CS229 Project: Music Search Engine

Shaoqing Xiang
xsq2003@stanford.edu
December, 2008

1. Introduction

With the abundance of digital music files on the internet, how to efficiently and effectively find a music piece is crucial. Conventional music search systems utilize text information of the music, such as the title of a song, a singer's name, or the lyrics of a song. In many cases, such text-based music search tool is not sufficient. Often, a user may remember how to sing part of a song he/she has once heard, and wants to find the complete piece of music on the internet, but not knowing any other information about the music (i.e. tile, composer name), the search can not be done with conventional search tools. Such scenario demands the use of a query-by-humming (QBH) music search system.

A QBH system is a search tool that takes audio file- often a short piece of query melody hummed or played by the user- as the searching input, and by comparing the inputted audio file (query) with the music files (references) in the database, the QBH system finds the corresponding music files, part of which is most similar to the query sound.
Several approaches have been tried for this kind of system in the past, and various comparing algorithms are used, such as dynamic programming [1], edit-distance alignment [2], and linear alignment matching [3].

In this project, I focus on the melody comparing part of the QBH system, and have built a melody comparing engine based on the linear alignment matching algorithm.

2. QBH System composition

An overview of a QBH system is shown in Figure.1.

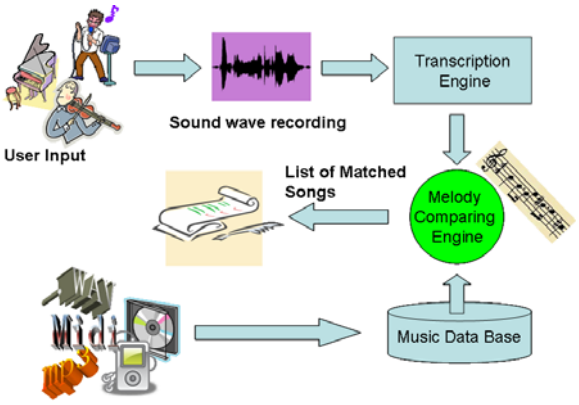

Figure.1 Overview of a QBH System

The system is mainly comprised of two parts, melody transcribing engine and melody comparing engine. The melody transcribing part takes the user query sound wav information (often in the format of .WAV file), divide it into many (often 1024) segments, and do time-frequency domain transformation to get the frequency information of each segment. Then the transcribing engine determines the pitch of sound each segment represents, and finally transcribes the whole piece of query sound waveform to the score or the melody contour it represents (a melody contour is the mathematical representation of a music score series, and it records the sound pitch value at any position of a song).

Several past CS229 projects have worked on transcription of audio files, so in this project, I focus myself on the melody comparing engine.

The task of the melody comparing engine is to find out how similar the melody contour of the query music is to the contour of each music piece in the database, and then returns one or several music pieces that have the highest similarity score in the database.

Several aspects need to be taken into consideration for the melody comparing engine: first, the user is singing part of a song, and we have no prior knowledge about which part of the song it is, in other words, we don't know the exact location in each song to compare with the query; secondly, users may sing in a different tempo (speed) with the original song, so we don't know how much to scale the query in the time space before comparison; thirdly, not most people can remember and sing with the absolute pitch of a song, actually a song is recognized as having the same melody when the pitch of all its notes are transposed with the same amount, therefore the absolute pitch of each note is not so important, only the intervals between notes matters.

In the light of the analyze above, the comparing should not be affected by the absolute pitch value and time scale of the query, and the shape of the query's melody contour is the real 'value' that we want to compare to the various contour shapes in the music files from the database. And the comparing of the query with each reference music file is a learning process that the engine finds out the time scale, position in the reference music, and pitch transposition value that fit the query contour shape best with the reference contour shape. As figure 2 shows.

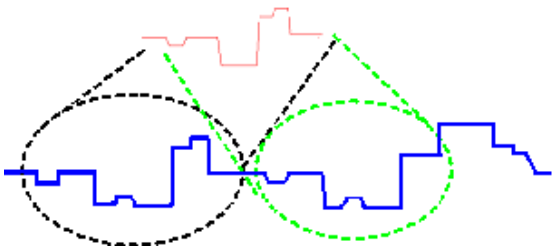

Figure.2 Melody Contour Shape Fitting

Finally, the comparing algorithm should be able to tolerate small pitch interval error and small melody rhythm mismatch, since it is often hard to sing a song with exactly correct rhythm and tune.

