Efficient Acoustic Index for Music Retrieval with Various Degrees of Similarity

Cheng Yang*
Department of Computer Science
Stanford University
Stanford, CA 94305, U.S.A.
yangc@cs.stanford.edu

ABSTRACT

Content-based music retrieval research has mostly focused on symbolic data rather than acoustical data. Given that there are no general-purpose transcription algorithms that can convert acoustical data into musical scores, new methods are needed to do music retrieval on acoustical data. In this paper, we review some existing methods on content-based music retrieval, discuss different definitions of music similarity, and present a new framework to perform music indexing and retrieval. The framework is based on an earlier prototype we developed, with significant improvements.

In our framework known as *MACSIS*, each audio file is broken down into small segments and converted into feature vectors. All vectors are stored in a high-dimensional indexing structure called *LSH*, a probabilistic indexing scheme that makes use of multiple hashing instances in parallel. At retrieval time, small segments of audio matches are retrieved from the index and pieced together using the Hough Transform technique, and results are used as the basis to rank candidate matches.

1. INTRODUCTION

You hear a song on the radio that sounds familiar, but you cannot recall the name. How can you find it out?

A band is playing a familiar tune that you think is a variation of some other piece that you heard a long time ago. How do you get the original piece?

You are a talented artist and have published a new song, but worry that others may have stolen your theme, modified it, and posted it somewhere on the web. How can you check whether it is on the web?

All of the above are examples of the content-based music retrieval problem, i.e., given a segment of a music piece, find similar occurrences from the database. Developments in internet technology have made a large volume of multimedia data, in particular music audio data, available to the general public; however, content-based music search technologies did not keep up: almost all "music search" engines rely on file names or other text labels to do the search. These methods become useless when meaningful text descriptions are not available. 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.

Computer representation of music comes in two different ways. One way is symbolic representation based on musical scores. For each note in the score, it keeps track of pitch, duration (start time/end time), strength, as well as other pertinent information. 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 [2, 4, 14]. Polyphony refers to the scenario when multiple notes occur simultaneously in music. As we know, most music pieces today are polyphonic. Transcription of general polyphonic signal is extremely hard. Because of this, music retrieval algorithms that deal with acoustic data are very different from those that deal with symbolic data.

In Section 2 we will give some background information, including a classification of current music retrieval systems, different definitions on music similarity and the problem we are trying to solve. In Section 3 we will discuss our algorithms in detail. Sections 4 and 5 will present experimental results and future work.

* Supported by a Leonard J. Shustek Fellowship, part of the Stanford Graduate Fellowship program, and NSF Grant IIS-0085896.

Database \ Query		Symbolic	Acoustic	
			Monophonic	Polyphonic
Symbolic		**S**	***QBH***	**?**
Acoustic	Monophonic	*Solvable, but may not be interesting in practice*		**?**
	Polyphonic	**?**	**?**	**A**

Figure 1: Classification of music retrieval systems

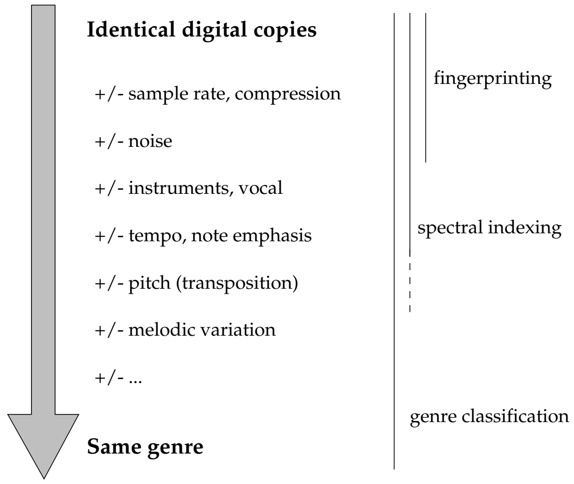


Figure 2: Different definitions of music similarity

2. BACKGROUND

Depending on music data types used in queries and the underlying database, music retrieval systems can be classified according to Figure 1.

