The MACSIS Acoustic Indexing Framework for Music Retrieval: An Experimental Study

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

ABSTRACT

We present an efficient and scalable system that indexes acoustic music data for content-based music retrieval. Both the music database and input queries are given in raw audio formats without metadata or other symbolic information; retrieval is targeted at music pieces which are "similar" to the query sound clip.

Our framework is designed as a series of modular pipeline stages and phases. Each music file entering the pipeline is transformed into spectrogram vectors and then into *characteristic sequences*, representing small segments of audio features that can tolerate some noise and tempo variations. These sequences are placed in a high-dimensional indexing structure. Retrieval results from the index are ranked based on alignment of short matching segments. Each module of the framework can be independently changed or replaced, and we study their effects by a set of experiments.

1. INTRODUCTION

Past research on content-based music retrieval has primarily focused on symbolic data rather than acoustic data. With symbolic data representation of music, the data file keeps track of each note's pitch, duration (start time/end time), strength, as well as other pertinent information. Examples of such representations include MIDI and Humdrum, with MIDI being the most popular format. Many operations on such data are variations of string matching methods. With acoustic data representation, however, no information about individual notes is given; only audio intensity values are recorded as a function of time, sampled at a certain rate, often compressed to save space. Examples include .wav, .au and MP3. Symbolic music 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 from raw acoustic signals). Therefore, content-based music retrieval on acoustic data would require a new set of techniques that differ significantly from symbolic music data retrieval algorithms.

Because the majority of music data available on the internet are represented in one of the acoustic formats, it is important to have search and retrieval algorithms on such data. Potential applications include automatic music identification, music analysis, plagiarism detection, copyright enforcement, etc. 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 metadata description or file names.

Most music pieces have multiple notes occurring simultaneously, a scenario known as *polyphony*. Polyphony is handled in symbolic representations by allowing the notes' timings to overlap. In acoustic representations, polyphony results in signals that are the sum of signals that would have been generated by individual notes. With access only to the final sum of signals, not individual components, it is very difficult to reconstruct the individual notes from polyphonic audio. 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 [2, 4, 14], not in the general polyphonic case. In our work, we deal with polyphonic music data without attempting transcription.

2. BACKGROUND

Much of previous work on content-based music retrieval falls into two categories: symbolic query on a symbolic database, or monophonic acoustic query on a symbolic database.

- Symbolic query on a symbolic database: Both the query and the underlying database are in symbolic formats, such as MIDI and Humdrum. Retrieval problems on such symbolic data can typically be addressed 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.
- Monophonic acoustic query on a symbolic database: This problem, also known as *Query by Humming* or *QBH*, has received considerable attention during the past few years [3, 5, 7, 8, 11, 12]. In such systems, the input query is typically 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. In order 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.

Both of these categories deal with symbolic music databases. New methods need to be developed to handle acoustic music databases. Our research focuses on similarity retrieval from a polyphonic acoustic database in response to a polyphonic acoustic query. Related work in this area can be categorized based on different definitions of "similarity," which are discussed next.

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

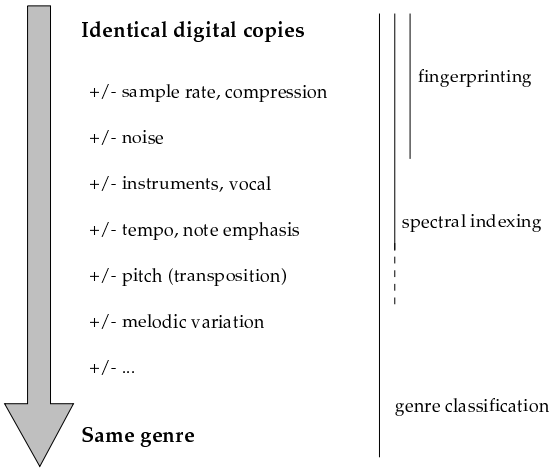

Figure 1: Different definitions of music similarity

2.1 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 1. 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 regular 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 music recordings that are almost identical 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 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, 15, 16, 17]. In many cases, machine learning techniques such as automatic clustering are 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 focus on this definition here.

2.2 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 [18] we also employed a dynamic programming matching approach based on a cost model, but preprocessed the signals to identify peaks and only matched 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 proposed a prototype system using spectral indexing [19], and further developed it into a scalable framework known as MACSIS (*Music-Audio Characteristic Sequence Indexing System*) [20]. The framework involves multiple pipeline modules which can be independently redesigned and replaced. In this paper, we study the effects of several design choices involving such modules.

