A Query-by-Singing System based on Dynamic Programming[1]

Jyh-Shing Roger Jang, Ming-Yang Gao
{jang, gao}@cs.nthu.edu.tw

Computer Science Department, National Tsing Hua University, Hsinchu, Taiwan

Keywords

Content-based Music Retrieval, Audio Indexing and Retrieval, Dynamic Programming, Dynamic Time Warping, Audio Signal Processing, Query by Singing

Abstract

This paper presents a query-by-singing system of which the comparison engine is mostly based on the concept of dynamic programming (DP). The system, known as CBMR (Content-Based Music Retrieval), facilitates the content-based song database retrieval via users' acoustic inputs. CBMR first takes the user's acoustic input from a microphone and converts it into a pitch vector. Then two levels of comparison procedures, both based on the concept of dynamic programming, are invoked to compute the similarity between the user's pitch vector and that of each song in the database. CBMR then shows a ranked result according to the computed similarity scores. We have tested CBMR extensively and found the performance is satisfactory for average people with mediocre singing capability. Our studies demonstrate the importance of using dynamic programming for elastic matching in multimedia informational retrieval.

1. Introduction

This paper presents a query-by-singing system that is based on two levels of dynamic programming as its comparison engine. The system facilitates the content-based song database retrieval via users' acoustic inputs, which means that the system's friendly interface allows the users to retrieve songs based on a few notes sung or hummed naturally to the microphone. Therefore the traditional way of song retrieval (particularly in the applications of karaoke or digital music library) via the search of keywords of titles, singers or lyrics can be avoided.

The tasks of the system can be categorized into three stages: preprocess, on-line process, and postprocess. For the preprocess stage, we need to read each song and put the melody/beat information into several indexed files for quick access. Usually the songs in the database are in MIDI (Music Instrument Digital Interface) format, which contains all the music elements of a song and it is equivalent to sheet music. Before extracting the pitch/beat information, we also need to identify the major track of the MIDI file. The major track is more or less equivalent to the vocal track of a MIDI file, which can be defined as the track that, when hearing it, human can identify the intended song immediately.

During the on-line process stage, the user can specify a query by singing or humming a piece of tune (that contains several notes for identifying a song) to a PC microphone directly. Several time-domain-based methods, such as autocorrelation or average magnitude difference function, are used to find the most likely pitch, and some heuristics (such as continuity and energy levels) are employed to eliminate unwanted pitches which might result from either unvoiced segments of the acoustic input or the undesirable effect of pitch doubling.

At the postprocess stage, the computed pitch vector together with related timing information are transformed into the same middle representation that was used in encoding the pitch/beat information of songs in the database. Before invoking similarity comparison, the middle representation must be transformed into a format that does not rely on the absolute values of the identified pitches. This is usually done by the difference operator, as reported by previous work [1,7,8,9,10]. However, we found that the difference operator amplifies noises and leads to poor performance. Thus we adopt a heuristic method to shift the input pitch vector to have the same average value as that of each song's pitch in the database. Moreover, we can apply key transposition to find a better shifted pitch vector for a better match in dynamic time warping. The system then returns a ranked list according to similarity scores.

The preprocess requires human interaction to identify the major track for extracting the pitch/beat information from a song in MIDI format. Once the major track is

[1] This is an on-going project supported by Cweb Inc. (http://www.4music.com.tw) and the Excellency Project sponsored by the National Science Council at Taiwan, where the authors express many thanks. (This paper is published in the International Workshop on Intelligent Systems Resolutions (the 8th Bellman Continuum), PP. 85-89, Hsinchu, Taiwan, Dec 2000.)

specified, the pitch/beat transcription and the similarity score computation are totally automatic. The average response time of our system (with about 800 songs) takes about 3 seconds on a Pentium III 800 when the user specifies a query from the start of a song. The performance is satisfactory, as long as the users can sing or hum the intended song with correct relative pitches.

