Eyes4Ears - More than a Classical Music Retrieval System

Dirk Habich and Wolfgang Lehner
Dresden University of Technology
Database Technology Group
01062 Dresden, Germany
{dirk.habich,lehner}@inf.tu-dresden.de

Alexander Hinneburg
Martin-Luther-University of Halle/Wittenberg
Database and Data Mining Group
06099 Halle,Germany
hinneburg@informatik.uni-halle.de

Philip Kitzmantel and Mathias Kimpl
PXP Software AG
Vienna, Austria
{philip.kitzmantel,mathias.kimpl}@pxpgroup.com

Abstract

Content-based similarity search for music retrieval attracted a lot attention in recent information retrieval research. Most music applications (e.g. several commercial web portals) offer to search music files, which however is limited to key-word-based search on subjects like genre or artist. Other similarity search approaches base on abstract metrics, which are defined on feature vectors representing psycho-acoustic or physical properties. However, it is still an open problem to adapt the search towards the user own music similarity measure.

This paper presents a new music retrieval system which is more than a classical music management system. The system architecture is based on a client/server architecture and offers methods to manage a large music database efficiently. In addition to key-word-based search methods the Eyes4Ears system provides also methods for content-based similarity search. Furthermore, we report an attempt to cross the semantic chasm between user and search system by an adaptive retrieval approach.

1 Introduction

The development and the widely-spread usage of the MP3 format and other digital audio formats initiated a change of managing the personal music collection. Instead of putting CDs into a shelf nowadays many people store their own music collection in file systems of their computers. So when looking for suitable music in a particular situation, they have to browse through directory listings of file names instead of orienting themselves according to the cover pictures at the CD's envelopes.

A more modern management of digital music files can be done with special software or with extended music players, like Winamp[1] or Windows Media Player[2]. In this case, the included meta information in music files is used for the organization and this approach enables an efficient key-word-based search mechanism for music files. Also this approach works well for small music file collections, the usage of a relation database system is essential for large music collections. The advantage of database systems is that they provide functionalities to deal with large data sets as well as efficiently search methods within the data sets.

The key-word-based search is not always sufficient to answer all possible user questions, e.g. music similarity. Content-based similarity search in music data is an active research area since nearly a decade. Such a search approach extends the classical key-word-based search, so that the users can find pieces of music on the basis of a given query music file. In particular, content-based similarity search enables the discovering of new music files for the user, which are interesting for him, because they match to him music taste.

While in the beginning mostly symbolic music representations like MIDI files has been studied, recently also raw audio data were used for content-based similarity retrieval. Most approaches dealing with the latter form of music data rely on some features extracted in advance from the files. These feature sets can be classified into physical and psychoacoustic feature sets. The first category of feature sets describes the music at the physical level in terms of frequency, amplitude and several derived measurements. Psychoacoustic features try to represent what a human recog-

[1] http://www.winamp.com
[2] http://www.microsoft.com/windows/windowsmedia/default.aspx

nizes, when hearing music. These features are more expensive to determine, but reveal the advantage of being less sensitive to noisy elements in the data, which a human does not recognizes or which the user can filter out. However, there still remain subjective factors of what music is considered similar by a human.

Relevance feedback is a well known technique from information retrieval to iteratively adapt a systems' similarity measure towards the users' needs. In general, during the feedback process firstly the system produces an initial result, from which the user may pick relevant records. Then the feedback is used by the system to come up with a revised version of the result set, which is assumed to contain the previously selected records but also some new records, which might be also relevant to the user. Then the next iteration starts until the iterative process converges. Beside several technical things, regarding how the feedback is used by the system to determine a new query, there is a main problem in the application of the relevance feedback idea to music retrieval, namely how to pick the relevant music files efficiently.

Our Contribution

In this paper we present our Eyes4Ears music application server allowing managing large music collections. In addition to the meta information of music files we store also content-based feature vectors in a database. The Eyes4Ears application server is based on the Client/Server architecture. The server offers methods for key-word-based as well as content-based search. The provided methods are published as Web Services, so that a wide range of client applications can use these services. In this paper we will mainly focus on the following issues:

- We explain a new feature extraction technique, which is used to describe the content of music files. The feature vectors are then used for music similarity search.
- We propose a novel adaptive relevance feedback technique, which is based on interactive genetic algorithms.

The reminder of the paper is structured as follows. In section 2 we discuss related work and in section 3 we shortly present our developed Eyes4Ears application server. The feature extraction method as well as a basic approach of content-based retrieval is pointed out in section 4. Section 5 outlines our adaptive music retrieval approach. Furthermore, we present in section 6 future work and then we conclude the paper.