3. ACOUSTIC INDEXING FRAMEWORK

Figure 2 shows the basic structure of our index-based retrieval system, *MACSIS*. It consists of three phases, which are summarized below:

- Phase 1 (preprocessing phase): Raw audio is converted into "indexable" items known as *characteristic sequences*, which can be viewed as points in a high-dimensional vector space with an appropriate distance measure. Each characteristic sequence represents a short segment of music data. Both the audio database and the query are processed in this way. This phase can be further divided into a pipeline of multiple steps: from raw audio to spectrum; from spectrum to event vector; from event vector to characteristic sequence. Each step in the pipeline can

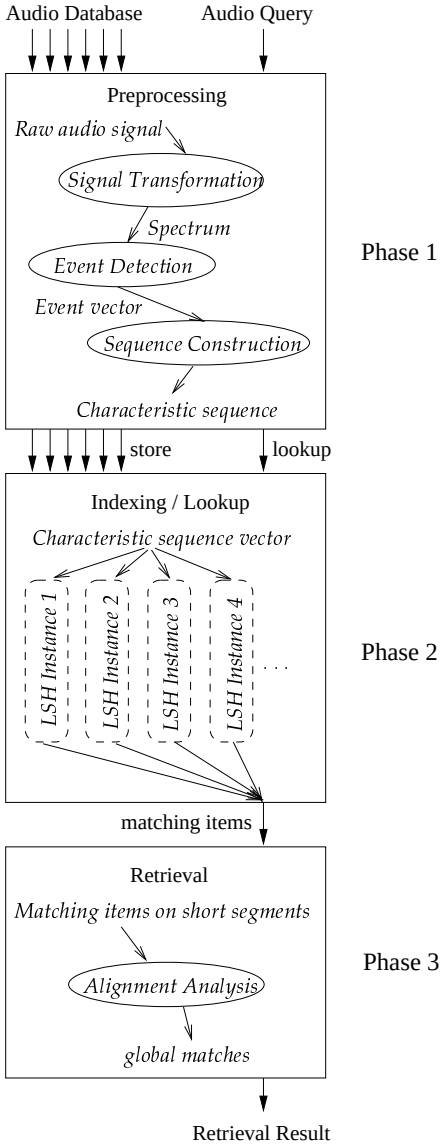

Figure 2: Structure of MACSIS

be implemented in different ways, leading to different results.

- Phase 2 (indexing / lookup phase): Characteristic sequences for the audio database (generated by Phase 1) are indexed in a high-dimensional indexing scheme known as *Locality-Sensitive Hashing*, or *LSH* [9]. This scheme 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. At query time, characteristic sequences for queries are looked up in the LSH index to get matching items.
- Phase 3 (retrieval phase): Matches on characteristic sequences are pieced together to find "global" matches. During retrieval, a characteristic sequence from a 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. 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 discuss two different methods (Linearity Filtering vs. Hough Transform) to determine "global" matches based on partial matches of characteristic sequences.

At query time, the pipeline in Phase 1 can take real-time audio as input and produce characteristic sequences for lookup and retrieval on-the-fly.

Each phase has several design choices which affect final performance. In the following sections we discuss such choices. For a detailed treatment of algorithms please refer to [20].

3.1 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.

This phase can be regarded as a pipeline of three subcomponents.

3.1.1 From Raw Audio to Spectrum

We use the Short-Time Fourier Transform (STFT) to extract instantaneous frequency distributions from the signals. We 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. Other types of signal processing may also be used here; in this paper we only focus on the STFT.

3.1.2 From Spectrum to Event Vector

This step involves analyzing the spectrogram to identify significant "events" in the recording, and to obtain a compact representation of vectors corresponding to events. In our implementation described below, "events" are defined as peaks in signal power.

