Music Database Retrieval Based on Spectral Similarity

Cheng Yang*
Department of Computer Science
Stanford University
yangc@cs.stanford.edu

Abstract

We present an efficient algorithm to retrieve similar music pieces from an audio database. The algorithm tries to capture the intuitive notion of similarity perceived by human: two pieces are similar if they are fully or partially based on the same score, even if they are performed by different people or at different speed.

Each audio file is preprocessed to identify local peaks in signal power. A spectral vector is extracted near each peak, and a list of such spectral vectors forms our intermediate representation of a music piece. A database of such intermediate representations is constructed, and two pieces are matched against each other based on a specially-defined distance function. Matching results are then filtered according to some linearity criteria to select the best result to a user query.

1 Introduction

With the explosive amount of music data available on the internet in recent years, there has been much interest in developing new ways to search and retrieve such data effectively. Most on-line music databases today, such as Napster and mp3.com, rely on file names or text labels to do searching and indexing, using traditional text searching techniques. Although this approach has proven to be useful and widely accepted, it would be nice to have more sophisticated search capabilities, namely, searching by content. Potential applications include "intelligent" music retrieval systems, music identification, plagiarism detection, etc. Traditional techniques used in text searching do not easily carry over to the music domain, and people have built a number of special-purpose systems for content-based music retrieval.

*Supported by a Leonard J. Shustek Fellowship, part of the Stanford Graduate Fellowship program, and NSF Grant IIS-9811904.

Music can be represented in computers in two different ways. One way is based on musical scores, with one entry per note, keeping track of the pitch, duration (start time / end time), strength, etc, for each note. Examples of this representation include MIDI and Humdrum, with MIDI being the most popular format. Another way is based on acoustic signals, recording the audio intensity as a function of time, sampled at a certain frequency, often compressed to save space. Examples of this representation include .wav, .au, and MP3.

A simple software or hardware synthesizer can convert MIDI-style data into audio signals, to be played back for human listeners. However, there is no known algorithm to do reliable conversion in the other direction. For decades people have been trying to design automatic transcription systems that extract musical scores from raw audio recordings, but have only succeeded in monophonic and very simple polyphonic cases [1, 3, 9], not in general polyphonic case[1]. In Section 3.1 we will explain briefly why it is a difficult task to do automatic transcription on general polyphonic music.

Score-based representations such as MIDI and Humdrum are much more structured and easier to handle than raw audio data. On the other hand, they have limited expressive power and are not as rich as what people would like to hear in music recordings. Therefore, only a small fraction of music data on the internet is represented in score-based formats; most music data is found in various raw audio formats.

Most content-based music retrieval systems operate on score-based databases, with input methods ranging from note sequences to melody contours to user-hummed tunes [2, 5, 6]. Relatively few systems are for raw audio databases. A brief review of related work will be given in Section 2. Our work focuses on raw audio databases; both the underlying database and the user query are given in .wav audio format. We develop algorithms to search for music pieces similar to the user query. Similarity is based on the intuitive notion of similarity perceived by humans: two pieces are similar if

[1] Polyphony refers to the scenario where multiple notes occur at the same time, possibly by different instruments or vocal sounds. As we know, most music pieces are polyphonic.

they are fully or partially based on the same score, even if they are performed by different people or at different tempo.

In the next section we will discuss some previous work in this area. In Section 3 we will start with some background information and then give a detailed presentation of our algorithm to detect music similarity. Section 4 gives experimental results, and future directions will be discussed in Section 5.

2 Related Work

Examples of score-based database (MIDI or Humdrum) retrieval systems include the ThemeFinder project (http://www.themefinder.org) developed at Stanford University, where users can query its Humdrum database by entering pitch sequences, pitch intervals, scale degrees or contours (up, down, etc). The "Query-By-Humming" system [5] at Cornell University takes a user-hummed tune as input, converts it to contour sequences, and matches it against its MIDI database. Human-hummed tunes are monophonic melodies and can be automatically transcribed into pitches with reasonable accuracy, and melody contour information is generally sufficient for retrieval purposes [2, 5, 6].

Among music retrieval research conducted on raw audio databases, Scheirer [7, 8] studied pitch and rhythmic analysis, segmentation, as well as music similarity estimation at a high level such as genre classification. Tzanetakis and Cook [10] built tools to distinguish speech from music, and to do segmentation and simple retrieval tasks. Wold *et al*. at Muscle Fish LLC [11] developed audio retrieval methods for a wider range of sounds besides music, based on analyses of sound signals' statistical properties such as loudness, pitch, brightness, bandwidth, etc. Recently, *CD (http://www.starcd.com) commercialized a music identification system that can identify songs played on radio stations by analyzing each recording's audio properties.