2.1 Classification of Music Retrieval Systems

In the category marked "S" in Figure 1, both the query and the underlying database are in symbolic formats, such as MIDI and Humdrum. The music retrieval problem becomes similar to a string matching problem, and can be solved by methods derived from text searching techniques. Several systems of this type have been implemented, including the Themefinder project (`http://www.themefinder.org`), where the symbolic database can be searched using pitch sequences, intervals, approximate contours, etc.

The adjacent category, monophonic acoustic query with a symbolic database, represents a problem which has received considerable attention during the past few years. The problem is known as *Query by Humming*, or *QBH* [3, 5, 7, 8, 11, 12, 15]. In such systems, the input query is typically given as a user-hummed melody through a microphone, and the melody is analyzed and matched against a symbolic database. Human-hummed tunes are monophonic melodies and can be automatically transcribed into pitches with reasonable accuracy. To compensate for possible inaccuracies of human-hummed tunes, some systems use approximate contour information (up, down, etc) or beat information to aid the retrieval process.

Because monophonic transcription is relatively easy, it is conceptually feasible to solve the problem of retrieving from monophonic acoustic databases with monophonic or symbolic queries. However, such systems would be of little practical value, since there is not much monophonic audio data available to start with. Therefore, we mark the corresponding categories as "solvable, but not interesting in practice".

The categories with question marks remain open problems. Due to the lack of general-purpose polyphonic transcription algorithms, we cannot expect to solve these problems by reducing them to monophonic or symbolic cases.

Finally, the shaded area marked "A" is what we are going to focus on, namely, retrieval from a polyphonic audio database in response to a similar audio query. At first sight, this area may seem even harder than those marked with question marks. However, this is not the case: solving problems in this category does not necessarily require polyphonic transcription. In fact, we do not attempt to do polyphonic transcription in our work, but try to process and match spectrograms without converting them into symbolic abstractions.

Depending on the definition of "similarity," this category can be further divided into different directions, as shown in Figure 2.

2.2 Music Similarity Definitions

There is no consensus on the exact meaning of music similarity, and it can be defined over a broad continuum, as in Figure 2. On one extreme, we can regard only identical digital copies of music as "similar," and anything else as dissimilar. With this definition, the music retrieval problem reduces to a generic data retrieval problem, which can be achieved through traditional hashing and indexing techniques. Alternatively, we can tolerate some distortions due to sample rate change, compression, or noise, and still regard the results as similar. Or, we can tolerate changes in instruments, vocal parts or tempo, and still regard the results as similar, as long as the musical "score" remains unchanged. On the other extreme, we can totally disregard the score, and define similarity as "within the same genre" - a somewhat subjective notion. The difference in similarity definitions brings about different directions in music retrieval research.

"Fingerprinting" techniques focus on finding almost-identical music recordings while tolerating small amount of noise distortions [1]. The central idea is to extract some representative "digital signatures" from the acoustic data which can be hashed and retrieved efficiently. Several proprietary systems have been developed at companies such as Relatable (`http://www.relatable.com`), which is working with Napster on implementing music file filters to enforce copyright protection, and *CD (`http://www.starcd.com`), whose music identification system tracks songs played on radio stations and provides users with real-time identification results.

Genre classification techniques focus on classifying music data (sometimes speech data or other forms of audio data) based on high-level properties such as energy distribution, timbre features, brightness, texture, rhythmic patterns, and so on [13, 16, 17, 18]. Machine learning techniques such as automatic clustering are often employed during training.

Somewhere between fingerprinting approaches and genre classification approaches, there is a large area which remains mostly unexplored. The corresponding similarity definition involves similarity in musical scores, regardless of tempo, instrumentation or performance style. This definition reflects an intuitive notion of "same song" as perceived by human listeners. We are going to address this problem with a spectral indexing technique.

