QueST: Querying Music Databases by Acoustic and Textual Features

Bin Cui[1*] Ling Liu[2] Calton Pu [2] Jialie Shen [3] Kian-Lee Tan[4]

[1] Department of Computer Science & National Lab on Machine Perception Peking University, China
bin.cui@pku.edu.cn

[2] College of Computing Georgia Institute of Technology
{lingliu, calton}@cc.gatech.edu

[3] Singapore Management University
jlshen@smu.edu.sg

[4] National University of Singapore
tankl@comp.nus.edu.sg

ABSTRACT

With continued growth of music content available on the Internet, music information retrieval has attracted increasing attention. An important challenge for music searching is its ability to support both keyword and content based queries efficiently and with high precision. In this paper, we present a music query system – $QueST$ (Query by acou$S$tic and Textual features) to support both keyword and content based retrieval in large music databases. $QueST$ has two distinct features. First, it provides new index schemes that can efficiently handle various queries within a uniform architecture. Concretely, we propose a hybrid structure consisting of Inverted file and Signature file to support keyword search. For content based query, we introduce the notion of similarity to capture various music semantics like melody and genre. We extract acoustic features from a music object, and map it to multiple high-dimension spaces with respect to the similarity notion using PCA and RBF neural network. Second, we design a result fusion scheme, called the Quick Threshold Algorithm, to speed up the processing of complex queries involving both textual and multiple acoustic features. Our experimental results show that $QueST$ offers higher accuracy and efficiency compared to existing algorithms.

Categories and Subject Descriptors

H.3 [**Information Systems**]: Information Search and Retrieval; J.5 [**Arts and Humanities**]: Music

General Terms

Algorithms, Experimentation, Performance

Keywords

Music,Search, Textual feature, Acoustic feature, Similarity notion

*This work is supported by the NSFC under grant No. 60603045.

MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany.

1. INTRODUCTION

The availability of music in digital form continues to grow at an astonishing speed. There are also growing efforts in various organizations to build digital libraries with large collections of music for education, entertainment and research purposes. The rapid growth and continued availability of music data has created a number of new challenges for managing, exploring and retrieving music data with rich query expressions, high accuracy and efficiency. Music databases typically have significantly different and in many ways more complex data structures compared to traditional library systems. Music data can be organized and queried by different textual and acoustic features. For example, one way to organize and search music data is to use auxiliary text information, such as the song title, the artist's name, etc. In addition, music data can also be represented by the semantic information of music objects, i.e., acoustic content, such as timbre, rhythm and pitch.

A music database stores all the music collections and provides a rich set of querying capability to allow users with levels of musical and information retrieval expertise to retrieve music objects of interest in various formats and by various mechanisms. For example, some users may conduct text-based search by specifying meaningful keywords such as the singer's name and the song title. Other users may search the music database by its semantic content. For example, a musicologist may use a few bars of a music score to find similar music pieces in the database or to determine whether a composition is original; a layperson may just hum a tune and let the system identify the songs with similar melody.

According to the music information retrieval literature, there are two widely accepted and yet independent approaches to accessing music objects in music databases:

- **Query-by-text**: Many of the existing music search engines use text descriptions to label music objects. In these applications, short annotations or text labels are used for interpreting or indexing the music objects in the databases. The system basically retrieves music objects whose text labels match the query text. The effectiveness of this approach depends on the success of text searching capabilities, as well as the accuracy and precision of text labelling. However, manually text-labelling large music collections is an expensive and time-consuming process.
- **Query-by-content**: These kinds of queries are essential in a situation where users cannot describe the queries of interest by keywords but rather would like to query the music databases by, for example, playing a piece of music or humming. The query-by-content approaches to date suffer from

at least two main problems. First, users might not always have a sample piece of music at hand to issue a query. Second, a user may prefer to query the music database via certain high level semantic concepts, such as find all music objects with similar genre or instrument configuration as the query music object.

To support both keyword and content based retrieval in large music databases, we present a music query system – $QueST$. All incoming music objects first pass through a musical feature extraction module. Various sets of features are extracted through analysis of the music content, such as text descriptions and acoustic content. Second, an independent index for each musical representation is constructed, i.e. a hybrid index structure for keyword-based music queries, and multiple high-dimensional indexes for content based queries. Third, upon receiving a user query, the system first identifies the query type based on the query conditions specified, and decides which index should be used for query processing.