3. Comparing Algorithm

Two features determine the shape of the query melody contour: the rhythm and the pitch intervals. My algorithm is based on the Linear Alignment Matching algorithm (LAM) introduced in [4] to separately evaluate the rhythm/pitch similarity of the query and the reference music. I further introduced pitch transposition to ensure a better fit. The specific comparing algorithm between a query and a reference music piece having similar total amount of notes is as following:

The learning process for the best fit first starts with aligning the notes onset time of the two melody contours to get the largest rhythm similarity. The query contour is first scaled to the same time span with the reference contour, and the beginning time of each note in the two contours are compared in series, if the difference is smaller than a dynamic threshold, I consider this as a rhythm match, and rescale the whole query contour to align the two matching notes (under the constraint that all the formerly aligned notes remain aligned after the rescaling). Then, the rhythm score is calculated according to the total alignments found. This alignment algorithm is tolerant to note insertion/deletion and small rhythm mismatch.

After the two contours are aligned with respect to rhythm, a pitch score is calculated according to the pitch difference between them, using a special distance function. The transposition value that corresponds to the least pitch difference is applied to the query.

The total similarity score is the weighted sum of

the rhythm & pitch score. Figure.3 shows several intermediate results during a matching iteration between two contours, where red plot is the query, and the blue plot is the reference melody.

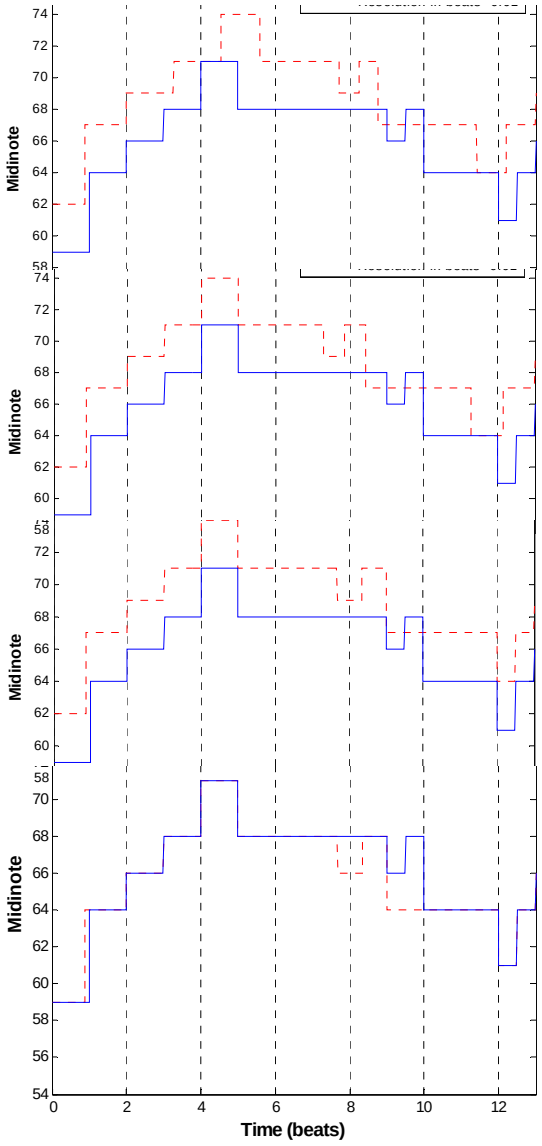

Figure.3 contour matching process

The detailed description of the comparing algorithm can be found in [3]. I introduced pitch transposition to the query contour to further fit the shape of two contours.

The above algorism works for comparing two melody sequences with similar length. However, in most cases, we need to compare a short query melody with a much longer reference melody. This problem is solved as following: all notes sequences that have comparable note amount with the query are excerpted from the long reference melody, and compared with the query separately. The sequence with the highest similarity score is the position in the reference melody that mostly resembles the query melody, and this maximum similarity score represents the similarity between the query and the reference melody.

The computational time for comparing a query of 12 notes and a reference melody of 100 notes is less than 0.5 second.