2 Related Work

The published works about music include music analysis, classification as well as content-based similarity search. An early overview about audio information retrieval is given in [4]. Audio information retrieval systems differ in the kind of the underlying music, which is raw audio data (MP3,WAV,...) and symbolic data (e.g. MIDI representation) and in the way of querying such a system. As most of music is not available in symbolic representation we concentrate in our work on the retrieval of raw music data.

Music classification includes techniques to automatically categorize music in various genres and has a long history from speech recognition. Tzanetakis and Cook [17] described an automatic classification method for a hierarchy of musical genres. They used three feature sets to represent timbral texture, rhythmic content and pitch content. Foote [3] constructs a tree-based vector quantizer with the help of mel-frequency cepstral coefficients (MFCC), which have been widely used in speech recognition. For each audio signal a histogram of the relative frequencies to each class is constructed. These histograms are also used for retrieval and allow retrieving audio documents by acoustic similarity.

In the MuscleFish approach [18] raw data sound files are analyzed and a specific set of psychoacoustic features are derived from them. Such a feature vector includes attributes like loudness, pitch, bandwidth and harmonicity. For retrieval a distance in the feature space is defined, which is computed between a given query vector and all feature vectors in the music database. Finally records are ranked by distance (closer means more similar). The MuscleFish system is a so called query-by-example system, because it retrieves similar music for an example sound file from the music data base. ARTHUR [5] is another query-by-example retrieval system working on raw data that retrieves similar music on the basis of long-term structures. The long-term structure is determined from the envelope of audio energy versus time in one or more frequency bands.

SEMEX [13] and C-BRAHMS [12] are also query-by-example retrieval systems, which, however, work on symbolic music data, e.g. MIDI representation. The other way to query an audio retrieval system is to hum the music. In this case the systems are called query-by-humming, like those described in [8, 11]. The systems work on symbolic music data.

All the mentioned audio information retrieval systems do not consider different user preferences, because each system determine the similar records independently from the user. However, users may have different emotions to the same music, so for each user the similarity measure for music files may be different. Recently Hoashi et. al [10] proposed a system adapting known relevance feedback tech-

niques to music retrieval. As it may be tedious for the user to pick relevant results especially from music, s/he have to listen at least a part of each music file, Hoashi el. al proposed a method to generate user profiles based on genre preferences, which are refined by the relevance feedback. We propose an orthogonal method to ease relevance feedback by proposing a novel screen space-saving visualization technique, which allows the user to visually compare music files.

The aim of some works about music analysis were to produce summary excerpts, thumbnails of music [2] or music snippets [14]. This works can also be used to ease relevance feedback, but listening of the thumbnails or snippets is still necessary.

There are many different approaches to visualize raw music data. The audiogram of the waveform is well known, where time is on the x-axis and amplitude is on the y-axis. Another visualization is the spectrogram. In this case the amount of frequency y at time t is encoded by the brightness of the pixel at the coordinate (t, y). The darker the pixel, the more that frequency contributes to the signal at that time. This representation can be computed using the Fourier transformation (see section 4 for details).

A new alternative is the comparisonics waveform display[3], where the audiogram of the waveform contains additional information. The commanding frequency at time t is represented through different colors. Foote and Uchihashi [6, 7] proposed the beat spectrogram which graphically illustrates rhythmic variation of time. Pampalk et. al [15] also used rhythm patterns to estimate the similarity between music files and subsequently to train a self organizing map, which is used to visualize a whole music collection as *Islands of Music*. The produced visualization yields an interesting overview of a collection of music files. Performance visualization [9] generates a three-dimensional scene, which shows different elements of the music. A drawback of most of the music visualizations is their dependency on time, which causes the visualizations to have different sizes and makes the graphical comparison of music files difficult. To allow the comparison of many music pieces at the same time without scrolling we propose a space-limited music visualization to efficiently use the available screen space.