In this work, we focus on providing a general solution for querying large music databases by both textual and acoustic features, and contribute to its advancements in the following ways:

- We design an index for keyword queries that uses only textual features. We treat each text descriptor as an index term, and build a hybrid structure that integrates an Inverted file [6, 16] and a Signature file [5] to index the text features to support single and multiple keywords search.
- For content based query, we introduce the notion of similarity in terms of acoustic features of music objects to precisely capture the music semantics. Thus the system can answer the query effectively according to the user preferences, like melody, genre, etc. The Radial Basis Function (RBF) neural network is used to generate similarity oriented musical features for different notions based on the original semantic features. To improve the computational efficiency, we adopt PCA to reduce the acoustic feature dimensionality before index construction. Furthermore, a learning algorithm has been proposed to train the system.
- A novel result fusion scheme, called Quick Threshold Algorithm (QTA), is designed for efficient processing of complex hybrid queries involving both textual and acoustic features.

The remainder of this paper is organized as follows. In section 2, we review some background and related work. Section 3 presents the detailed algorithms of the proposed scheme. Section 4 describes a performance study and gives a detailed analysis of results. Finally, we draw some conclusions for this work.

2. PRELIMINARIES

In this section, we first describe different query types in a general music query system, followed by the acoustic representation of music data, and the basic indexing algorithms used as the basis for the $QueST$ indexing schemes.

2.1 Query Type Definition

In a general music query system, we need to support querying by acoustic and textual features through multiple types of queries. Query classification has been widely investigated in the community of information retrieval and query answering [11, 24]. Various schemes have been proposed to categorize free/regular-form factual queries. However, such approaches cannot achieve high accuracy for query categorization, which makes them impractical for real-life applications. Therefore, in $QueST$, a checkbox style interface is constructed to allow user to select query type and input different keywords and/or music sample for further processing. Generally, music queries are typically classified into three categories:

- **Keyword query ($K - query$):** A K-query finds a music item using certain keywords, e.g., "Find music items of Beatles" and "Find music clip performed by Beatles and production year is 1966.".
- **Content query with certain similarity ($C - query$):** In a C-query, a query sample is used to locate music items using a certain type of similarity notion, e.g., "Find music items similar to the humming input (similar melody)" or "Find music items with similar genre to the query example".
- **Hybrid query ($H - query$)**: In this paradigm, queries are expressed using both keywords and samples. Music items that are similar to the samples (based on single or multiple similarity notions) and labelled with the same keywords are returned. For example, "Find music items that has a similar melody to the query sample and production year is 2000.". In some cases, the user only provides a sample to find the music under multiple similarity notions. We treat this as a hybrid query also.

For the content based query, we define multiple similarity notions, e.g. melody, genre and instrument. For example, musical genres [21] are categorical labels created by humans to characterize pieces of music. A musical genre is characterized by the common characteristics shared by its members. These characteristics typically are related to the instrumentation, rhythmic structure, and harmonic content of the music. The humming query is one kind of content based query, which requires the system to return the music items that have a similar melody. We note that the query classes we have introduced above may not be applicable for arbitrary music retrieval systems. As an example, music archives may not have many humming queries in some cases, but they may instead have more keyword queries. And the systems might have different and even more query classes. But the idea of query type and similarity notion is generally feasible, and our system can be easily extended to support more query classes and similarity notions.

2.2 Music Representation

The most classic descriptor for music representation is the textual features, such as the title, artist, album, composer, genre, etc. In addition to the metadata of music files, we can also extract the content feature information from the music files. Content-based music retrieval systems have attracted much research interest recently [12, 14, 19, 20, 21, 18].