The response time could take much longer if the user specifies a query from anywhere in the middle of a song. Moreover, the 800 songs in the system could not nearly match the number of songs in a real-world application, such as a typical popular karaoke bar, which can host up to 10,000 songs. Therefore, we are adding more songs to the database to match the status of a real-world application. More importantly, with more and more songs in the database, we are investigating tree-based search techniques to shorten the response time significantly without sacrificing the performance.

2. Related Work

Content-based multimedia information retrieval is a popular and useful research topic, but most research is focused on the media of text, image and video [6]. Even there are several publications on audio retrieval [2], the research on music retrieval based on acoustic inputs are not common. However, as the digital music is all over the internet, the capability to be able to search/retrieve digital music files are getting more and more important.

Ghias et al. [1] published one of the early papers on query by singing. They applied autocorrelation to obtain the fundamental frequency, and the pitch vector is then cut into notes. To accommodate the problem of different starting key, the obtained notes are converted into ternary contour of three characters: U (up, meaning this note is higher than the previous one), R (repeat, meaning this note is the same as the previous one), D (down, meaning this note is lower than the previous one). The comparison engine, based on longest common subsequence, is then applied to the ternary contours to find the most likely song. However, due to the limited computing power at that time, their pitch tracking takes 20-45 seconds, and there were only 183 songs in the database.

R. J. McNab et al. [7,8,9,10], in collaboration with New Zealand Digital Library, have published several papers on their experiment of query by singing. They applied Gold-Rabiner algorithm [13] for pitch tracking, and the pitch vector was then cut into notes based on energy levels and transition amounts. There were about 9400 songs in their database and they are the first one to put their system on the web. Their system, though lack of performance evaluation in terms of recognition rate, is still considered an excellent example of content-based music retrieval for real-world applications.

3. CBMR System

In this section. we shall explain the operations of CBMR. In particular, we will focus on the dynamic-programming-based comparison procedure, which is the most crucial factor in determining the performance of the system.

3.1. Input Collection

The acoustic input (which could be singing, humming, or music instrument playing) is recorded from a PC microphone directly with a length of 8 seconds, sample rate of 11025, 8 bit resolution and single channel (mono). The system can also reads the acoustic input from a WAV file directly. Before further processing, the acoustic input have to go through a low-pass filter at 1047 Hz in order to cut down unwanted high-frequency components.

3.2. Pitch Tracking

Once a clean acoustic input of 8 seconds is obtained, we need to do **pitch tracking** in order to identify the pitch frequency with respect to time. The acoustic input is first put into frames of 512 points, with 340 points of overlap; this corresponds to 1/64 second for each pitch frequency. Then every 4 pitch frequencies are averaged to merge into a single frequency, thus the final pitch vector has a time scale of 1/16 second.

There are plenty methods for pitch tracking [1] in the literature. For our system, we have implemented pitch tracking using two methods:

1. Autocorrelation function [4]
2. Average magnitude difference function (AMDF) [4]

Both methods are performed in time domain and have comparable performance. Moreover, for software implementation on current Pentium-based PC, both methods are fast enough for real-time pitch tracking. However, when considering chip/hardware implementation, AMDF does not require multiplication and is thus less computation-intensive.

After obtaining the pitch frequencies, we use the following formula to transform them into the representation of **semitone**:

$$semitone = 12 * \log_2\left(\frac{freq}{440}\right) + 69$$

The semitone representation here is equivalent to the one used by MIDI format, where 69 represents central LA (A440, 440 Hz). Subsequent smoothing and comparison operations are based on semitones.

3.3. Pitch Smoothing

The pitch contour obtained in the previous step may not be smooth enough due to the following reasons:

1. Unvoiced segments and random noise can cause unreasonably high pitch.
2. A strong second harmonics can produce pitch doubling effect.