4. Test Setup

Since I have only built the melody comparing engine, and don't have the music transcription engine, the music files I use as the queries and references need to be in a format that is easy to retrieve music scores from. MIDI file is a good option for this requirement, because unlike most other formats (WAV, mp3, etc) which store the waveform of music, MIDI directly stores pitch value and onset time of the notes. I use Matlab MIDI Toolbox version 1.0 [4] to read MIDI file into Matlab.

The reference music database I use is composed of 100 mono-channel MIDI files downloaded from the Themefinder music archive [5] and from my own collections. The genres of the music pieces are mainly classic and pop.

The query melodies is generated by me using composing software called 'Composer Master 2008', it transfers music scores into a hearable MIDI file. I compose the query melodies after hearing and remembering some part of the reference music pieces.

10 queries are made, because I am not an expert in music, inevitably there is un-intended error in

the queries such as rhythm/pitch variation, tempo change and music note transpose, etc. This simulates the real scenario in which the user generates the query music by humming, and there is likely to be variation of the melody he/she hums from the original piece. And the music transcribing engine is also likely to introduce error. In the 10 queries I have made, one is intentionally made out of tune, and another one has a very different rhythm with the original piece. This is to test how my melody comparing engine deals with largely distorted query input.

5. Test Results and Discussion

For 7/10 of the queries, the comparing engine retrieves the correct songs in the database as the most similar to the queries. For 8/10 queries, the correct song is found among top 3 matches retrieved by the comparing engine. Figure.4 shows the retrieving success rate VS. the number of top matches allowed to display.

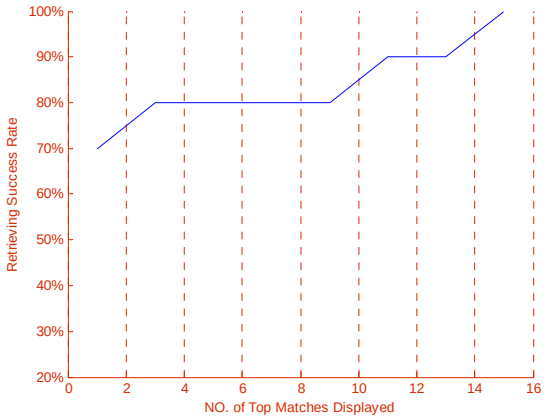

Figure.4 retrieving success rate VS. NO. of top matches displayed

Most queries can find their correct matches within 3 songs that have the top similarity score. The query which is intentionally made out of tune find its correct match within 9 top search results, and the query which is intentionally made poor in rhythm is the one that fit worst with the database, the correct match is among the top 15 search results. This shows that rhythm similarity counts more than pitch similarity. It can be explained by the algorithm I use. The comparing process first try to align the two melodies in rhythm, if the two melody have a large difference in rhythm, then the notes of the two melodies can not be correctly aligned, so even if the query is perfect in pitch, my later pitch score calculation will still return a low value, because the notes are not aligned correctly. This gives rise to problems, since normally a song with correct pitch but poor rhythm is considered to be closer to the original piece than a song with correct rhythm but poor pitch. In the future, the comparing need to be improved that the rhythm matching and pitch matching are no longer coupled together.

6. Conclusion

In this project I built a melody comparing engine based on the LAM algorithm. The engine is able to retrieve the correct song for 80% of the queries among the 3 top matches. Future work needs to be done to decouple the rhythm matching and pitch matching process.

References

[1] Sung-Phil Heo, Motoyuki Suzuki, Akinori Ito, Shozo Makino, and Hyun-Yeo Chung, '*Error Tolerant Melody Matching Method in Music Information Retrieval*', A. Nürnberger and M. Detyniecki (Eds.): AMR 2003, LNCS 3094, pp. 212–227, 2004.

[2] Thomas Gersic, '*Music Melody Matching Machine*', http://www.gersic.com

[3] Ya-Dong Wu, Yang Li, Bao-Long Liu, '*A New Method For Approximate Melody Matching*', Proceedings of the Second International Conference on Madune Learning and Cybernetics, Wan, 2-5 November 2003
[4] MIDI Toolbox version 1.0, http://www.jyu.fi/musica/miditoolbox/
[5] MIDI file archive from Themefinder, http://composer.themefinder.org/