MACS: MUSIC AUDIO CHARACTERISTIC SEQUENCE INDEXING FOR SIMILARITY RETRIEVAL

Cheng Yang

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

ABSTRACT

We present a prototype method of indexing raw-audio music files in a way that facilitates content-based similarity retrieval. 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.

Local peaks in signal power are identified in each audio file, and a spectral vector is extracted near each peak. Nearby peaks are selectively grouped together to form "characteristic sequences" which are the basis for indexing. A hashing scheme known as "Locality-Sensitive Hashing" is employed to index the high-dimensional vectors. Retrieval results are ranked based on the number of final matches filtered by some linearity criteria.

1. OVERVIEW AND RELATED WORK

Developments in internet technology have made a large volume of multimedia data, in particular music audio data, available to the general public, and yet there are not many search tools that can help users search through these data. Most existing search tools rely on file names or text labels, but they become useless when meaningful text descriptions are not available. As is happening with Napster users right now, people intentionally hide file names or text labels for legal reasons, and a larger and larger collection of music data is becoming unsearchable.

A truly content-based music retrieval system should have the ability to find similar songs based on their underlying score or melody, regardless of their text description. In addition, it can be used for such potential applications as music identification, plagiarism detection, copyright enforcement, etc. Traditional techniques used in text searching do not easily carry over to the music domain, and new technology needs to be developed.

As we know, 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 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. MIDI-style data can be synthesized into audio signals easily, but there is no known algorithm to do reliable conversion in the other direction (i.e., music transcription), except in monophonic or simple polyphonic cases [1, 3, 10]. Transcription of general polyphonic signal is extremely hard.

Past research on content-based music retrieval has primarily focused on score-based data such as MIDI, with input methods ranging from note sequences to melody contours to user-hummed tunes [2, 5, 7]. Because MIDI-style data is very structured, string matching or text searching methods can be applied. Very little has been done on retrieving raw audio music. However, 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. 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 they are fully or partially based on the same score, even if they are performed by different people or at different tempo.

Among music retrieval research conducted on raw audio databases, Scheirer [8, 9] studied pitch and rhythmic analysis, segmentation, as well as music similarity estimation at a high level such as genre classification. Tzanetakis and Cook [11] built tools to distinguish speech from music, and to do segmentation and simple retrieval tasks. Wold *et al.* at Muscle Fish LLC [12] 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. In our previous work [13] we also employed a dynamic programming matching approach based on a cost model, but preprocess the signals to identify peaks and only match the spectrograms near the peaks. Furthermore, we used some linearity filtering criteria to distinguish between a good match and a bad match. Both of these approaches lack scalability, and performance deteriorates rapidly when the database gets large. To address this issue, we propose a new scalable framework using indexing, therefore moving one step closer towards a practical music retrieval system.

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

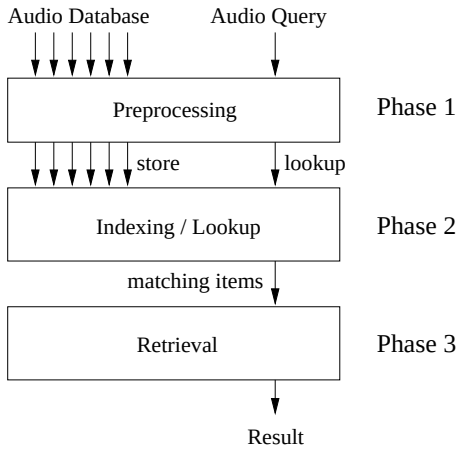

Figure 1: Structure of an Index-based Retrieval System

2. THE MACS INDEXING APPROACH

In this section we start with an architectural overview of the indexing algorithm, identify the main challenges, and then give a step-by-step description of our MACS (Music Audio Characteristic Sequence) indexing method.

2.1. Framework

Figure 1 shows the basic structure of an index-based retrieval system, which consists of three phases. Phase 1 (preprocessing phase) converts all raw audio data into "indexable" items, typically points in a high-dimensional vector space with an appropriate distance measure. Phase 2 (lookup phase) takes a query, converts it into indexable items as in Phase 1, and then performs a lookup in the index to get the best matching items. Phase 3 (retrieval phase) evaluates all the matching items and decides which of the original music pieces is the best candidate.