2.3 Problem Statement

Informally, our problem can be defined as: given a query music clip in raw audio format, find similar pieces from the audio database, where 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. Retrieval results should be a list of songs ranked by computed similarity estimate.

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

The first two types overlap with what existing fingerprinting techniques can handle. Our objective is to handle the other similarity types as well as the first two.

Foote [6] experimented with this kind of 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 [20] we also employed a dynamic programming matching approach based on a cost model, but preprocess the signals to identify peaks and only match spectrograms near the peaks. Furthermore, we used some linearity filtering criteria to distinguish between a good match and a bad match. Both of

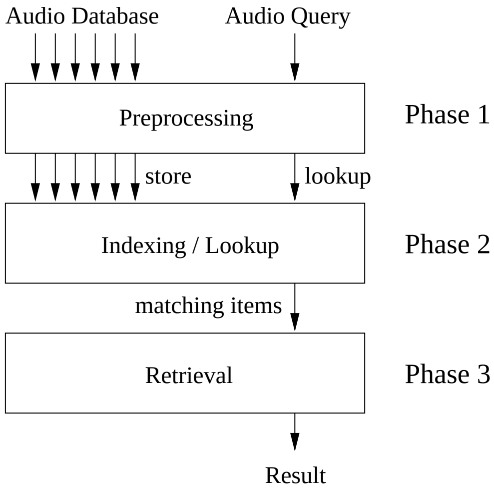


Figure 3: Structure of an index-based retrieval system

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. This framework is based on our prototype system described in [19], with some significant improvements in Phase 2 and Phase 3, as explained in the next section.

3. THE MACSIS FRAMEWORK

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 indexing system known as MACSIS (Music-Audio Characteristic Sequence Indexing System).

3.1 Overview

Figure 3 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. It also processes query audio clips into the same type of vector items. Phase 2 (indexing/lookup phase) creates an index of all vectors generated by Phase 1 for the audio database. It also takes query vectors, similarly processed by Phase 1, and performs 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. To address this issue, we propose a "characteristic sequence" method. First, we extract spectrogram feature values near signal intensity peaks; then, we try to align the peaks into equally-spaced beats; finally, we concatenate signal features near estimated beat locations.

- In phase 2, it is a nontrivial task to implement a high-dimensional index where "similar" (but not necessarily identical) vectors can be retrieved efficiently. We use the "Locality-Sensitive Hashing" (LSH) scheme [9], which runs many hashing instances in parallel: in each instance, the vector is hashed by a function so that similar vectors are "likely" to be hashed to the same value, with a certain probability.
- In phase 3, it is necessary to have a systematic way of determining high-level similarity based on a set of matches among short, indexable items. During retrieval, a vector from the query may match many different items in the database; it may even have multiple matches within the same music piece, since many music patterns tend to repeat itself. Partial matches need to be pieced together to get "global" matching results. Possible tempo changes add to the complexity of the problem. We make use of the fact that tempo changes must be uniform in time, and use Hough Transform to solve this final step.

In the following sections we describe the three phases in more detail.

3.2 Generation of Characteristic Sequences

Techniques used in Phase 1 are similar to those proposed in our earlier work [19].

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.

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 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.
3. Identify peaks in the power plot, where peak is defined as a local maximum value within a neighborhood of a predefined size. This definition helps remove certain bogus local "peaks" which are immediately followed or preceded by higher values. Intuitively, these peaks roughly correspond to distinctive notes or rhythmic patterns, but with some errors, which will be compensated later.