Foote [4] experimented with music similarity detection by matching power and spectrogram values over time using a dynamic programming method. He defined a cost model for matching two pieces point-by-point, with a penalty added for non-matching points. Lower cost means a closer match in the retrieval result. Test results on a small test corpus indicated that the method is feasible for detecting similarity in orchestral music. Part of our algorithm makes use of a similar idea, but with two important differences: we focus on spectrogram values near power peaks only, rather than over the entire time period, therefore making tempo changes more transparent; furthermore, we evaluate final matching results by some linearity criteria which is more intuitive and robust than the cost models used for dynamic programming.

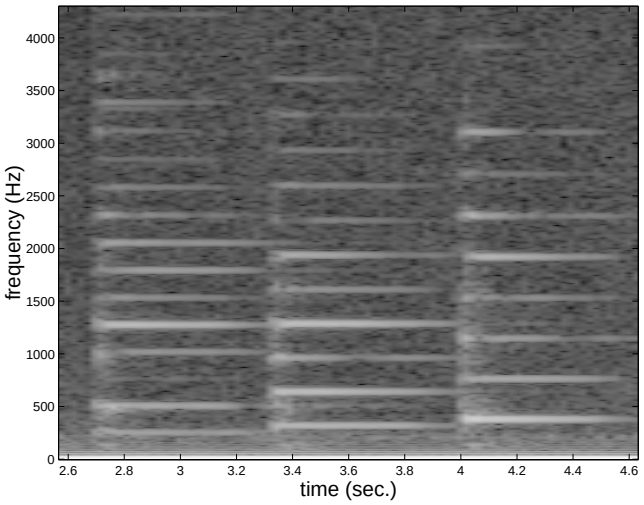

Figure 1. Spectrogram of piano notes C, E, G

3 Detecting Similarity

In this section we start with some background information on signal processing techniques and musical signal properties, then give a detailed discussion of our algorithm.

3.1 Background

After decompression and parsing, each raw audio file can be regarded as a list of signal intensity values, sampled at a specific frequency. CD-quality stereo recordings have two channels, each sampled at 44.1kHz, with each sample represented as a 16-bit integer. In our experiments we use single-channel recordings of a lower quality, sampled at 22.05kHz, with each sample represented as an 8-bit integer. Therefore, a 60-second uncompressed sound clip takes $22050 \times 60 = 1,323,000$ bytes.

We use the Short-Time Fourier Transform (STFT) to convert each signal into a spectrogram: split each signal into 1024-byte-long segments with 50% overlap, window each segment with a Hanning window and perform 2048-byte zero-padded FFT on each windowed segment. Taking absolute values (magnitudes) of the FFT result, we obtain a spectrogram giving localized spectral content as a function of time. Since the details of this process are covered in most signal processing textbooks, we will not discuss them here.

Figure 1 shows a sample spectrogram on the note sequence of middle C, E and G played on a piano. The horizontal axis is time in seconds, and the vertical axis is frequency component in Hz. Lighter pixels correspond to higher values. If we zoom in to time $4.1s$ and look at the frequency components of note G closely, we notice that it has many peaks (Figure 2), one at 392 Hz (its *fundamental frequency*) and several others at integer multiples of 392 Hz