3 Eyes4Ears-System

The overall aim of the Eyes4Ears project is on the hand the development of a music management server allowing handling a large music collection efficiently. On the other hand, the server should also provide effectively classical key-word-based as well as the content-based search methods of music objects. In an client/server environment, one

[3] www.comparisonics.com

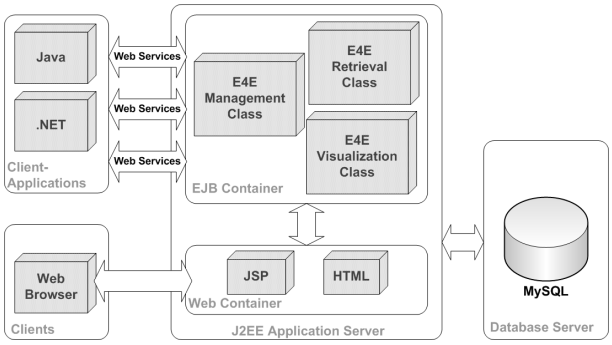


Figure 1: Eyes4Ears system architecture

of the most important requirement is the capability to enable a large number of users/clients to use the provided services. This requirement implies further:

- Concurrent access on data: The Eyes4Ears methods operate on the same data and this concurrent access should be done without any anomalies.
- Transactional Behavior: Either the whole action, e.g. insert new music object in the database, takes place or nothing.
- Access Control: In such an application context the concept of identity is essential.

For this reason we decided us to realize the Eyes4Ears-System on the basis of the J2EE platform, which supports the development of enterprise applications with multiple tiers. The multiple tiers are typically: *Client Machines*, *J2EE Application Server* and *Database Server*. The above considered system-level issues are handled by the J2EE platform and the developer can focus on solving business problems. The corresponding Eyes4Ears architecture is illustrated in figure 1. We use the Open-Source application server JBoss[4] as J2EE platform.

The J2EE application server consists of two containers, a Web container and an EJB container. The Web container handles HTTP requests, and retrieves and delivers HTML documents as a response by means of the methods in the EJB Container. The Web container can be seen as a presentation manager for web browsers. The components inside the EJB (Enterprise JavaBean) container represent the application-specific business logic. The container provides services like lifecycle management, security, transaction management, concurrency and many more. The Enterprise JavaBeans components can be distinguished in three different types: *entity*, *session* and *message-driven beans*.

The entity beans represent entities that are stored in persistent storage, such as a database. Every entity bean has a

[4] http://www.jboss.org

uniquely identifier by a primary key. In our application context entity beans are e.g. Authors, Albums and many more. As persistent storage system we use a MySQL[5] database system. Session beans manage processes and tasks. The Eyes4Ears session beans can be classified as follows:

- *E4E Meta Management Class*: The methods in this class assume all tasks, which are responsible of meta information management of music objects. A typical method is *insertMusicObject(String fileName)*. The method determines the meta information of the specified music object. If the music object does not exist in the database, the new object will be stored. Furthermore, the class also provides efficient key-word-based on subjects like genre or artist.

- *E4E Retrieval Class*: Our content based retrieval approaches are represented by this class and therefore provides corresponding methods. Sections 4 and 5 will deal with these approaches.

- *E4E Visualization Class*: This class provides methods for visualization of music objects, which are then used in the iterative and interactive retrieval of music objects.

The lifecycle of session beans are dependent on a session between a client and a server. While session beans provides remote interfaces that define which methods can be invoked, a message-driven bean subscribes to or listen for messages. Therefore, message-driven beans are a mechanism to process asynchronous messages. An invocation of server methods at a client is done through some kind of remote method invocation (RMI) protocol. The drawback of the existing RMI protocols like Java RMI and CORBA is that these technologies are not truly platform-independent. For example to use Java RMI, you need a Java virtual machine and the Java Programming language.

Web Services represent a new paradigm in the distributed computing. They are XML-based, independent and modular applications, which are published and usable over the network. That implies that applications or other Web Services can use services over the network without to explicitly integrate them. Web Services are truly platform-independent and enterprise beans can be exposed as Web Services. The Eyes4Ears system provides their methods as Web Services and these Web Services can be invoked by other J2EE applications as well as applications which are written in other programming languages on a various platforms.

Existing Client applications, like Windows Media Player or Winamp have now a simple standardized interface to invoke Eyes4Ears methods. In particular the content-based retrieval of similarity music objects is an interesting feature and this extends the functionality of such clients.