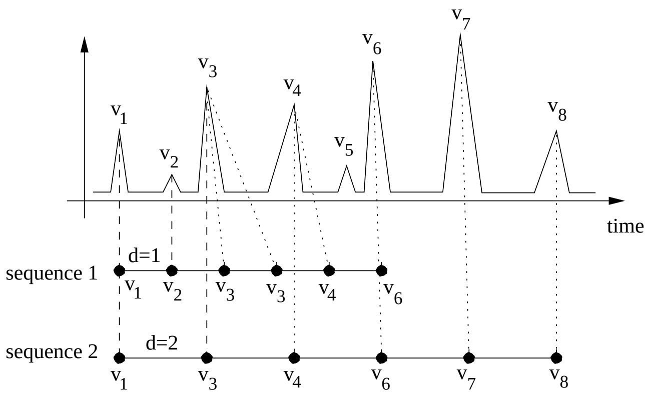


Figure 4: Construction of Characteristic Sequences

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. For each 180-dimensional vector u_j, convert it into a 24-dimensional vector v_j that represents 24 different pitch levels in the following way: $v_{j,k}$ $(k = 1, \ldots, 24)$ represents the pitch p_k Hz and is defined by: $v_{j,k} = \sum_{i=1}^{6} G_{j,ip_k}$, where G_{j,ip_k} is the frequency component of vector u_j at frequency ip_k Hz. This step generates a list of 24-dimensional vectors, v_j, $j = 1, 2, \ldots, n$, where n is the number of peaks 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 $t_k \le \tau < t_{k+1}$, i.e., the last peak no later than time τ. It follows that $V_{t_i} = v_i$, and V_τ is undefined for $\tau < t_1$.
6. 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

$$\{v_s, v_{s+d}, V_{t_s+2(t_{s+d}-t_s)}, V_{t_s+3(t_{s+d}-t_s)}, \ldots,$$
$$V_{t_s+(M-1)(t_{s+d}-t_s)}\}$$

   where M is the predefined *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 $24M$.

   For the example in Figure 4, the sequence with $d = 1$ is $\{v_1, v_2, v_3, v_3, v_4, v_6\}$, which does not align well with the beat (due to the bogus peak v_2); on the other hand, the sequence with $d = 2$ is $\{v_1, v_3, v_4, v_6, v_7, v_8\}$, which aligns perfectly with the beat. Each v vector is 24-dimensional, so each length-6 sequence here is 144-dimensional. 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 v_2 here.

In our experiments, we take $M = 6$ and $D = 3$.

3.3 Indexing

Suppose, during retrieval, we would like to compare two characteristic sequences $\{s_1, s_2, \ldots, s_M\}$ and $\{r_1, r_2, \ldots, r_M\}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be a random sample of $1, 2, \ldots, M$, and let $e(\alpha)$ be the correlation coefficient between the two vectors $\{s_{\alpha_1}, s_{\alpha_2}, \ldots, s_{\alpha_n}\}$ and $\{r_{\alpha_1}, r_{\alpha_2}, \ldots, r_{\alpha_n}\}$. The expected value of $e(\alpha)$ is a natural indicator of perceived similarity between the two characteristic sequences s and r.

Each characteristic sequence obtained from the previous section is a high-dimensional vector and can be indexed. The design goal of such an index is to facilitate retrieval of "similar" vectors as defined above. We use the Locality-Sensitive Hashing (LSH) scheme [9] to perform indexing. Given a query vector, LSH is a fast probabilistic scheme that returns approximate matches with a controllable false positive and false negative rate.

Figure 5 illustrates our LSH implementation. It consists of many different "hashing instances," where each hashing instance is designed to map vectors into hash values so that "similar" vectors are hashed into the same hash value with high probability. Each hashing instance has its own hash table to store hash values as well as the corresponding pointers to original vectors. All hashing instances and hash tables are independent of each other, and can be looked up in parallel. If two vectors are truly similar, their hash values are likely to agree in many of such instances. Therefore, when we process a query lookup from the hash tables, we focus on those vectors that match the query on many different hashing instances.