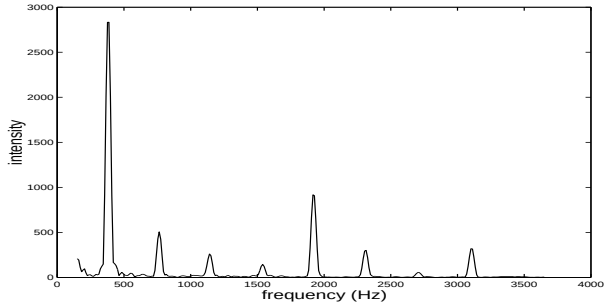

Figure 2. Frequency components of note G played by a piano

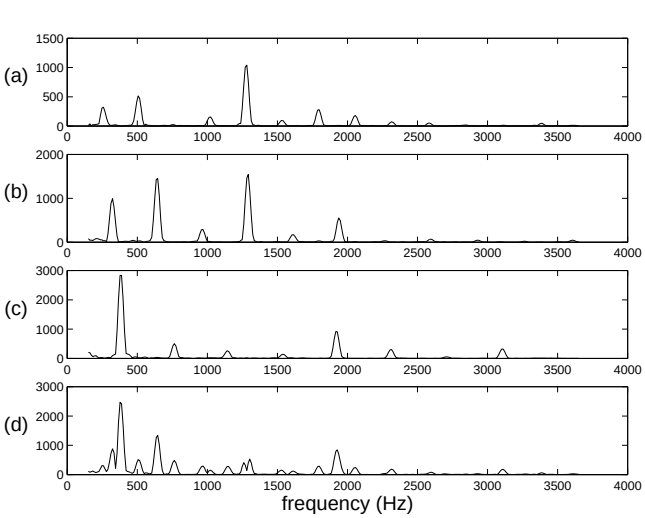

Figure 3. Illustration of polyphony

(its *harmonics*). Fundamental frequency corresponds to the pitch (middle G in this case), and the pattern of harmonics depends on the characteristics of the musical instrument that plays it.

When multiple notes occur at the same time ("*polyphony*"), their frequency components add. Figure 3(a)-(c) show the frequency components of C, E and G played individually, while Figure 3(d) shows that of all three notes played together. In this simple example it is still possible to design algorithms to extract individual pitches from the chord signal C-E-G, but in actual music recordings, many more notes co-exist, played by many different instruments, of which we do not know the patterns of harmonics. In addition, there are sounds produced by percussion instruments, human voice, and noise. The task of automatic transcription of music from arbitrary audio data (i.e., conversion from raw audio format into MIDI) becomes extremely difficult, and remains unsolved today. Our algorithm, as in most other music retrieval systems, does not attempt to do transcription.

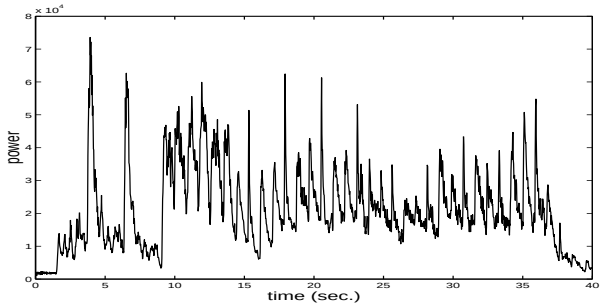

Figure 4. Power plot of Tchaikovsky's Piano Concerto No. 1

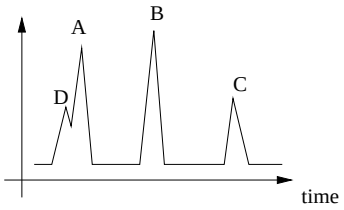

Figure 5. True peak vs. bogus peak

3.2 The Algorithm

The algorithm consists of three components, which are discussed separately.

1. Intermediate Data Generation.

   For each music piece, we generate its spectrogram as discussed in Section 3.1, and plot its instantaneous power as a function of time. Figure 4 shows such a power plot for a 40-second sound clip of Tchaikovsky's Piano Concerto No. 1. Next, we identify peaks in this power plot, where peak is defined as a local maximum value within a neighborhood of a fixed size. This definition helps remove bogus local "peaks" which are immediately followed or preceded by higher values. For example, in Figure 5, A, B, C are true peaks but D is a bogus peak. Intuitively, these peaks roughly correspond to distinctive notes or rhythmic patterns. For the 60-second music clips used in our experiments, we typically find 100-200 peaks in each of them.

   After a list of peaks is obtained, we extract the frequency components near each peak. We take 180 samples of frequency components between 200Hz and 2000Hz. Average values over a short time period following the peak are used in order to reduce sensitivity to noise and to avoid the "attack" portions produced by certain instruments (short, non-harmonic signal segments at the onset of each note).

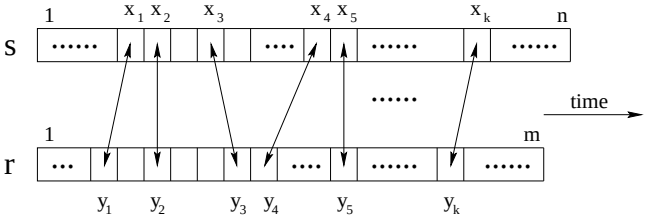

Figure 6. Set of matching pairs

In the end, we get n spectral vectors of 180 dimensions each, where n is the number of peaks obtained. We normalize each spectral vector so that they each have mean 0 and variance 1. After normalization, these n vectors form our intermediate representation of the corresponding music piece. Typically each new note in a piece corresponds to a new peak, and therefore to a vector in this representation. Notice that we do not expect to capture all new notes in this way, and will almost certainly have some false positives and false negatives. However, later stages of the algorithm will compensate for this inaccuracy.