[5]http://www.mysql.com

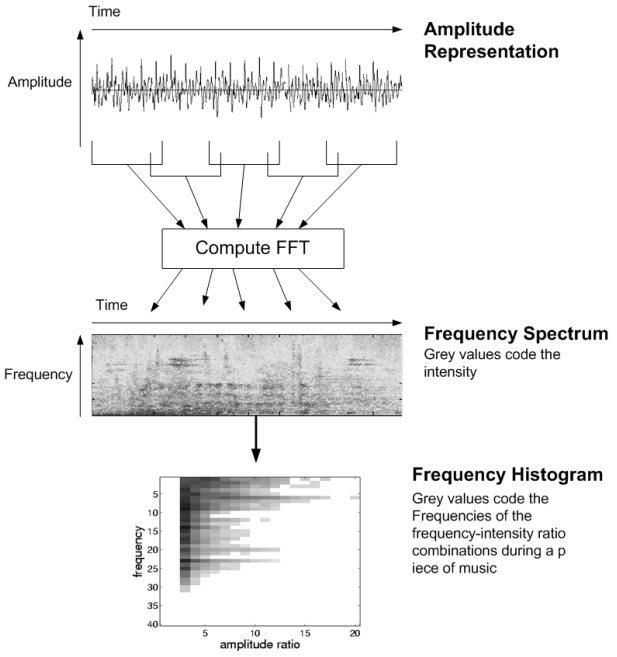
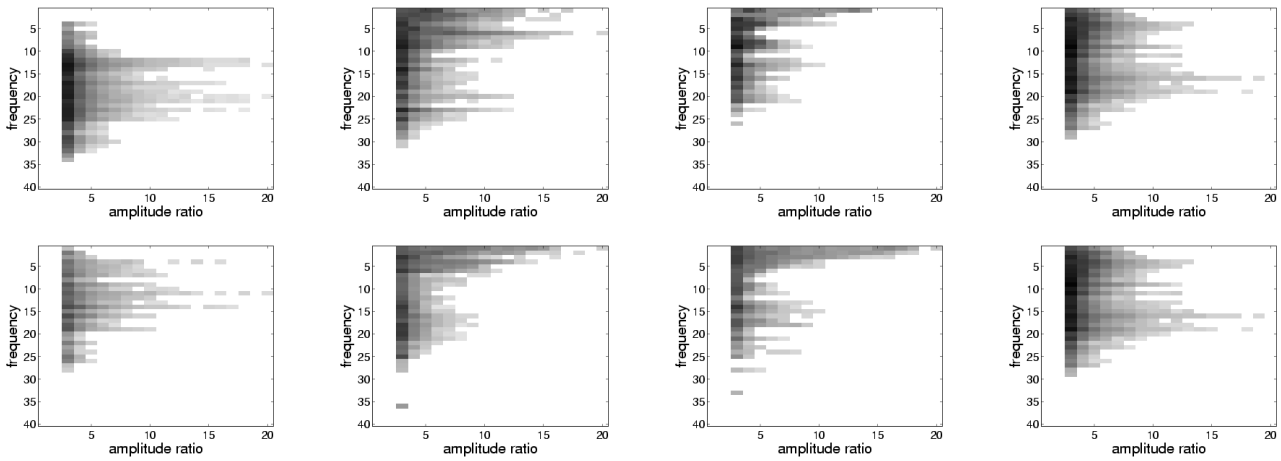


Figure 2: The figure shows an overview about the transformation procedure of the audio data into a length independent vector representation. The result of the transformation is a histogram, which describes the frequencies of combinations of a particular frequency and an intensity value. The histogram counts the occurrences of all combinations during a piece of music.

4 Content-Based Similarity Search

By nature, pieces of music are continuous signals, which firstly have to be digitalized to process them by computers. For the digitalization the signal is scanned at discrete points in time and measured analogous values are stored as digital numbers. The time between the scan points determines the sampling frequency and the precision of the digital number for the amplitude signal is called quantization. For the right recognition the sampling frequency of a signal has to be at least twice as large as the maximum frequency in the signal. As the human ear can detect frequencies from 20 Hz to 20 kHz, high quality audio data are usually sampled with 44,1kHz. The result of the digitalization is a time discrete signal s, which is stored on the computer using various music formats like WAV or MP3, whereas MP3 additionally compress the signal.

Discrete Signals can be characterized and decomposed using the discrete Fourier transformation. The intuition is to decompose an arbitrary signal into several sinus signals of different frequency, amplitude and phase. For a discrete signal $s(n)$, which is defined on the time points



(a) Classic (b) Pop (c) Techno (d) Musical

Figure 3: The figures (a-d) show our frequency-ratio histogram for two music file for each genre.

$n \in \{0, 1, 2, \ldots, N-1\}$, the N-point discrete Fourier transformation is defined as:

$$S(f) = \sum_{n=0}^{N-1} s(n) e^{-j\frac{2\pi}{N} \cdot n \cdot f}$$
$$\texttt{with } f = 0, 1, \ldots, N-1 \texttt{ and } j = \sqrt{-1}$$

The fast Fourier transformation (FFT) is a complex algorithm, which reduces the run time complexity of the discrete Fourier transformation from $O(n^2)$ to $O(n \cdot \log n)$ under certain conditions.