The key to the design is to find a family of hashing instances so that "similar" vectors can be hashed into the same hash value with high probability. Our design is shown in Figure 6, which gives an example of one hashing instance. First, the raw vector goes through a simple dimensionality-reduction routine which picks a random subset of its dimensions. Next, the sampled dimensions are normalized (to zero mean and unit variance) and passed through a low-resolution quantization grid, so that each dimension is quantized and converted into a small integer. Finally, the resulting vector of integers is passed through a universal hashing function in which each dimension is multiplied by a random weight and their sum is taken to form the final hash value, modulo the number of hash buckets.

All the random parameters (dimension samples, quantization grid lines, hashing weights) are pre-generated and fixed for each hashing instance, but vary across different instances. If two raw vectors are "similar," i.e., with a high correlation coefficient with sampled dimensions, they must be close to each other in Euclidean space with these sampled dimensions after normalization [21], so they have a good chance of being mapped to the same point after quantization. Of

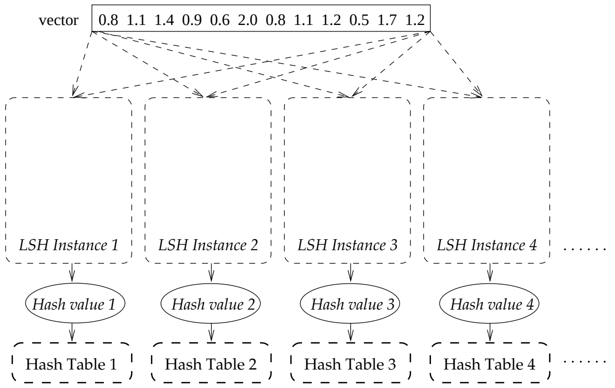


Figure 5: Multiple hashing instances with independent hash tables

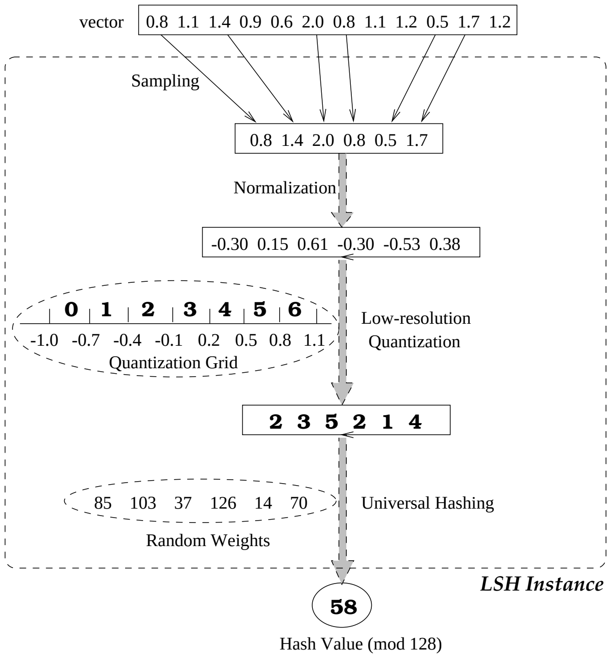


Figure 6: A sample hashing instance

course they may also be unlucky and happen to lie across the quantization grid lines even though they are close to each other. In such cases they will not be mapped to the same hash value. However, since we have many hashing instances running in parallel, there is a high probability that they would be mapped to the same hash value in several such instances. Such probabilities cannot be quantitatively analyzed without an exact mathematical definition of the similarity measure. However, the above reasoning suggests that similarity between raw vectors can be represented by the number of hashing instances that it matches in terms of hash values.