1. Plot the instantaneous power as a function of time.
2. 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.
3. 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.
4. Convert each 180-dimensional vector u_j into a more compact representation of β dimensions. We give two alternatives here:
   - Comb method: For each 180-dimensional vector u_j, convert it into a β-dimensional vector v_j that represents β different pitch levels: $v_{j,k}$ $(k = 1, \ldots, \beta)$ represents the pitch p_k Hz and is defined by: $v_{j,k} =$

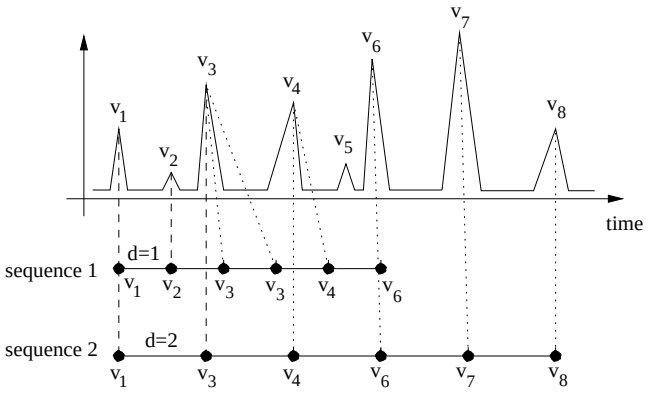

Figure 3: Construction of Characteristic Sequences

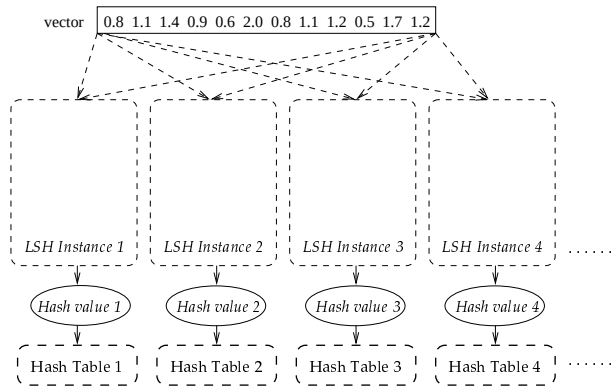

Figure 4: Multiple hashing instances with independent hash tables

$\sum_{i=1}^{6} G_{j,ip_k}$, where G_{j,ip_k} is the frequency component of vector u_j at frequency ip_k Hz. This step effectively convolves the original spectrum with a family of comb-like filters. The result is a list of β-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.

- Cepstrum method: For each 180-dimensional vector u_j, take logarithm and perform another Fourier Transform on it. The result is a list of cepstrum coefficients. We take a subset of β coefficients from it to form a β-dimensional vector v_j.

The final list of vectors v_j, $j = 1, 2, \ldots, n$, are the "event vectors" that represent n events determined by power peaks.

3.1.3 From Event Vector to Characteristic Sequence

Event vectors only give information on instantaneous spectral contents around each event; they do not specify how the music progresses. To capture information on rhythm and music progression, we put together a series of event vectors to construct "characteristic sequences."

When peaks (events) are detected in the previous step, we keep track of time offsets of the n peaks, $t_1, t_2, \ldots, t_n$, and define V_τ to be the event 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 $\tau < t_1$.

With this notation, we 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 3.

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 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 if each v vector is β-dimensional, the dimensionality of each characteristic sequence is βM.

For the example in Figure 3, 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. If each v vector is β-dimensional, each length-6 sequence here is 6β-dimensional. The purpose of having the bracketing control $d \in [1, D]$ is to offset the effects of possible bogus peaks that survived step 2 in Section 3.1.2, such as v_2 here.

3.2 Indexing

Each characteristic sequence generated by Phase 1 is a high-dimensional vector and can be indexed. The design goal of such an index is to facilitate retrieval of "similar" vectors, where similarity is indicated by high correlation along a subset of dimensions. Given a query vector, a Locality-Sensitive Hashing scheme (LSH) [9] is a fast probabilistic scheme that returns approximate matches with controllable false positive and false negative rates.

Figure 4 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. 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. In our hashing design, each raw vector first goes through a simple dimensionality-reduction routine which picks a random subset of its dimensions. Then, 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

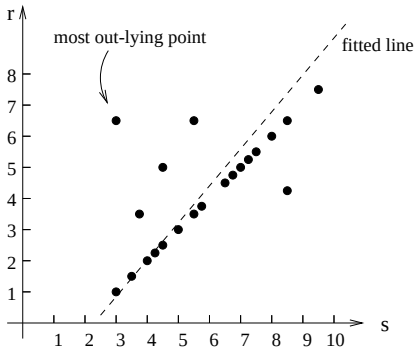

Figure 5: Illustration of Linearity Filtering

each hashing instance, but vary across different instances. If two raw vectors are "similar," i.e., with a high correlation coefficient with sampled dimensions, it can be shown that 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 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 they match in terms of hash values.