2. Matching.

   This component matches two music pieces against each other and determines how close they are, based on the intermediate representation generated above. Matching comes in two stages: minimum-distance matching and linearity filtering.

   (a) Minimum-distance matching

   Suppose we would like to compare two music pieces with spectral vectors $s_1, s_2, \ldots, s_n$ and $r_1, r_2, \ldots, r_m$ respectively. Define e_{ij} to be root-mean-squared error between vectors s_i and r_j. It can be shown that e_{ij} is linearly related to the correlation coefficient of the original spectra near peak i of the first piece and peak j of the second one. A smaller e_{ij} value corresponds to a larger correlation coefficient. (See [13] for proof.) Therefore, e_{ij} is a natural indicator of similarity of the original spectra at corresponding peaks.

   Let $M_k = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$ be a set of k matches, pairing s_{x_1} with r_{y_1}, s_{x_2} with r_{y_2}, etc, as shown in Figure 6. ($1 \leq x_1 < x_2 < \cdots < x_k \leq n$, $1 \leq y_1 < y_2 < \cdots < y_k \leq m$.)

   Given the following subsets of s and r vectors: $S^a = \{s_1, s_2, \ldots, s_a\}$, $R^b = \{r_1, r_2, \ldots, r_b\}$, and a particular match M_k ($k \leq a \leq n, k \leq b \leq m$), define the distance of S^a and R^b with respect to M_k as:

$$D_{a,b,M_k} = (\sum_{i=1}^{k} e_{x_i y_i}) + \beta(a + b - 2k)$$

   and the minimum distance between S^a and R^b as:

$$D_{a,b} = \min_{M} D_{a,b,M}$$

   The distance definition is basically a sum of all matching errors plus a penalty term for the number of non-matching points (weighted by β). Experiments have shown that $\beta = 0.5$ works reasonably well.

   The minimum distance D_{ab} can be found by a dynamic programming approach, because

$$D_{0,i} = D_{i,0} = \beta i$$

   and for any $i > 0, j > 0$,

$$\begin{aligned} D_{i,j} \quad = \min \quad & \{D_{i-1,j-1} + e_{ij}, D_{i-1,j-1} + 2\beta, \\ & D_{i,j-1} + \beta, D_{i-1,j} + \beta\} \end{aligned}$$

   The optimal matching set M_k that leads to the minimum distance can also be traced from the dynamic programming algorithm.

   Based on the definitions above, the minimum distance between the two music pieces with spectral vectors $s_1, s_2, \ldots, s_n$ and $r_1, r_2, \ldots, r_m$ is $D_{n,m}$, and can be found with dynamic programming.

   (b) Linearity filtering

   Although the previous step gives the minimum distance and optimal matching based on the distance function, it is not robust enough for music comparison. Experiments have shown that certain subjectively dissimilar pieces may also end up with a small distance score, therefore appearing similar to the system. To make the algorithm more robust, further filtering is needed.

   Figure 7 shows two ways to match s against r, both with 10 matches. Both may yield a low matching score, but the top one is obviously better than the bottom one. In the top one, there is a slight tempo change between the two pieces, but the change is uniform in time. In the bottom one, however, there is no plausible explanation for the twisted matching. If we plot a 2-D graph of the matching points of s on the horizontal axis vs. the corresponding points of r on the vertical axis, the top match would give a straight line while the bottom one would not.

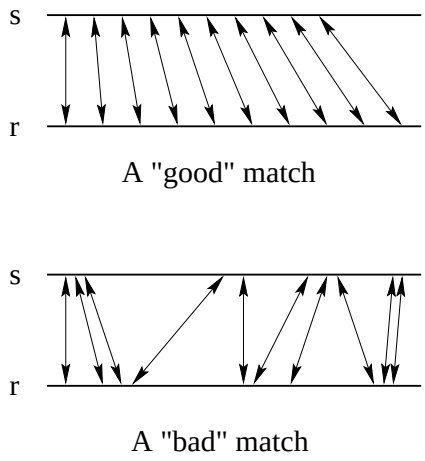

Figure 7. "Good" vs. "bad" matching

Formally, the matching set

$$M_k = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$$

can be plotted on a 2-D graph, with the original location (time offset) of peaks $x_1, x_2, \ldots, x_k$ (of the first music piece) on the horizontal axis and that of peaks $y_1, y_2, \ldots, y_k$ (of the second piece) on the vertical axis. If the two pieces were indeed mostly based on the same score, the plotted points should fall roughly on a straight line. Without tempo change, the line should be at a 45-degree angle. With possible tempo change, the line may be at a different angle, but it should still be straight.