3.4 Matching with Hough Transform

Each characteristic sequence represents a short segment of a music piece. To find similar music given a query, we first break down the query into a set of characteristic sequences and perform an index lookup. Each lookup may generate

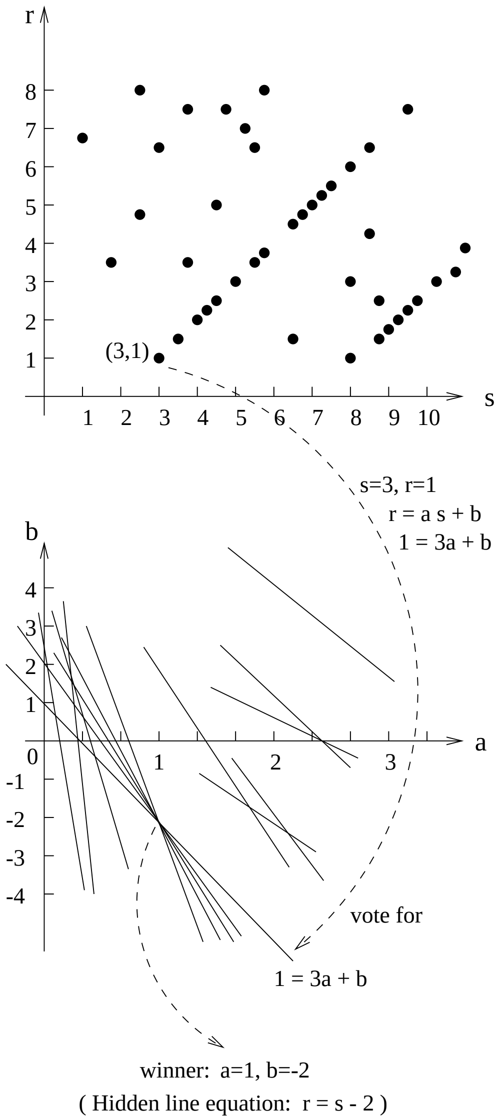


Figure 7: Illustration of Hough Transform

a set of matches. Each match contains a tuple (query-offset, matching-offset) which indicates time offsets of the two matching points. Since music tempo changes must be uniform in time, we observe that all matches on short segments must be well aligned; in other words, if we plot the matching tuples on a 2-D graph, we should be able to see a straight line if the two pieces are based on the same score.

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, we should be able to find many of the plotted points along a straight line. Without tempo change, the line should be at a 45-degree angle. With possible tempo change, the line will be at a different angle, but it still should be straight.

In Figure 7, the top graph shows a possible plot with matching points. The longest "line" (starting at $(3, 1)$ at a 45-degree angle) corresponds to the actual match, while possible shorter lines (such as the one to the right) are results of repeated patterns common in real-world music. Other scattered points are due to error or random matches. Each point has an associated weight, which indicates confidence of the match (obtained from the previous indexing step).

To find the hidden straight line from the matching plot, we utilize a computer vision technique known as Hough Transform [10], which we discuss below. Compared with the Linearity Filtering approach used in our previous work [19], the Hough Transform is more accurate and takes roughly an equal amount of time to execute.

We model the hidden straight line by the equation $r = as + b$, where r and s are the matching time offset values of two music pieces. The values of a and b are estimated in the following way: each matching tuple (s_0, r_0) votes (with its own weight) for a set of possible (a, b) values that satisfy $r_0 = as_0 + b$. In this case, each matching tuple votes for a straight line in (a, b) space. After all matching tuples cast their votes, the (a, b) pair that receives the highest vote becomes the winner. If the vote is large enough, then we conclude that $r = as + b$ is the hidden straight line we are trying to find. (See the bottom graph of Figure 7). This vote can also be used to measure confidence in matching these two music pieces.

In our implementation, we take a query audio clip, convert it into characteristic sequences and feed them into the indexing engine; when the matching results come back, we pick those music pieces with a large number of matching points, apply Hough Transform to them, and rank them based on the highest vote from the Hough Transform algorithm. One way to speed up retrieval is to process only a small sample of vectors (characteristic sequences) from the query rather than the entire query. This will be discussed further in the next section.