As the Fourier transformation of a whole music file delivers only the average frequency distribution and even the FFT of a music file consumes a lot of CPU time, the frequency decomposition of audio data is modified as following. First the audio signal is disassembled into m overlapping time windows of a certain length (e.g. 16 millisecs). Then for each window the discrete Fourier transformation is applied. As sharp cutting at the window borders would lead to artefacts in the frequency distribution, the signal is fade in and out at the window borders. For this purpose typically the signal within a window is multiplied by the Hamming-function [1].

The FFT of overlapping time windows delivers a sequence of frequency spectra with intensity values for each frequency. This very detailed representation of a music file is unsuitable for efficient similarity search because of the large size. It is desirable to have a compact representation, whose size is independent from the length of music files. Figure 2 shows an overview of the transformation of the music file towards the feature vector.

To get coarser frequency intervals the range from 20Hz-22kHz is logarithmically divided into k intervals. The logarithmic intervals are chosen due to fact that humans recognize frequencies in a nearly logarithmic way [16]. The intensities are transformed into ratio values of the maximal occurring intensity in each music object. This transformation considers the fact, that not all music objects sound equally loud. The intensity ratio value range $[0, 1]$ is divided as well into l intervals. To average the audio data over time we use a two-dimensional histogram with k columns and l rows. Typical values are $k = 40$ and $l = 20$. To fill the histogram the time-dependent spectrogram from the FFT is traversed according to the time axis and the occurrences of the particular combinations of frequency and intensity ratio are counted. More formally a histogram h for a music file which is split into m windows for the FFT is defined as matrix:

$$h = \frac{1}{m} \cdot [h_{i,j}] \text{ with } 1 \leq i \leq k \text{ and } 1 \leq j \leq l$$

where $h_{i,j}$ is the count of the occurrences of frequency interval i with intensity ratio interval j. For each music object we save the histogram as well as the maximal occurring intensity value.

The figure 3 shows histograms of different pieces of music. To make the two-dimensional frequency-intensity ratio histogram as compact as possible we map the frequencies to gray values. Due to the very skewed frequency distribution, that means some values occur very often, we visualized the histogram h with different squared frequencies like $h^{1/2}, h^{1/4}$ resp. $h^{1/8}$. The version based on $h^{1/4}$ shows the best contrast for most of our test music objects, therefore we used this data transformation for visualization. The test music objects in the figure 3 illustrate similarity and dissimilarity of pieces of music of different genres.

The histograms characterize the content of music files in a compact way and therefore they are now the basis for a

efficient similarity metric for music files. The distance between two frequency histograms a, b can be measured using the Euclidean metric as follows:

$$dist(a,b) = \sqrt{\sum_{i=1}^{k}\sum_{j=1}^{l}(a_{i,j} - b_{i,j})^2}$$

This similarity metric can be expressed as SQL statement, so that the determination of similar music object to a given query piece of music is done by the database system. The drawback of this approach is that each user gets the same result for a music object. Normally, each user has its each preference to each music file that means a similarity search should determine a user-dependent result for each user. To achieve that, we assume that not all counts in the histogram may be of equal importance. To approximate this effect we use the generalization to projected nearest neighbor search [**?**]. Genetic search is used to find good projections, where the nearest neighbors are more meaningful than those found by the full-dimensional metric. However, due to the very large dimensionality ($d = 20 \cdot 40 = 800$) this method is not effective. To make the genetic search working we propose to restrict the search space of projections, so that the all intensity ratios of the same frequency are included in a projection. In that way we have to search only for meaningful frequency intervals.

5 Adaptive Music Retrieval Approach

The goal of every similarity search is to find records from a collection having the user in mind while submitting the query. Despite the goal may be not perfectly achievable, due to the limited representation of the database objects, it is still desired to approximate the goal as far as possible. It is natural to formulate such a problem as optimization task, which is in our case to find a subspace defined by a projection P, in which the objects relevant to the user are the closest objects to the query.

