody extraction from polyphonic MIDI. The performance of the proposed algorithm is evaluated on the dataset downloaded from the Internet. Experimental results show that the algorithm can distinguish the melody from the MIDI files automatically and the precision can reach 95% above. We have applied this method in a hamming retrieval system. Compared with the manually labeled method, our method can identify the melody from MIDI files efficiently and automatically. We will study how to apply this algorithm into other types of musical files in future.

REFERENCES

[1] Yuen-Hsien Tseng, "Content-Based Retrieval for Music Collections", Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press, New York, NY, USA, 1999, pp.176-182.

[2] H. Shih, Shrikanth S. Narayanan, "Automatic Main Melody Extraction from MIDI Files with a Modified Lempel-Ziv Algorithm", Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, Hong Kong, 2001, pp 9-12.

[3] Jia-Lien Hsu, Arbee L. P. Chen, C. C. Liu, "Efficient Repeating Pattern Finding in Music Database", Proceedings of the seventh international conference on Information and knowledge management, ACM Press, New York, NY, USA, 1998, pp.281-288.

[4] Zhao Fang, Wu Yadong, "Melody Extraction Method from Polyphonic MIDI Based on Melodic Features", Compute Engineering, Publishing House of Journal of Computer Engineering, Beijing, China, 2007, 33(2), pp.165-167. (In Chinese)

[5] Hofman-Engl, L, "Melodic Similarity and Transformations: A Theoretical and Empirical Approach", PhD Thesis, Department of Psychology, Keele University, UK, 2003.2.

[6] Wagner, R., Fischer, M., "The string-to-string correction problem", Journal of the ACM, ACM Press, New York, NY, USA, 1974, 21(1), pp.168-173.

[7] Doraisamy, "Robust Poly-phonic Music Retrieval with N-grams", Journal of Intelligent Information Systems, Kluwer Academic Publishers Hingham, MA, USA, 21(1), pp.53–70.

[8] Thomas H. Cormen, Charles E.Leiserson, Ronald L.Rivest Clifford Stein, Introduction to algorithms. The MIT Press, USA, 2001.