In this step of linearity filtering, we examine the graph of the optimal matching set obtained from dynamic programming above, fit a straight line through the points (using least mean-square criteria), and check if any points fall too far away from the line. If so, remove the most outlying point and fit a new line through the remaining points. Repeat the process until all remaining points lie within a small neighborhood of the fitted line. (In the worst case, only two points are left at the end. But in practice we stop when fewer than 10 points remain.)

The total number of matching points after this filtering step is taken as an indicator of how well two pieces match. As will be shown in Section 4, this criterion is remarkably effective in detecting similarity.

3. Query Processing.

All music files are preprocessed into the intermediate representation of spectral vectors discussed earlier. Given a query sound clip (also converted into the intermediate representation), the database is matched against the query using minimum-distance matching and linearity filtering algorithm. The pieces that end up with the highest number of matching points (and if above a certain threshold) are selected as answers to the user query.

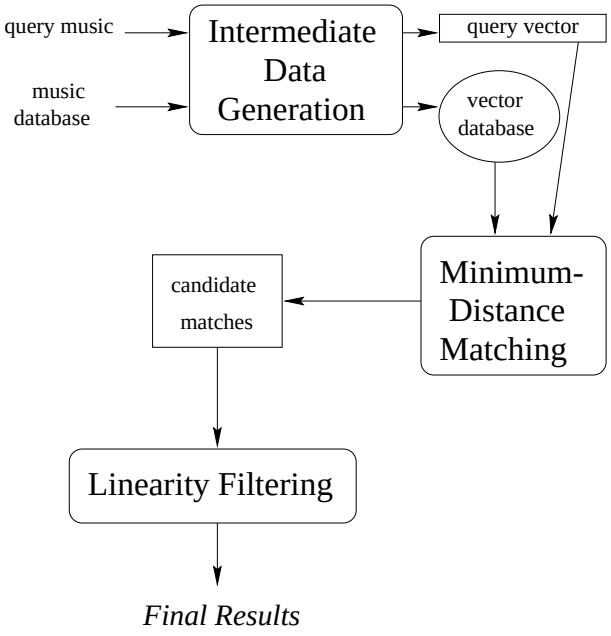

Figure 8. Summary of algorithm structure

Figure 8 summarizes the overall structure of the music retrieval algorithm.

3.3 Complexity Analysis

Time complexity of the preprocessing step is $O(n)$, where n is the size of the database. Because only "peak" information is recorded in the spectral vector representation, space required is only a fraction of the original audio database.

Dynamic programming for minimum-distance matching takes $O(m^2)$ time for each run, $O(m^2 n)$ overall, where m is the expected number of peaks in each piece. Because m is much less than n when the database is large, it can be regarded as a constant and $O(n)$ is the dominant factor.

Linearity filtering takes a negligible amount of time in practice, although its worst-case complexity is also up to $O(m^2 n)$.

Overall, assuming m is a constant factor, the algorithm runs in $O(n)$ time for each query. When the database gets large, the running time of $O(n)$ may be too slow. We are experimenting with indexing schemes [12] which will give better performance.