To eliminate these undesirable pitches, the system has a simple end-point detection based on energy levels. If the energy level is lower than a threshold, then the corresponding pitch semitones are set to zero. Also if the identified pitch semitones are higher than 84 (or 1047 Hz in frequency), they are also set to zero. Moreover, to ensure overall smoothness, we put the pitch contour through a medium filter of size 5.

3.4. Dynamic Time Warping and Key Transposition

For elastic match, we can simply use dynamic time warping (DTW) [4] to compute the distance between the input pitch vector and that of each songs in the database. Suppose that the input pitch vector (or test vector) is represented by $t(i), i = 1, \ldots, m,$ and the reference pitch vector (reference vector) by $r(j), j = 1, \ldots, n$. These two vectors are not necessarily of the same length and we can apply dynamic time warping to match each point of the test vector to that of the reference vector in an optimal way. That is, we want to construct a $m \times n$ DTW table $D(i, j)$ according to the following forward dynamic programming algorithm:

Optimal value function:

$D(i, j)$ is the minimum distance starting from the left-most side ($i = 1$) of the DTW table to the current position (i, j).

Recurrence relation:

$$D(i, j) = d(i, j) + \min \begin{cases} D(i-2, j-1) \\ D(i-1, j-1) \\ D(i-1, j-2) \end{cases}$$

where $d(i, j)$ is the node cost associated with $t(i)$ and $r(j)$ and can be defined as

$$d(i, j) = |t(i) - r(j)| - \eta$$

In the preceding formula, η is a small number and its function is to bias the search toward a longer matched path instead of a shorter one. Moreover, the recursion shows that the optimal path can allow the test input to be within half to twice the size of the reference vector. This constrain is reasonable since it is rather difficult to sing/hum a song more than twice or below half of the speed of the original song.

Boundary conditions:

The boundary conditions for the above recursion can be expressed as

$$D(i,1) = \infty, i = 2, \ldots, m$$
$$D(1, j) = |t(1) - r(j)|, j = 1, \ldots, n$$

The first equation ensures that the optimal DTW path never starts from the middle of the test vector. The second equation indicates that the optimal DTW path can start from anywhere in the middle of the reference vector. These conditions are reasonable since we would like to match the whole test vector to a part of the reference vector.

The cost of the optimal DTW path is defined as

$$\min_j D(m, j)$$

After finding the optimizing j, we can back track to obtain the entire optimal DTW path.

In practice, m = 16*8 = 128 since we usually have 8 seconds of recording; For this study, we assume the user always sing/hum from the start of a song, therefore we only need to consider the test vector as the beginning part of each song. We assume that the reference vector cannot be longer than 1.4 times the size of the test vector, hence n = 16*8*1.4 = 179. Thus we need to construct a DTW table of size up to 128×179. The size of the DTW table could be smaller since usually we discard silence (or rests) of both the test and reference semitones. (If we want the match to start from anywhere in the middle of a song, then for a typical song of 3 minutes, n=16*60*3=2880. Hence the DTW table is huge with size 128×2880. It is doable but it is out of the scope of this paper.)

Besides constructing the DTW table for computing each similarity scores, we still need to deal with the problem of different keys for different users. For instance, a

female singer usually has a higher overall key than a man. To deal with the problem, first we shift the test vector to the same mean as that of the part of the reference vector having the same length as the test vector. Moreover, to guarantee better performance, we can optionally take a heuristic approach that shifts the entire input pitch vector to a suitable position that can generate the minimum DTW distance. The method for this **key transposition** procedure can be described next:

1. Set initial parameters and make t and r zero mean:

$$\begin{cases} span = 4 \\ center = 0 \\ t = t - mean(t) \\ r = r - mean(r) \end{cases}$$

(The last two equations make both t and r zero mean.)

2. Compute the DTW distances:

$$\begin{cases} s_{-1} = dtw(r, t - span) \\ s_0 = dtw(r, t) \\ s_1 = dtw(r, t + span) \end{cases}$$

3. Find the minimum DTW distance and update $center$:

If $s_{-1} = \min\{s_{-1}, s_0, s_1\}$, then

center = center – span

else if $s_1 = \min\{s_{-1}, s_0, s_1\}$, then

center = center + span

4. Update $span$ and check stopping condition:

If $span > 2$, $span = span / 2$, go to step 3.