In our experiments in Section 4, we will study the effects of adjusting the number of dimension samples, as well as the quantization grid size.

3.3 Alignment Analysis

Each characteristic sequence represents a short segment of a music piece. To find similar music given a query, we break down the query into a set of characteristic sequences and perform an index lookup. Each lookup may generate a set of matches. Our task in this step is to interpret the set of short segment matches to determine which music piece is the "global" match. The key fact to be used here is that a global match must have a set of short segment matches that are well aligned; tempo changes must be uniform in time.

Each match between two music pieces s and r contains a tuple (query-offset, matching-offset) which indicates time offsets of the two matching points. If we plot a 2-D graph with the matching time points of s on the horizontal axis vs. the corresponding points of r on the vertical axis, a "good" match would contain a straight line while a "bad" match would not. Without tempo change, the straight 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.

We discuss two different methods to find out how well two pieces match: Linearity Filtering and Hough Transform.

3.3.1 Linearity Filtering

Linearity Filtering is illustrated in Figure 5. We examine the 2-D graph of matching points, 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 out-lying 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.

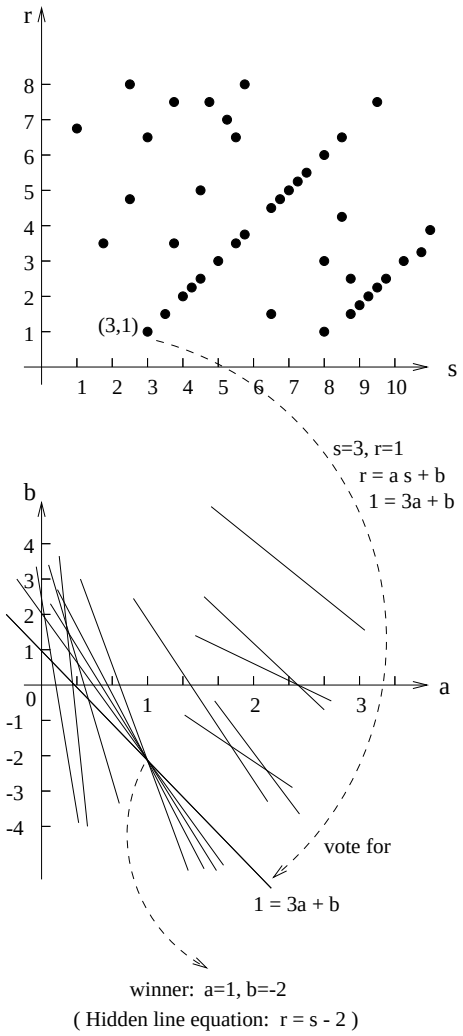

Figure 6: Illustration of Hough Transform

3.3.2 Hough Transform

Hough Transform is a computer vision technique that locates "hidden" lines or shapes from an image [10]. We use it here to help us locate the straight line hidden in the 2-D matching plot.

In Figure 6, 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 parallel 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 step).

To find the hidden straight line from the matching plot, 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. The bottom graph of Figure 6 illustrates the voting process. This vote can also be used to measure confidence in matching these two music pieces.

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.

Five types of similarity are defined according to Section 2.2. 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 returned.

Our result graphs are based on a set of 220 queries. Both classical and modern music queries are used. 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. Due to the lack of standardized testbed collections among music retrieval researchers, we are not able to provide performance comparisons with other systems.

As we discussed in Section 3, the MACSIS framework consists of many modular components that can be independently modified to affect performance. In this section, we focus on the following variable components; each component has a default choice when we modify other components:

- At query time, one way to speed up retrieval is to process only a small sample of characteristic sequences from the query rather than the entire query. The samples can be taken randomly with a fixed sampling rate. We evaluate performance with different query-sampling rates. Default choice: sampling rate = 5%.
- After peaks are extracted during event detection, we compare two methods to construct event vectors: the comb method and the cepstrum method, as described in Section 3.1.2. The dimensionality of event vectors, β, is an adjustable parameter. Default choice: comb method, $\beta = 24$.
- During construction of characteristic sequences described in Section 3.1.3, we adjust the following parameters: sequence length M and bracketing control D. Default choice: $M = 6$, $D = 3$.
- During each LSH instance described in Section 3.2, we adjust the number of dimension samples and quantization grid size. Default choice: number of dimension samples = 8, quantization grid size = 1.2.
- During alignment analysis, we compare two methods: Linearity Filtering and Hough Transform, as described in Sections 3.3.1 and 3.3.2. We study how they respond to changes in query-sampling rate. Default choice: Hough Transform.