4. EXPERIMENTS

Experiments have been conducted on a database of 2,000 minutes of music recordings. Our data collection is done in two ways, by digitally ripping CDs into .wav format, and by recording CDs or tapes into PCs through a low-quality microphone. The latter is intended to add some realistical noise to test the system's robustness and performance in a practical environment. Both classical and modern music are included, with classical music being the focus. Queries are given in 30- to 60-second clips taken from the database.

Five types of similarity are defined according to Section 2.3. Type I is the easiest (identical digital copies) while Type V is the most difficult (similarity in melody). Sound samples of each type can be found at the website `http://www-db.stanford.edu/~yangc/musicir/`. "Correct" similarity pairs (based on the music title) are hand-annotated in the database for evaluation purposes, but are not made available to the retrieval algorithm. For each query, the retrieval engine ranks candidate items from the database. If the "correct" answers appear within the top 5 matches, then it is considered correct. For each similarity type, retrieval accuracy is defined as the percentage of "correct" answers given by the retrieval engine. Our result graphs are based on a set of 220 queries. Due to the lack of standardized testbed collections among music retrieval researchers, we are not able to provide performance comparisons with other systems.

Figure 8 shows the average retrieval accuracy for each similarity type. Because LSH indexing is probabilistic in nature, each run may generate a slightly different result due to random initializations. Therefore, we run queries multiple times with different random seeds, and average the results. As one can see, the system performs very well for similarity types up to Type IV (same score but different performances). Type V is not well handled, since it includes pitch transpositions which are not taken into account with our feature vector design.

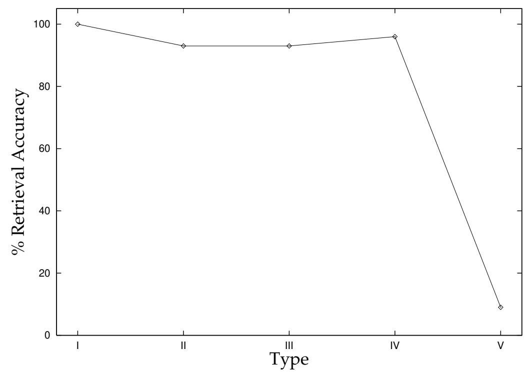
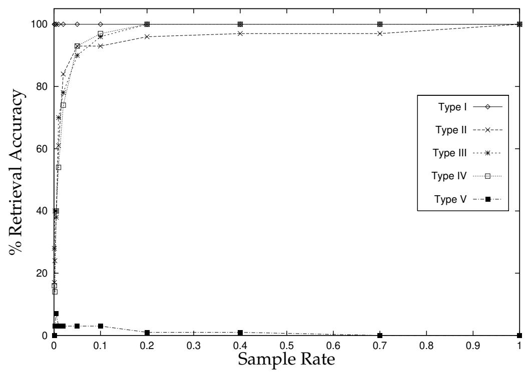


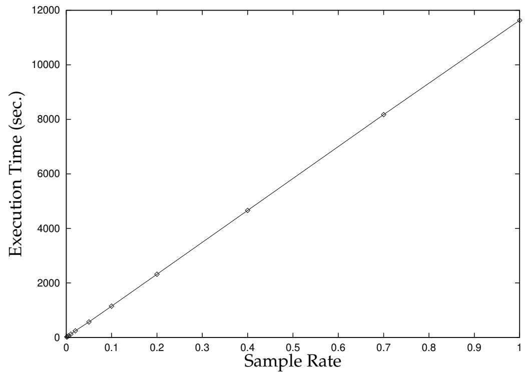
Figure 8: Retrieval accuracy for different similarity types

As mentioned at the end of last section, retrieval speed can be improved by using only partial queries. Figure 9(a) shows the retrieval accuracy of using "sampled" queries at different sampling rates. A subset of characteristic sequences from the query clip is sampled at random and fed into the retrieval engine. Figure 9(b) shows the corresponding execution times on the entire set of test queries. The "sampled" query approach results in a significant speedup, while not sacrificing much in performance when the sampling rate is above 5%.