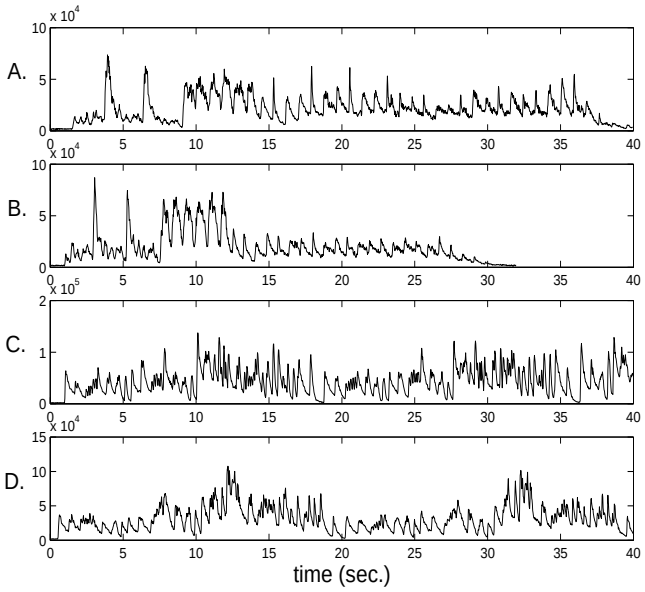

Figure 9. Power plots

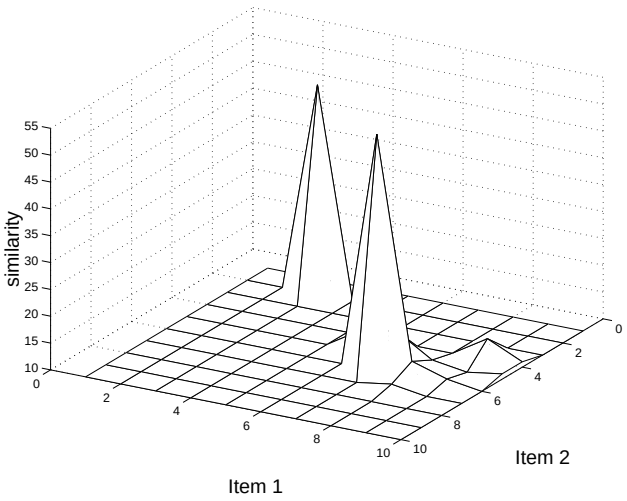

Figure 10. Pairwise matching result

4 Experiments

Our data collection is done by recording CDs or tapes into PCs through a low-quality PC microphone. No special efforts are taken to reduce noise. This setup is intentional, in order to test the algorithm's robustness and performance in a practical environment. Both classical music and modern music are included, with classical music being the focus. Instead of taking the entire pieces, only 30- to 60-second clips are taken from each piece, because that much data is generally enough for similarity detection.

We identify five different types of "similar" music pairs, with increasing levels of difficulty:

- Type I: Identical digital copy
- Type II: Same analog source, different digital copies, possibly with noise
- Type III: Same instrumental performance, different vocal components
- Type IV: Same score, different performances (possibly at different tempo)
- Type V: Same underlying melody, different otherwise, with possible transposition

Sound samples of each type can be found at `http://www-db.stanford.edu/~yangc/musicir/`.

Figure 9 shows the power plots of two different performances of Tchaikovsky's Piano Concerto No. 1 (A and B) and two different performances of Chopin's "Military" Polonaise (C and D). Both pairs are of Type-IV similarity. Each pair was performed by different orchestras, published by different companies. There were variations in tempo as well as in performance style. From the power plots it can be seen that notes are emphasized differently. Nevertheless, both pairs yield small distance scores after minimum-distance matching. On the other hand, a few dissimilar pairs also yield scores that are not large, such as Tchaikovsky's Piano Concerto No. 1 (A) vs. Brahms' Cradle Song (referred to as E from now on), and Chopin's "Military" Polonaise (D) vs. Mendelssohn's Spring Song (referred to as F from now on).

Figure 11 shows sample plots of optimal matching sets before linearity filtering (solid lines connecting the dots), where the horizontal axis is time (in seconds) of the first piece and vertical axis is time of the second piece. A straight line is fitted through each set of matching points (dashed lines). As is clear from the plots, A and B are truly similar (almost all points are colinear), while A and E are not; C and D are truly similar, while D and F are not.

After certain matching points are removed by linearity filtering, Figure 11 becomes Figure 12. The pairs (A, B) and (C, D) have 49 and 54 matching points respectively, while the other two pairs have fewer than 15 remaining matching points.

Figure 10 shows the pairwise matching result of a set of 10 music pieces, of which two pairs ((A, B) and (C, D)) are different performances of the same scores (with Type-IV similarity). The result is shown as a 10×10 matrix where the entry (i, j) gives the final number of matching points between two pieces i and j after linearity filtering. Because of symmetry only the upper triangle of the matrix is presented. Two peaks in the graph clearly indicate the discovery of the "correct" pairs.