Figure 7(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. Five lines in the plot correspond to five different similarity types. Figure 7(b) shows the corresponding execution times in seconds for the complete set of 300 test queries. The "sampled" query approach results in a significant speedup, while not sacrificing much in performance when the sampling rate is above 5%.

Figure 8 compares the comb method and the cepstrum method during event vector construction, while varying dimensionality β of event vectors. In general, the comb method performs better and faster than the cepstrum method. With the comb method, as event vector dimensionality is increased, accuracy goes up and then slightly down, but the difference is not significant. We set the default dimensionality to be 24, where the average retrieval accuracy is high.

Figure 9 studies the effects of changing M and D values during construction of characteristic sequences. As D goes up, retrieval accuracy improves, but execution time also increases, due to the increased number of characteristic sequences to be processed. As a compromise, we set the default D value to be 3, where execution time is not too high and retrieval accuracy is reasonable. As M increases, execution time decreases slightly because the total number of valid characteristic sequences goes down. However, a larger M means less tolerance to music variation and noise. On the other hand, a very small M may tolerate too much variation and cause different music pieces to be returned as similar. As a result, we choose 6 as the default value for M.

Figure 10 shows the retrieval accuracy and time when adjusting indexing parameters within LSH instances: number of dimension samples and quantization grid size. A smaller number of dimension samples leads to higher retrieval accuracy, because of the higher likelihood that two similar characteristic sequences get hashed into the same bucket. However, execution time skyrockets as the number of dimension samples is reduced, since the hash structure is no longer effective when too many items fall into the same hash bucket. Same for the quantization grid size: a larger grid tolerates more variation and makes similar sequences more likely to be hashed into the same bucket. So a larger quantization grid size leads to higher retrieval accuracy but longer execution time. On the other hand, when the number of dimension samples is set too small or the quantization grid size is set too large, accuracy begins to suffer, since dissimilar characteristic sequences start to appear "similar." As a compro-