To describe our adapted similarity search method in a more formal way, we firstly need a representation for a projection of the high dimensional feature space. As we stick to axis-parallel projections restricted in the way described above, we need for the representation of a $d' \cdot l$-dimensional projection P with $d' \in \mathbb{N}, d' < k$ exactly d' integer values $\{i_1, \ldots, i_{d'}\} \subset \{1, \ldots, k\}$, which are the indices of the frequencies included in the particular projection. So the distance defined according to a picked projection P is:

$$dist_P(a,b) = \sqrt{\sum_{i \in \{i_1,\ldots,i_{d'}\}}\sum_{j=1}^{l}(a_{i,j} - b_{i,j})^2}$$

This measure can also be seen as a special case of the Mahalanobis distance, which is defined as quadratic form

$$dist_{Mahalanobis}(a,b) = \sqrt{(a-b)M(a-b)^T}, \quad a, b \in \mathbb{R}^{k \cdot l}$$

where M is a matrix with $k \cdot l$ rows and columns. In our case the matrix has to be a diagonal matrix with ones at the positions of the picked frequencies with all associated intensities.

A very flexible classes of optimization methods are genetic algorithms (GA). Inspired by evolution theory genetic algorithms iteratively search the space of possible solutions in parallel in several regions. Therefore, initially a set of possible solutions G_0, called population, is generated. In iteration i, each solution $s \in G_i$ is evaluated by determining its fitness value. Thereafter solutions are picked with probability proportional to the fitness and new solutions are generated from the picked ones by applying crossover operators. A crossover operator recombines properties of the parent solutions. As the solutions with higher fitness are more likely picked for recombination the iteration converges.

A possible solution to the similarity search problem is any particular axis-parallel projection of the $k \cdot l$ dimensional space. As pointed out in section 4 such a large number of parameters may be to much for genetic algorithms, it is reasonable to restrict the projections, that for each picked frequency interval all intensity levels are included in the projections. Therefore a projection P is given by a subset of $\{1, \ldots, k\}$. To make a projections representation more suitable for genetic algorithms, we code the integer values set of relevant frequencies as bit string of length k.

The initial population is a set of randomly generated bit strings each determining a particular projection. After evaluating the fitness of each projection, a new population is generated using crossover operators. In our case we tested several types of crossover operators, namely one-point, two-point and multipoint crossover. Each crossover operator takes two bitstrings and also outputs two bitstrings. The one-point crossover operator randomly determines a position i, which partitions each of two input strings in two halfs, say $a = a_1 \ldots a_i a_{i+1} \ldots a_k$ and $b = b_1 \ldots b_i b_{i+1} \ldots b_k$. Then two new strings are derived by exchanging the first part to form the two output strings, $a' = a_1 \ldots a_i b_{i+1} \ldots b_k$ and $b' = b_1 \ldots b_i a_{i+1} \ldots a_k$. Two-point and multi-point crossover works similar, the former determines two split points and forms the output stings by exchanging the middle part. The latter operator determines several exchange positions and forms the output by swapping the bits at the determined positions. In our tests we found single-point crossover performing most suitable, as the changes in the similarity measure are more smooth than those performed by the other operators.

In our case the fitness corresponds to the degree the similarity measure defined by a particular projection matches

the user's intensions and needs. Due to the diversity of human perception of music and different possible intensions for submitting music queries, determining fitness of a similarity measure in an automatic way is nearly impossible. However, the user can judge weather the result of similarity search is relevant or not. So we use the user's feedback to determine the fitness of a projection. But unfortunately the typical user can not evaluate the fitness of a projection, when only the indices of the involved frequency intervals are presented as a set of numbers. A better way is to perform the similarity search for each projection in the population and to present the top results of each search. This maps a projection to an ordered set of music pieces, which can be much better evaluated by the user. However, in case of music files the selection task is still very tedious as many music files have to be at least shortly listened. This is especially a problem, when title and short description of the music file are not expressive.

To ease the selection of relevant results we propose a visualization of content-based features of music files. The goal of the visualization is to enable the user to quickly compare the results found by similarity search and to give him an advice, which search comes close to her/his intuition. In the moment we use our visualization technique, which is described in section 4 and we mark the projections.

The following algorithm summarizes our framework.

Algorithm 1 Visual Music Retrieval Framework

Require: set of data vectors D, query point q, n number of points returned by a similarity search

1: Initialize randomly a set of projections G_0, $i \leftarrow 0$
2: **repeat**
3: **for all** $P \in G_i$ **do**
4: X$\leftarrow$ n nearest points to q out of D using $dist_P$
5: Draw a visualization for each music piece corresponding to a point in X
6: **end for**
7: $R \leftarrow$ the user picks the relevant projections
8: $G_{i+1} \leftarrow \{$recombinations of elements in $R\} \cup R$
9: $i \leftarrow i + 1$
10: **until** User is satisfied

6 Future Work

In this section we want to outline some ideas for future work. The framework described above works well for comparing whole music files. However, it may be meaningful to the user to look for a partial matching which can be motivated by inhomogeneous parts in music files. Often one part in a music file is very interesting for the user, so this part should be used to search music files which include similarity parts. One possibility for the user to express her/his intuition of the query is to explicitly specify the interesting part.