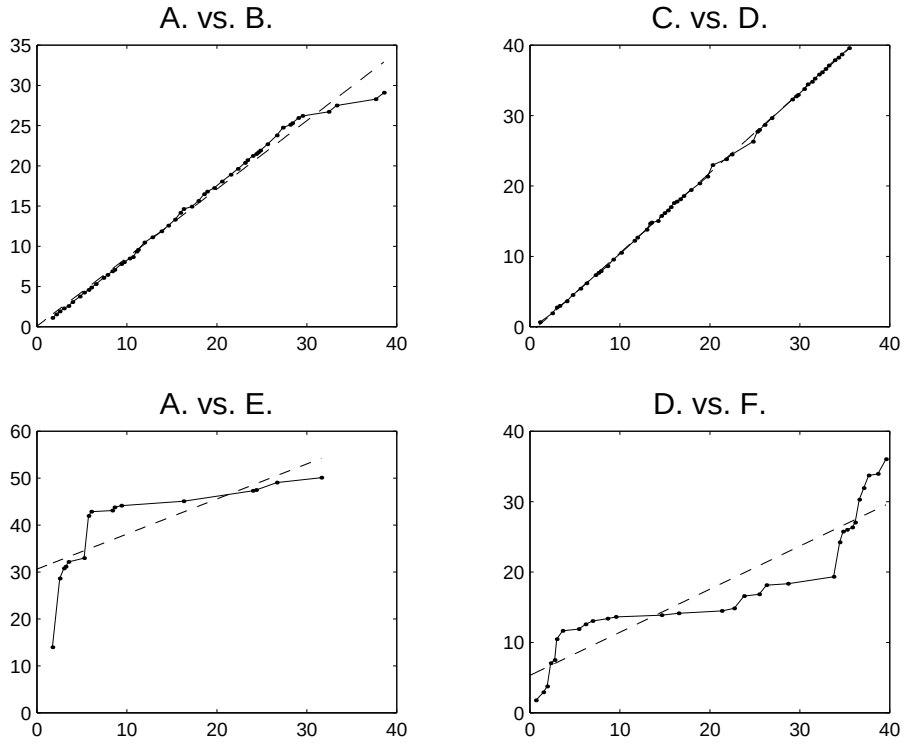

Figure 11. Matching plots before filtering

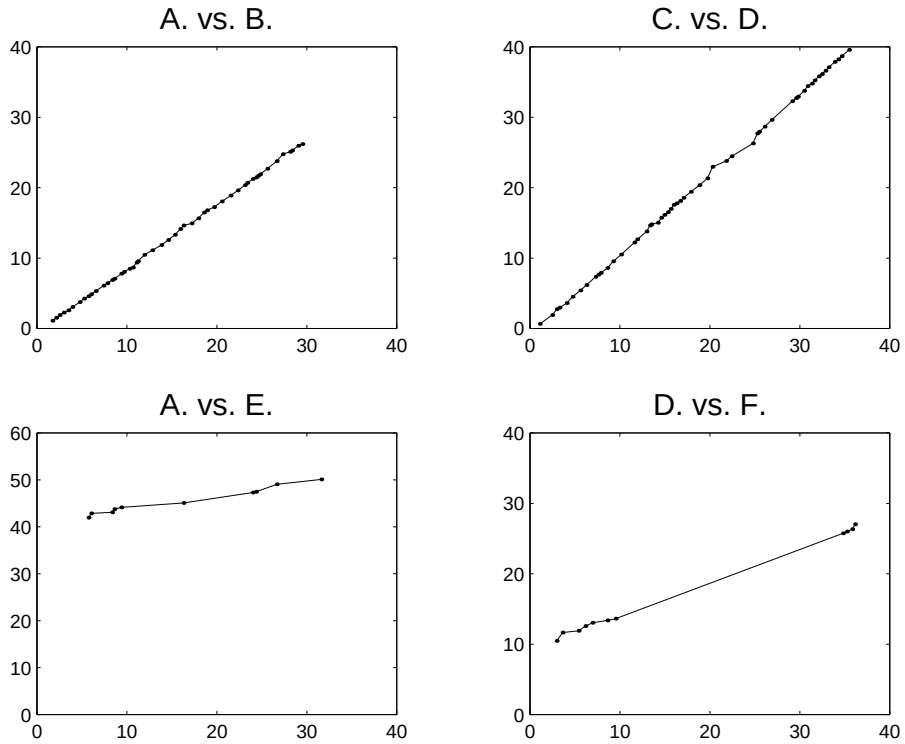

Figure 12. Matching plots after filtering

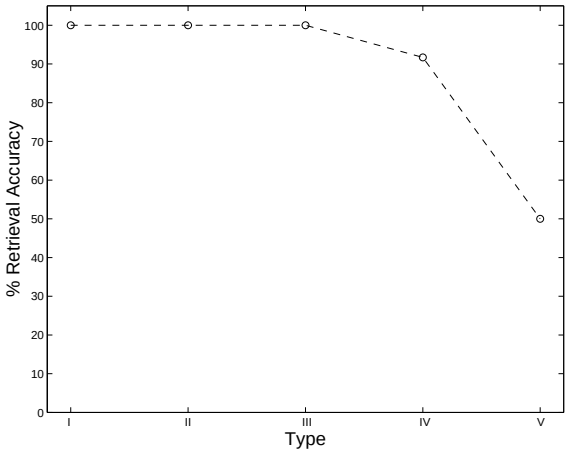

Figure 13. Retrieval Accuracy

More queries are conducted on a larger dataset of 120 music pieces, each of size 1MB. For each query, items from the database are ranked according to the number of final matching points with the query music, and the top 2 matches are returned. Figure 13 shows the retrieval accuracy for each of the five types of similarity queries. As can be seen from the graph, the algorithm performs very well in the first 4 types. Type-V is the most difficult, and better algorithms need to be developed to handle it.

5 Conclusions and Future Work

We have presented an efficient algorithm to perform content-based music retrieval based on spectral similarity. Experiments have shown that the approach can detect similarity while tolerating tempo changes, some performance style changes and noise, as long as the different performances are based on the same score.

Future research may include the study of the effects of various threshold parameters used in the algorithm, and to find ways to automate the selection of certain parameters to optimize performance.

We are experimenting with indexing schemes [12] in order to get faster retrieval response. We are also planning to augment the algorithm to handle transpositions (pitch shifts). Although transpositions of entire pieces are not very common, it is common to have small segments transposed to a different key, and it would be important that we detect such cases.

One other future direction is to design algorithms to extract high-level representations such as approximate melody contours. This task is certainly non-trivial, but it may be less difficult than transcription, and at the same time very powerful in similarity detection for complex cases.

Instead of using the peak-detection scheme during preprocessing, one can also incorporate existing rhythm detection algorithms to improve performance. Also, different algorithms may be suited to different types of music, so it may be helpful to conduct some analysis of general statistical properties before deciding which algorithm to use.