Otherwise stop the iteration.

In our most complicated setting, the key transposition operation involves 5 DTW distance computation. In other words, the computation of comparing an acoustic input with 800 songs in the database involves computing 4000 DTW tables, each with size of around 128×179.

To avoid the problem of different keys for different singer, most related work applied difference operator before invoking DTW or other similarity measure. However, difference operator amplifies noises and does not lead to acceptable performance in our simulation.

4. Performance Evaluation

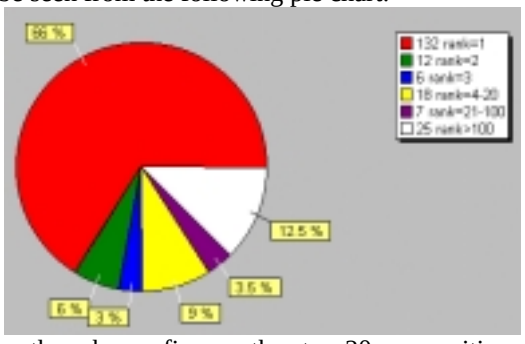
In this section we present the performance evaluation of our CBMR system. We have around 200 recorded clips of songs sung or hummed by 14 persons (9 males, 5 females), Each recording takes from 5 to 8 seconds. The recording conditions are: the sample rate is 11025, 8 bit resolution, single channel (mono). All of the recordings start from the beginning of a song. Some of the recordings include talking and laughing in order to test the robustness of the system. And there are about 800 songs in the database.

We divide the performance evaluation into two parts: one with the use of a single DTW for computing each similarity score, the other with the use of five DTWs to include key transposition. Since we are concerned with both correct recognition rate and computation time, we will list these data extensively in the following discussion. Also we always assume the match always starts at the beginning of a song.

We performed all the tests on a PC with a CPU of Pentium-III 800MHz, 256 MB of RAM.

4.1. Test of 1-DTW

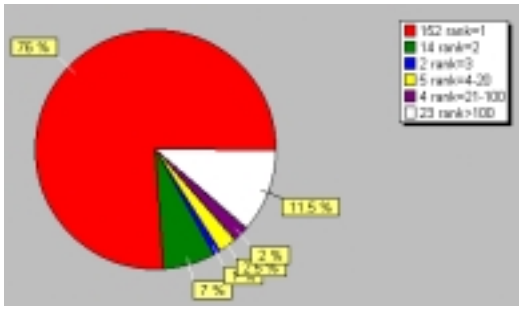
Without key transposition, the computation of each similarity score requires only one application of DTW. Therefore the computation time is shorter but the performance is not as good as the 5-DTW test to be explained next. The average response time for each recording is about 1.557 seconds, and the performance can be seen from the following pie chart:



From the above figure, the top-20 recognition rate (percentage of recordings that CBMR can find the correct intended songs in the top-20 ranking) is 84%, the top-3 recognition rate is 75%, and the top-1 recognition rate is 66%.

4.2. Test of 5-DTW

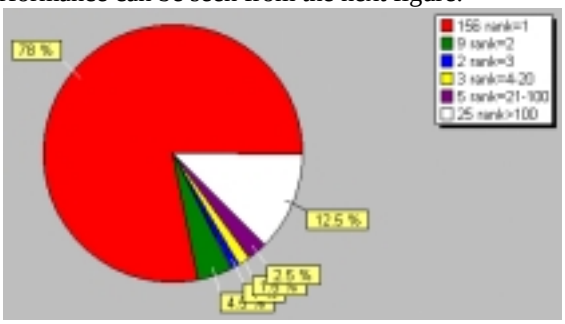
With the use of key transposition, the computation of each similarity score requires the use of 5 DTW. The performance can be seen from the following pie chart:



From the above figure, the top-20 recognition rate is 86.5%, the top-3 recognition rate is 84%, and the top-1 recognition rate is 76%. The average response time for each recording is about 2.556 seconds. As expected the computation is longer, but the performance is better.