Each of the three phases poses its challenges. In phase 1, the challenge is to come up with a way to generate "indexable" items in a good vector space whose distance measure reflects music similarity. We propose a "characteristic sequence" method to address this issue. In phase 2, it is a nontrivial task to index and retrieve vectors efficiently in very high dimensions. We use the "Locality-Sensitive Hashing" (LSH) scheme [6] to implement this. In phase 3, it is important to have a systematic way of determining high-level similarity based on a set of matches among indexable items. We use a linearity filtering method as suggested in our previous work [13]. In the following sections we describe the three phases in more detail.

2.2. Generation of Characteristic Sequences

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.

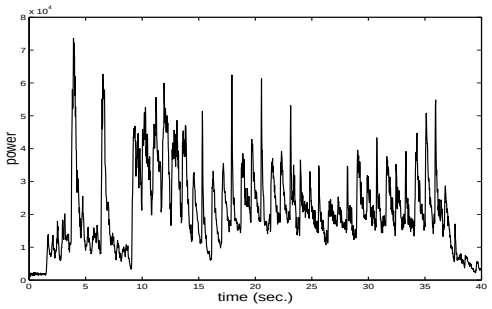

Figure 2: Power plot of Tchaikovsky's Piano Concerto No. 1

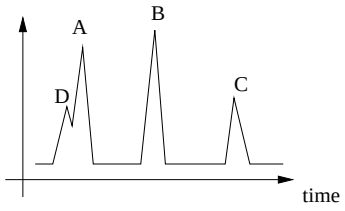

Figure 3: True Peak vs. Bogus Peak

For each audio file, a set of characteristic sequences is generated in the following way:

1. 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.
2. Plot the instantaneous power as a function of time. Figure 2 shows such a power plot for a 40-second sound clip of Tchaikovsky's Piano Concerto No. 1.
3. Identify peaks in the power plot, where peak is defined as a local maximum value within a neighborhood of a fixed size. This definition helps remove certain bogus local "peaks" which are immediately followed or preceded by higher values. For example, in Figure 3, A, B, C are true peaks but D is a bogus peak. Intuitively, these peaks roughly correspond to distinctive notes or rhythmic patterns, but with some errors, which will be compensated later.
4. Extract 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). This step generates a list of 180-dimensional vectors.
5. Pass each 180-dimensional vector through a set of 24 comb filters representing different pitch levels. Specifically, a comb filter representing a pitch at k Hz outputs the number $\sum_{i=1}^{6} G_{ik}$, where G_{ik} is the frequency component at frequency ik Hz. This step generates a list of 24-dimensional vectors, $v_1, v_2, \ldots, v_n$, where n is the number of peaks

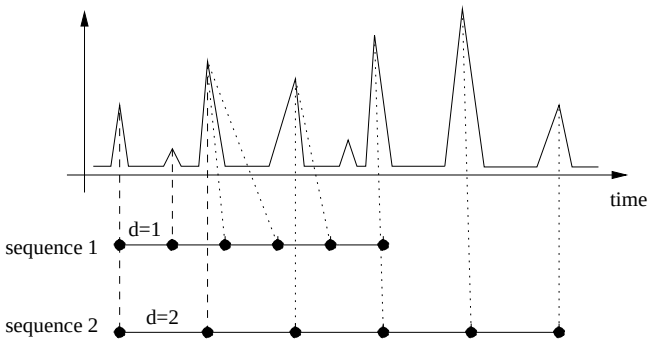

Figure 4: Construction of Characteristic Sequences

obtained. Intuitively, each vector estimates the pitch distribution at the corresponding time instant.

Additionally, we keep track of the time offsets of the original n peaks, $t_1, t_2, \ldots, t_n$, and define V_τ to be the vector v_k such that $t_k \leq \tau < t_{k+1}$, i.e., the last peak no later than time τ. It follows that $V_{t_i} = v_i$, and V_τ is undefined for $\tau < 1$.

6. Normalize vectors $v_1, v_2, \ldots, v_n$ so that they each have mean 0 and variance 1.

7. Construct a set of "characteristic sequences" as follows: for any two nearby peaks which are separated by fewer than D other peaks, identify a sequence of "follow-up" peaks which maintain roughly equal distance from each other, starting from the two original peaks. The process is illustrated in Figure 4. Formally, a characteristic sequence is given by

$$\{v_s, v_{s+d}, V_{t_s+2(t_{s+d}-t_s)}, V_{t_s+3(t_{s+d}-t_s)}, \cdots,$$

$$V_{t_s+(M-1)(t_{s+d}-t_s)}\}$$

where M is the pre-defined *length* of each sequence, s is the *starting point*, which ranges over all possible indexes, and d is the *bracketing control*, which takes on small integer values in the interval $[1, D]$. Note that each v vector is 24-dimensional, so the dimensionality of each characteristic sequence is $24M$.