Content-based retrieval of musical audio data is still a new area that is not well explored. There are many possible future directions, and this paper is only intended as a demonstration on the feasibility of certain prototype ideas, of which more extensive experiments and research will need to be done.

References

[1] J. P. Bello, G. Monti and M. Sandler, “Techniques for Automatic Music Transcription”, in *International Symposium on Music Information Retrieval*, 2000.

[2] S. Blackburn and D. DeRoure, “A Tool for Content Based Navigation of Music”, in *Proc. ACM Multimedia*, 1998.

[3] J. C. Brown and B. Zhang, “Musical Frequency Tracking using the Methods of Conventional and ’Narrowed’ Autocorrelation”, *J. Acoust. Soc. Am.* 89, pp. 2346-2354. 1991.

[4] J. Foote, “ARTHUR: Retrieving Orchestral Music by Long-Term Structure”, in *International Symposium on Music Information Retrieval*, 2000.

[5] A. Ghias, J. Logan, D. Chamberlin and B. Smith, “Query By Humming – Musical Information Retrieval in an Audio Database”, in *Proc. ACM Multimedia*, 1995.

[6] 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”, in *Proc. ACM Digital Libraries*, 1996.

[7] E. D. Scheirer, “Pulse Tracking with a Pitch Tracker”, in *Proc. Workshop on Applications of Signal Processing to Audio and Acoustics*, 1997.

[8] E. D. Scheirer, *Music-Listening Systems*, Ph. D. dissertation, Massachusetts Institute of Technology, 2000.

[9] A. S. Tanguiane, *Artificial Perception and Music Recognition*, Springer-Verlag, 1993.

[10] G. Tzanetakis and P. Cook, “Audio Information Retrieval (AIR) Tools”, in *International Symposium on Music Information Retrieval*, 2000.

[11] E. Wold, T. Blum, D. Keislar and J. Wheaton, “Content-Based Classification, Search and retrieval of audio”, in *IEEE Multimedia*, 3(3), 1996.

[12] C. Yang, “MACS: Music Audio Characteristic Sequence Indexing for Similarity Retrieval”, in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2001.

[13] C. Yang and T. Lozano-Pérez, “Image Database Retrieval with Multiple-Instance Learning Techniques”, *Proc. International Conference on Data Engineering*, 2000, pp. 233-243.