As we know at this point that not the whole reference music file is used for similarity search we have also to look for the best matching parts of the files in the resulting ranking. To enable this feature we need to store for each music file not only a single frequency histogram as introduced in the section 4 but a sequence of cumulative non-normalized histograms. For this we choose the time interval t between two consecutive histogram and we generate the non-normalized histograms by averaging over the intervals $[0, t], [0, 2 \cdot t], [0, 3 \cdot t] \ldots$ until the whole music file is finished. For each interval we store the non-normalized frequency histogram h^N. With this description it can be assumed, that only parts can be specified starting at the beginning of the music files. But users may not only interested in time intervals of length t. As we store non-normalized frequency histograms we can also compute frequency histograms for longer time intervals. Let be m_t the number of FFT windows per interval t, so the partial frequency histogram for the interval $[s \cdot t, e \cdot t]$ with $s, e \in \mathbb{N},\ s < e$ is defined as:

$$h_{e-s} = \frac{1}{m_t(e-s)} \cdot \left(h_e^N - h_s^N\right).$$

The time interval of the reference music file the user selected, is extended to have a length which is a multiple of t. The selected time interval q has therewith a length of $n \cdot t$ with $n \in \mathbb{N}$. So the similarity between the specific part q of the reference music file and a music file x in the database is not easy to determine. Similarity parts can occur at different placement that means at different time intervals in various music files. Therefore we compare the specific part q against all possible parts of a music file in the database by shifting q over the music file $[0, q], [t, q+t], [2 \cdot t, q+2 \cdot t], \ldots$ until the whole music file is finished. To compare with all possible parts we shift q by time interval t over the music files. The part with the smallest distance is considered to be the best matching part. Only this part is included into the final ranking of best matching files. Note, that at this stage of the project we are only interested in improving the effectiveness of the of the retrieval method.

Partial matching search is especially relevant for very inhomogeneous music pieces. The identification and the usage of the inhomogeneous music pieces in the retrieval process can improve the effectiveness the content-based search. Using our music visualization technique introduced in section 4 we show in figure 4 a sequence of consecutive intervals of 10 seconds length of a music piece. The visualization looks quite different due different themes of the music.

Furthermore, we plan to extend our visualization as well as our retrieval engine with rhythmic information of the music files. We also investigate wavelet transformation and

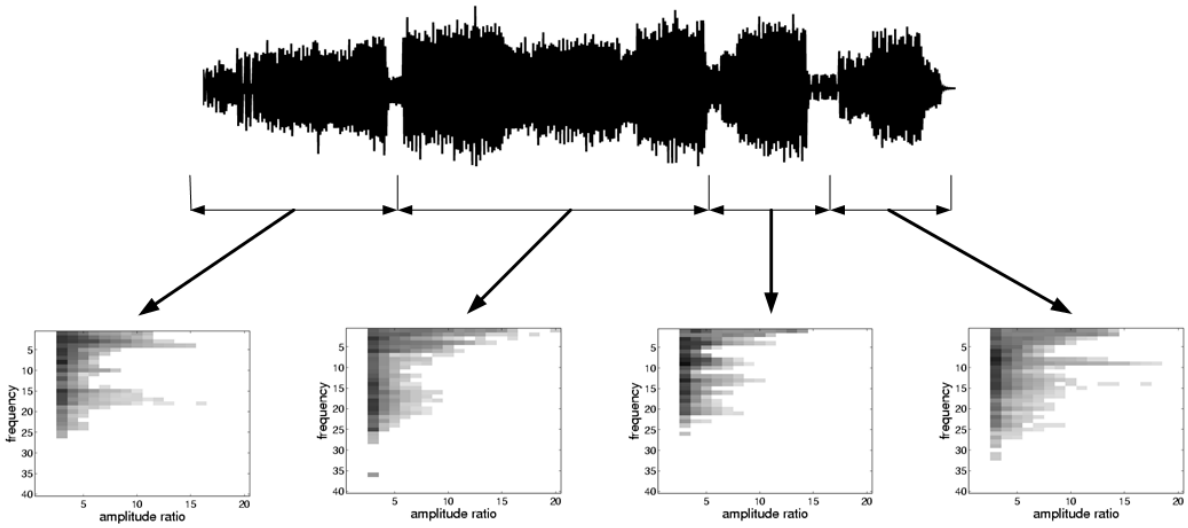


Figure 4: Visualization of several parts of a music file

want to extract more information at different granularities about the content of music files. Moreover, we want to efficiently support the music similarity search with database operators. The overall aim of the ongoing work is to develop a content-based music similarity search system, where the user can visually search music files without the need to explicitly expressing him/her emotional intuition about the query music file.