In our experiments, we take $M = 6$ and $D = 3$. The purpose of having the bracketing control $d \in [1, D]$ is to offset the effects of possible bogus peaks that survived step 3, such as the second peak in Figure 4.

2.3. Indexing

Suppose we would like to compare two characteristic sequences $\{s_1, s_2, \ldots, s_M\}$ and $\{r_1, r_2, \ldots, r_M\}$. Let e be root-mean-squared error between vectors s and r. It can be shown that e is linearly related to the correlation coefficient of s and r and represents a degree of similarity. A smaller e value corresponds to a larger correlation coefficient. (See [14] for proof.) Therefore, Euclidean distance in the space of characteristic sequences can be used to estimate similarity.

Each characteristic sequence obtained from the previous section is a high-dimensional vector and can be indexed. We use the Locality-Sensitive Hashing (LSH) scheme [6] to perform indexing. Given a query vector, LSH is a fast probabilistic scheme that returns approximate nearest neighbors with a controllable false positive and false negative rate. Due to space limitations, we cannot elaborate on the details here.

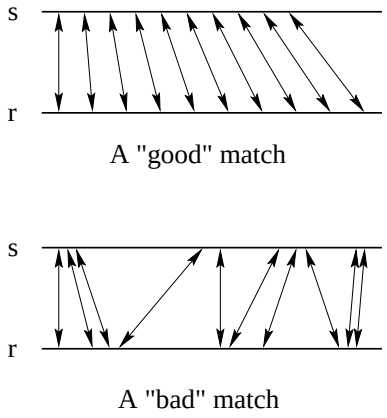

Figure 5: "Good" vs. "bad" matching

2.4. Matching with Linearity Filtering

To find similar music given a query music piece, we first break down the query into a set of characteristic sequences and perform an index lookup. Each lookup may generate a set of matches. Each match contains a tuple (query-offset, matching-offset) which indicates the time offsets of the two matching points. Figure 5 shows two ways to match s against r, both with 10 matches, 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.

Formally, let $M_k = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$ be the set of tuples of matching offsets. We can plot it on a 2-D graph, with $x_1, x_2, \ldots, x_k$ (of the first music piece) on the horizontal axis and $y_1, y_2, \ldots, y_k$ (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 matching tuples obtained from index lookup, 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 too few points remain.)

The total number of matching points after this filtering step is taken as an indicator of how well two pieces match. Music pieces from the database are ranked based on this number for retrieval.

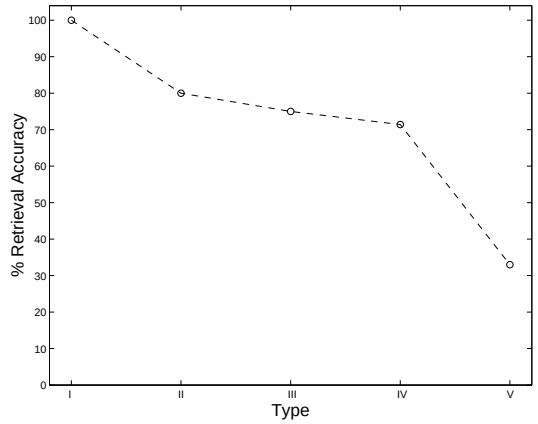

Figure 6: Retrieval Accuracy

3. 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 the website `http://www-db.stanford.edu/~yangc/musicir/` .

Queries are conducted on a 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 5 matches are returned. Figure 6 shows the retrieval accuracy for each of these five types of queries. As can be seen from the graph, the algorithm performs reasonably well in the first 4 types, but the last type is still too difficult.

4. CONCLUSIONS AND FUTURE WORK

We have presented an indexing 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 change is not too much and the different performances are based on the same score.

Further study is needed on the effects of various parameters used in the algorithm, in order to find ways to automate the selection of certain parameters to optimize performance.

Each of the three phases of the algorithm may need further refinement. Better signal processing techniques can be used in phase 1 to generate a more meaningful sequence representation; improved indexing techniques in phase 2 may further reduce false hits and speed up the lookup process; and more elaborate matching methods in phase 3 need to be developed to better compute high-level similarity from low-level matches.

We are also planning to augment the algorithm to handle more of the type-V case including transpositions (pitch shifts).

5. 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] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality", in *Proc. 30th Symposium on Theory of Computing*, 1998.

[7] 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.

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

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

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

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

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

[13] C. Yang, "Music Database Retrieval Based on Spectral Similarity", *Stanford University Database Group Technical Report* 2001-14.

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