Rather than selecting query vectors at random, we can also select them based on input time. As soon as the first t seconds of query data is obtained from the input device, retrieval is performed on the "truncated" audio data. This setting simulates a real-time environment where the retrieval engine can return results while the query is still being played. Figure 10(a) shows the retrieval accuracy as a function of the length of query input (when sampling rate is fixed at 5%), and Figure 10(b) shows the corresponding time required to process the entire set of queries. In this case, with a sampling rate of 5%, 20-30 seconds of a query can lead to reasonable accuracy. With a higher sampling rate, even shorter queries will be good enough.



(a)



(b)

Figure 9: Retrieval accuracy and time with sampled queries

5. SUMMARY AND FUTURE WORK

We have given a classification of current music retrieval systems based on music data types, and have presented our spectral indexing algorithm to perform content-based music retrieval on acoustical data with acoustical queries. 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 could lead

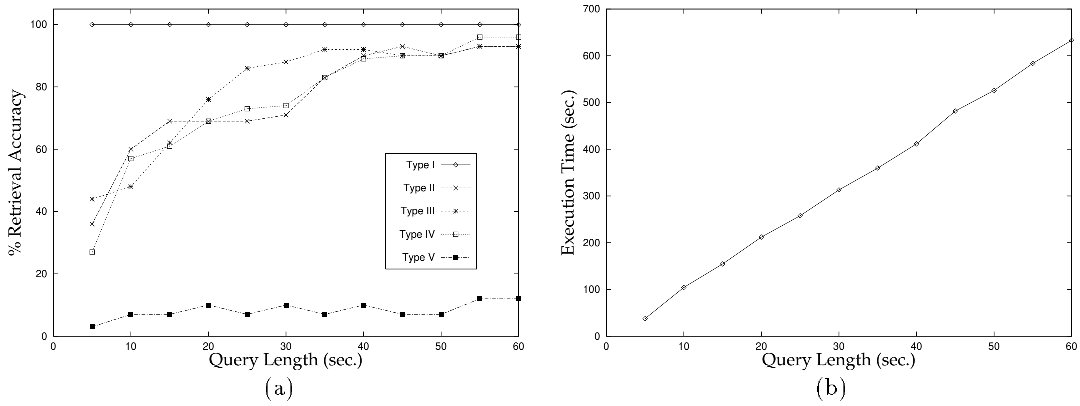


Figure 10: Retrieval accuracy and time with truncated queries

to more accurate high-level similarity estimation from low-level matches.

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

6. REFERENCES

[1] E. Allamanche, J. Herre, O. Hellmuth, B. Fröba, T. Kastner and M. Cremer, "Content-based Identification of Audio Material Using MPEG-7 Low Level Description", in *International Symposium on Music Information Retrieval*, 2001.

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

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

[4] 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.

[5] W. Chai and B. Vercoe, "Melody Retrieval On The Web", *Proc. Multimedia Computing and Networking*, 2002.

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

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

[8] G. Haus and E. Pollastri, "An Audio Front End for Query-by-Humming Systems", in *International Symposium on Music Information Retrieval*, 2001.

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

[10] R. Jain, R. Kasturi and B. G. Schunck, *Machine Vision*, McGraw-Hill, 1995.

[11] N. Kosugi, Y. Nishihara, T. Sakata, M. Yamamuro and K. Kushima, "A Practical Query-By-Humming System for a Large Music Database", in *ACM Multimedia*, 2000.

[12] 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.

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

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

[15] Y. H. Tseng, "Content-based retrieval for music collections", in *ACM-SIGIR*, 1999.

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

[17] G. Tzanetakis, G. Essl and P. Cook, "Automatic Musical Genre Classification of Audio Signals", in *International Symposium on Music Information Retrieval*, 2001.

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

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

[20] C. Yang, "Music Database Retrieval Based on Spectral Similarity", in *International Symposium on Music Information Retrieval*, 2001.

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