7 Conclusion

In this paper we present our Eyes4Ears music application server. In addition to the meta information of music files we store also content-based feature vectors in a database. The server offers methods for key-word-based as well as content-based search. The provided methods are published as Web Services, so that a wide range of client applications can use this services. The main part of the paper focused on our adaptive music retrieval approach. Our system allows the adaptation of the similarity measure to the user's intuition of the query and we are now able to determine for each user an individual result. To enable this feature we developed a new feature vector (frequency histogram) and use interactive genetic algorithms to translate the users intuition into terms of frequency intervals. Our interactive system solves one main problem, which is to enable the user to quickly pick relevant result, by offering visualizations of pieces of music. In combination to listing some pieces of music our visualization technique eases the judging the relevance of the results not heard so far. Furthermore, we outline some ideas for future work.

References

[1] L. Rabiner abd B.-H. Juang. *Fundamentals of Speech Recognition*. Pretence Hall Inc., 1993.

[2] Wei Chai and Barry Vercoe. Music thumbnailing via structural analysis. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 223–226. ACM Press, 2003.

[3] J. Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II, Proceedings of SPIE*, pages 138–147, 1997.

[4] Jonathan Foote. An overview of audio information retrieval. In *Multimedia System*, pages 2–10. Spriner-Verlag, 1999.

[5] Jonathan Foote. Arthur: Retrieving orchestral music by long-term structure. In *3rd International Conference on Music Information Retrieval (ISMIR)*, 2002.

[6] Jonathan Foote, Matthew Cooper, and Unjung Nam. Audio retrieval by rhythmic similarity. In *1st International Conference on Music Information Retrieval (ISMIR)*, 2000.

[7] Jonathan Foote and Shingo Uchihashi. The beat spectrum: A new approach to rhythm analysis. In *IEEE International Conference on Mulitmedia and Expo 2001*, 2001.

[8] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. Query by humming: Musical information retrieval in an audio database. In *ACM Multimedia 1995*, 1995.

[9] Rumi Hiraga, Reiko Mizaki, and Issei Fujishiro. Performance visualization: a new challenge to music through visualization. In *Proceedings of the tenth ACM international conference on Multimedia*, pages 239–242. ACM Press, 2002.

[10] Keiichiro Hoashi, Kazunori Matsumoto, and Naomi Inoue. Personalization of user profiles for content-based music retrieval based on relevance feedback. In *Proceedings of*

the eleventh ACM international conference on Multimedia, pages 110–119. ACM Press, 2003.

[11] Naoko Kosugi, Yuichi Nishihara, Tetsuo Sakata, Masashi Yamamuro, and Kazuhiko Kushima. A practical query-by-humming system for a large music database. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 333–342. ACM Press, 2000.

[12] Kjell Lemstrm, Veli Mkinen, Anna Pienimki, Mika Turkia, and Esko Ukkonen. The c-brahms project. In *4th International Conference on Music Information Retrieval (ISMIR)*, 2003.

[13] Kjell Lemstrm and Sami Perttu. Semex-an efficient music retrieval prototype. In *1st International Conference on Music Information Retrieval (ISMIR)*, 2000.

[14] Lie Lu and Hong-Jiang Zhang. Automated extraction of music snippets. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 140–147. ACM Press, 2003.

[15] Elias Pampalk, Andreas Rauber, and Dieter Merkl. Content-based organization and visualization of music archives. In *Proceedings of the tenth ACM international conference on Multimedia*, pages 570–579. ACM Press, 2002.

[16] J. G. Roederer. *Introduction to the Physics and Psychophysics of Music*. Springer, New York, 1979.

[17] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. In *IEEE Transactions on Speech and Audio Processing, Vol.10*, 2002.

[18] Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton. Content-based classification, search, and retrieval of audio. *IEEE MultiMedia*, 3(3):27–36, 1996.