4.2. Test of Combining 1-DTW and 5-DTW

An intuitive idea to improve the system is to take a hierarchical approach that uses 1-DTW to filter out 700 of the 800 songs, and leave only 100 songs for 5-DTW to do a detailed comparison. Presumably this hierarchical approach will shorten the response time without sacrificing the performance to much. The response time is about 3 seconds for each recording, and the performance can be seen from the next figure:



From the above figure, the top-20 recognition rate remains 85%, the top-3 recognition rate is 83.5%, and the top-1 recognition rate is still 78%. In other words, the performance is almost the same as that of 5-DTW, but the response time have been effectively reduced from 2.556 to 1.765 seconds.

5. Conclusions and Future Work

In this paper, a hierarchical approach for combining DTW-based comparison engines is proposed and used in a CBMR system. The performance and response time of the simulation demonstrate the feasibility of the system in operation. Future work includes the following items:

1. Note segmentation for quick performance evaluation based on note-level comparison.
2. Better smoothing techniques to enable the use of difference operator.
3. Other speedup schemes to allow the match start from anywhere in the middle of a song.

References

[1] A. Ghias, J. Logan, D. Chamberlain, B. C. Smith, “Query by humming-musical information retrieval in an audio database”, ACM Multimedia ’95 San Francisco, 1995. (http://www2.cs.cornell.edu/zeno/Papers/humming/humming.html)

[2] J. Foote, "An Overview of Audio Information Retrieval," In Multimedia Systems, vol. 7 no. 1, pp. 2-11, ACM Press/Springer-Verlag, January 1999.

[3] C. C. Liu and A. L. P. Chen, “A Multimedia Database System Supporting Content-Based Retrieval”, *Journal of Information Science and Engineering*, 13 ,369-398,1997.

[4] J. R. Deller, J. G. Proakis, J. H. L. Hansen, "Discrete-time processing of speech signals," New York :Macmillan Pub. Co. , 1993.

[5] J. Brown and B. Zhang, “Musical frequency tracking using the methods of conventional and ‘narrowed’ autocorrelation”. *Journal of the Acoustical Society of America*, Volume 89, Number 5, pages 2346-2354, 1991.

[6] M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, “Query by image and video content: the QBIC system,” IEEE Computers, Vol. 28, No. 9, pp.23-32, 1995.

[7] R. J. McNab, L. A. Smith, Jan H. Witten, “Towards the Digital Music Library: Tune Retrieval from Acoustic Input” ACM, 1996.

[8] R. J. McNab, L. A. Smith, Jan H. Witten, “Signal Processing for Melody Transcription” *Proceedings of the 19th Australasian Computer Science Conference*, 1996.

[9] R. J. McNab, L. A. Smith, “Melody transcription for interactive applications” *Department of Computer Science University of Waikato*, New Zealand.

[10] R. J. McNab, L. A. Smith, I. H. Witten and C. L. Henderson, "Tune Retrieval in the Multimedia Library,"

[11] N. Kosugi, Y. Nishihara, S. Kon'ya, M. Yamamura, and K. Kushima, "Music Retrieval by Humming – Using Similarity Retrieval over High Dimensional Feature Vector Space," pp 404-407, IEEE 1999.

[12] A. Uitdenbogerd and J. Zobel, "Melodic Matching Techniques for Large Music Databases", (http://www.kom.e-technik.tu-darmstadt.de/acmmm99/ep/uitdenbogerd/)

[13] B. Gold and L. Rabiner, "Parallel processing techniques for estimating pitch periods of speech in the time domain," J. Acoust. Soc. Am. 46 (2), pp 442-448, 1969.