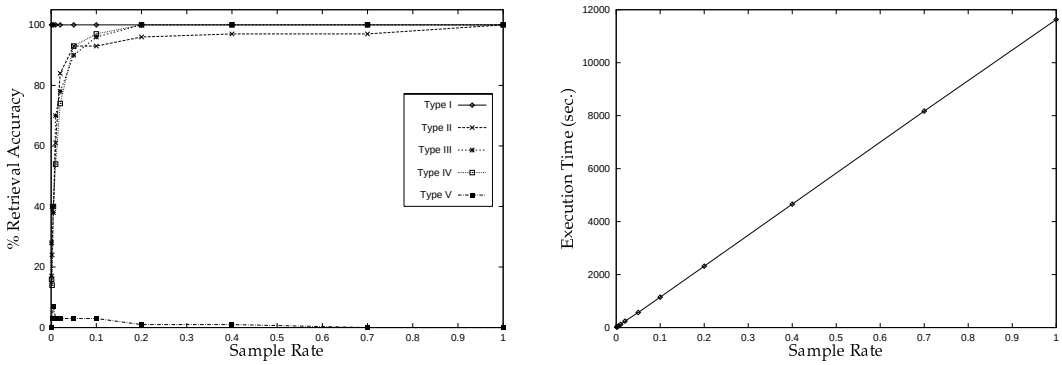

Figure 7: Retrieval accuracy and time with sampled queries

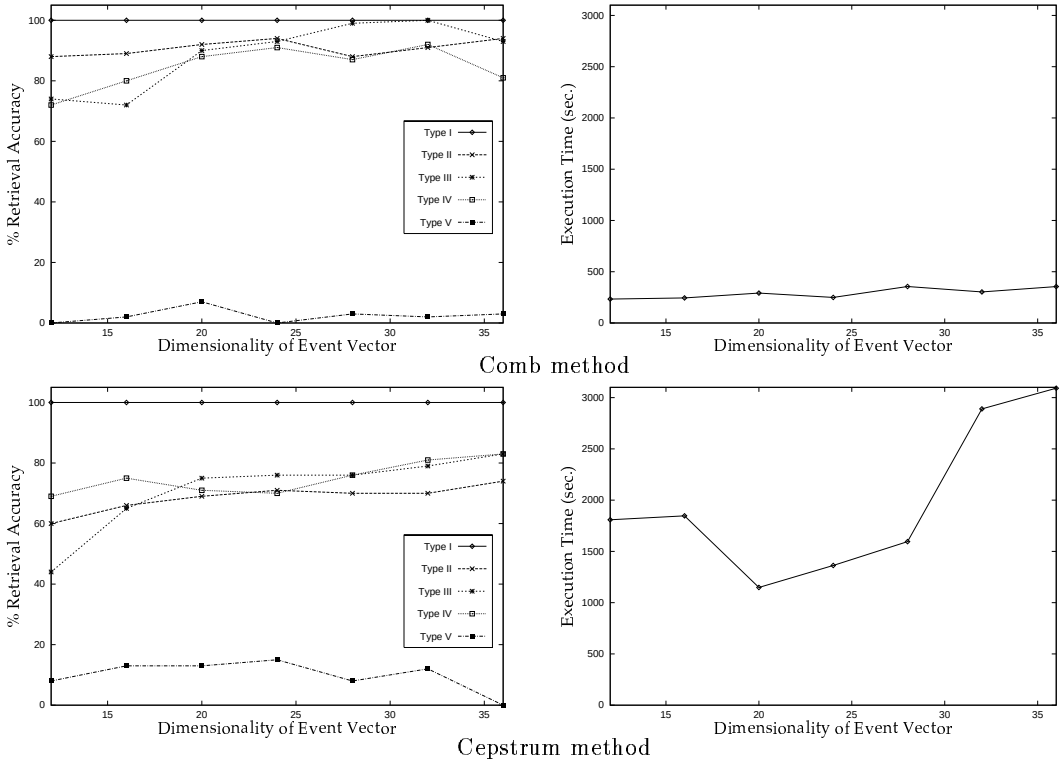

Figure 8: Retrieval accuracy and execution time for comb and cepstrum methods

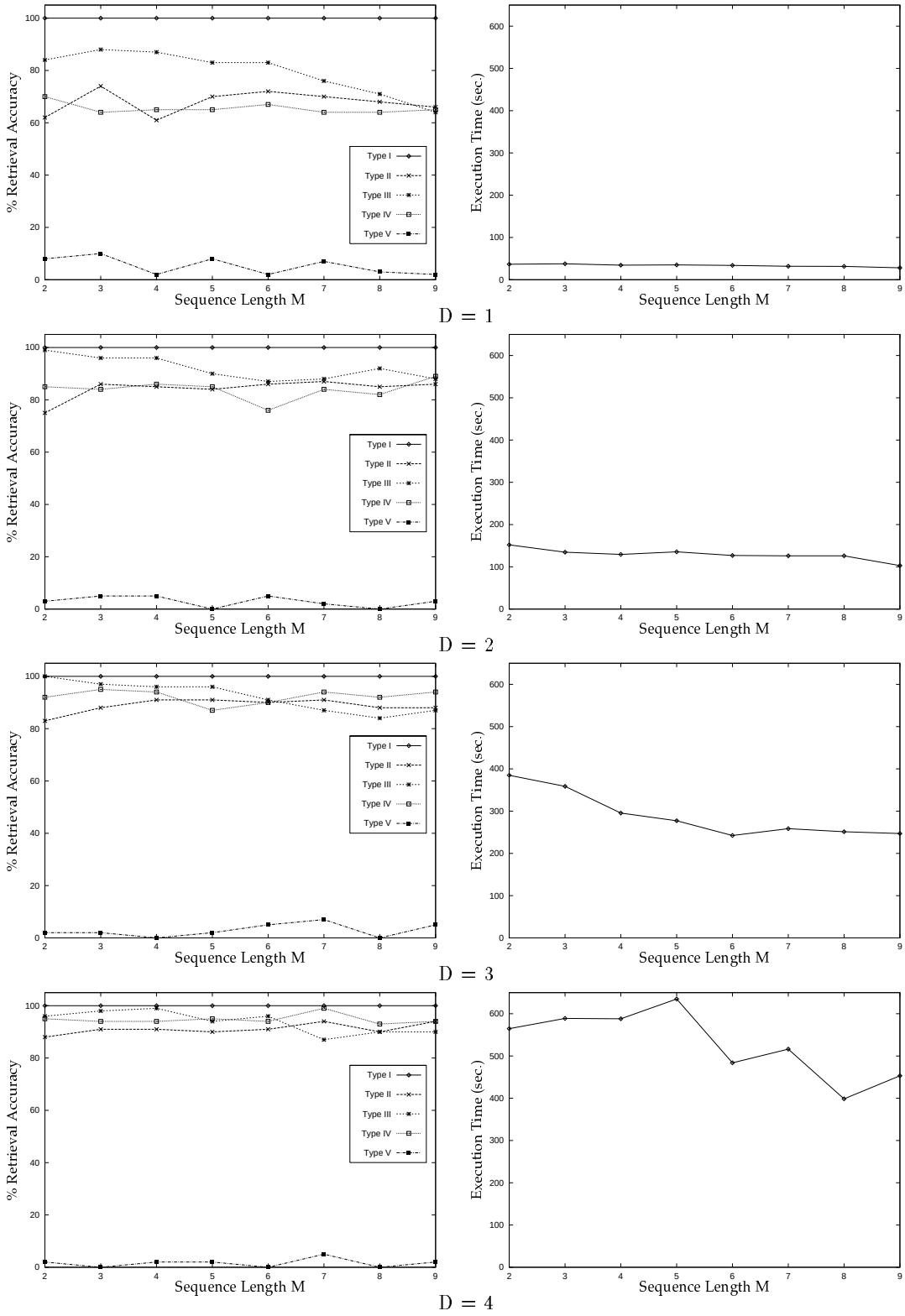

Figure 9: Retrieval accuracy and execution time for different M, D values

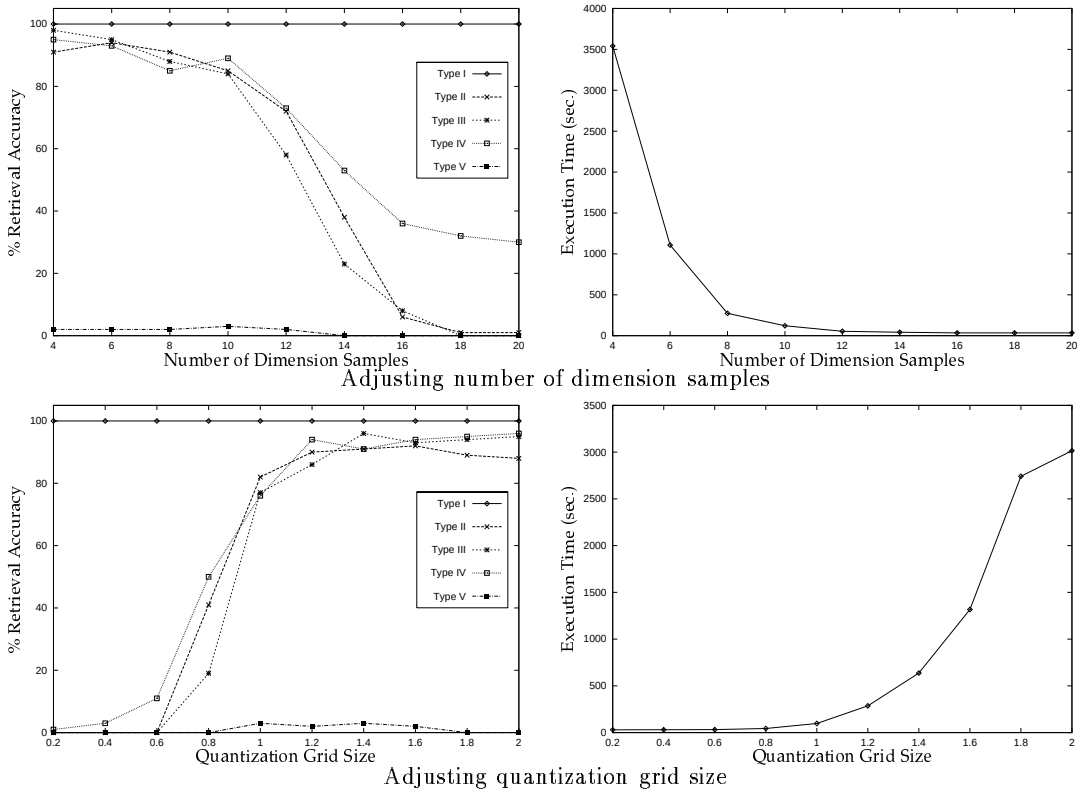

Figure 10: Retrieval accuracy and time for different indexing parameters

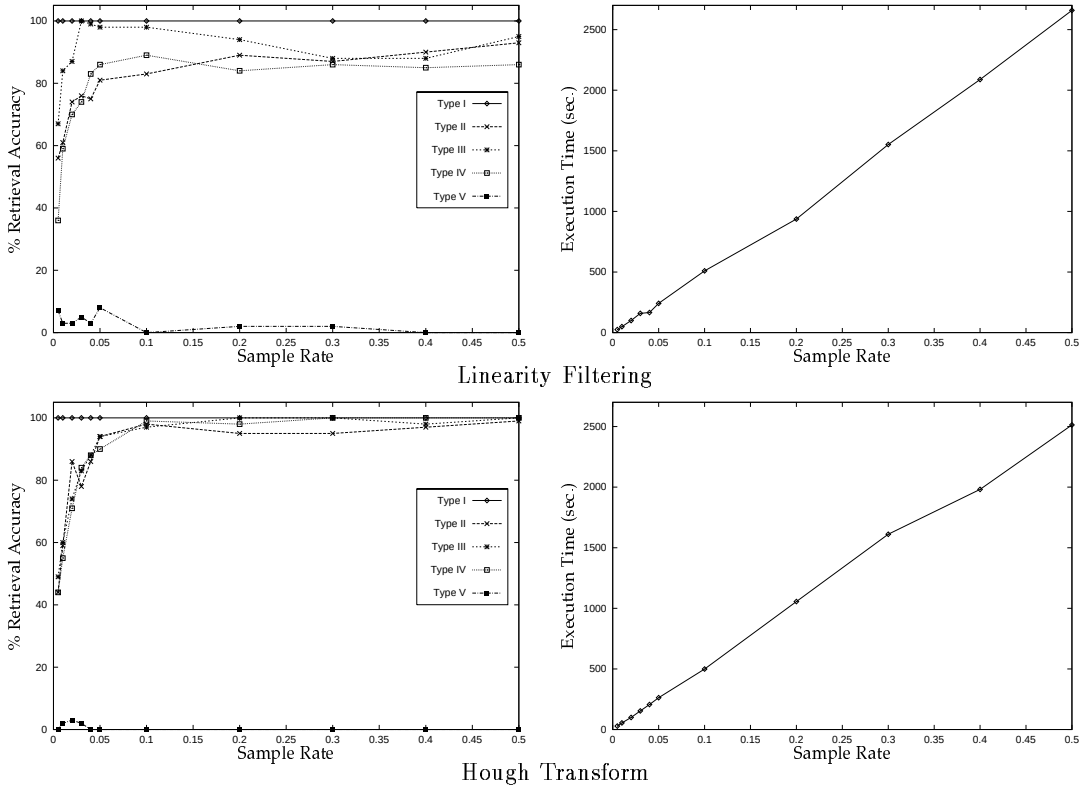

Figure 11: Retrieval accuracy and time for different alignment analysis methods

mise between retrieval accuracy and execution time, we set the default number of dimension samples to be 8, and the default quantization grid size to be 1.2.

Figure 11 compares Linearity Filtering with Hough Transform at different query-sampling rates. The two methods take roughly an equal amount of time to execute. In terms of retrieval accuracy, the Hough Transform is better, at almost all sampling rates. This difference is partly due to the fact that Linearity Filtering is sensitive to outlying points in the matching plot, and may not be able to locate the hidden straight line when too many random points exist. Hough Transform is more robust to this problem.

In most of our experiments, 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.

5. SUMMARY AND FUTURE WORK

We have presented our MACSIS spectral indexing framework to perform content-based music retrieval on acoustic data with acoustic queries. The framework involves multiple modules that can be implemented in different ways. A set of experiments have been conducted to study the effects of various design choices. Experiments have shown that proper choice of these modules can lead to an efficient system which detects music content similarity while tolerating tempo changes, some performance style changes and noise, as long as the changes are not too much and the different performances are based on the same score.

Further study is desired to find ways to automate 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; more elaborate matching methods in phase 3 could lead 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.

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

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

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

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

[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, "Efficient Acoustic Index for Music Retrieval with Various Degrees of Similarity", in *Proc. ACM Multimedia*, 2002.

[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.