In music retrieval, melody is the key content of music information, and is also a very important cue. Generally, the melodies of music pieces are stored in the database in the form of music score or music notes. A user can produce a query by keying in a sequence of music notes, playing a few notes on an instrument, or singing/humming through a microphone. The query melody is usually incomplete, inexact and may correspond to anywhere in the targeting melody. The query tune can be converted into a note-like representation using signal processing techniques, and hence string matching techniques can be applied to music retrieval. A number of approaches have been proposed for content-based symbolic music retrieval. Ghias et al [7] reported effective retrieval using query melodies that have been quantized to three levels, depending on whether each note has a higher, lower, or similar pitch as the previous one. Besides simplifying the pitch extraction, this allowed for less-than-expert singing ability on the part of the user. In [2], music melody was represented by four types of segments according to the

shape of the melody contour, the associated segment duration and segment pitch. Song retrieval was conducted by matching segments of melody contour. Generally, these approaches require precise detection of individual notes (onset and offset) out of the query. However, it is not uncommon for people to provide query with wrong or incomplete notes. The string matching result would suffer drastically when the error in note detection is not marginal. To deal with the such issues, the query processing can be done based on beats instead of notes. The statistical feature, such as tone distribution, is robust against erroneous query. However, this requires users to hum by following a metronome [9]. Such requirement could be difficult for users sometimes. When a tune is hummed from memory, the user may not be able to keep up with the correct tempo. Different notes (e.g. duple, triple, quadruple meters) of the music can also contribute to the difficulty. Additionally, most existing algorithms [15, 22, 27, 17] are computationally expensive (and hence impractical) for large music databases as the entirety of a database has to be scanned to find matching sequences for each query.

On the other hand, accurate pitch contour or melody information extraction from music files and input queries is a non-trivial task. In particular, it is extremely hard to extract melody from polyphonic music due to multiple notes occurring simultaneously in polyphonic music. Some existing works convert music data into indexable items, typically points in a high-dimensional space that represent acoustic features, such as timbral texture, rhythmic content and pitch-based signal content [21, 25, 13, 18]. This kind of approaches have two advantages: first, it can capture more semantic information besides the melody; second, the features can be easily indexed using high-dimensional structures, thus we can support query in large music databases efficiently. Therefore, we focus on acoustic retrieval in this paper. Based on [21, 13], we extract multiple intrinsic acoustic features and these are briefly discussed below:

- **Timbral textural features:** Timbral texture is a global statistical music property used to differentiate a mixture of sounds. Different components for this feature are calculated using the Short Time Fourier Transform, including spectral centroid, spectral flux, time domain zero crossings, low energy, spectral roll-off and Mel-frequency cesptral coefficients(MFCCs). The 33-dimensional feature vector produced contains: means and variance of spectral centroid, spectral flux, time domain zero crossings and 13 MFCC coefficients (32) plus low energy(1).
- **Rhythmic content features:** Rhythmic content indicates reiteration of musical signal over time. It can be represented as beat strength and temporal pattern. We use the beat histogram (BH) proposed by Tzanetakis et al. [21] to represent rhythmic content features. The main idea behind the calculation of BH is to collect statistics about the amplitude envelope periodicities of multiple frequency bands. The specific method for their calculation is based on a Discrete Wavelet Transform (DWT) and analysis of periodicity for the amplitude envelope in different octave frequency bands. In this study, the 18-dimensional feature vector is used to represent rhythmic information of music signal. It contains: relative amplitude of the first six histogram peaks (divided by the sum of amplitudes), ratio of the amplitude of five histogram peaks (from second to sixth) divided by the amplitude of the first one, period of the first six histogram peaks, and the overall sum of the histogram.
- **Pitch-based signal features:** Pitch is used to characterize melody and harmony information in music and can be extracted via the multi-pitch detection techniques. We use algorithm proposed by Tolonen et al. [**?**] to extract pitch based signal features. In this approach, the signal is first divided into two frequency bands. After that, amplitude envelopes are extracted for each frequency and summed to construct a pitch histogram. A 18-dimensional pitch feature vector incorporates: the amplitude and periods of the maximum six peaks in the histogram, pitch interval between the six most prominent peaks, and the overall sums of the histograms.
- **DWCHs:** Daubechies Wavelet Coefficient Histograms represent both local and global information by computing histograms on Daubechies wavelet coefficients at different frequency subbands. The distinguishing characteristics are contained in the amplitude variation, and consequently, identifying the amplitude variation would be essential for music retrieval. The DWCHs feature set contains four features for each of the seven frequency subbands along with nineteen traditional timbral features.

2.3 Basic Index Structures

A practical and useful music retrieval system should allow users to query by textual or/and acoustic features. To efficiently support such queries, we need various structures to index different musical features, e.g. text descriptions and acoustic features. Here we introduce some techniques for feature transformation and indexing.

The musical features we extracted from the music database are basically high-dimensional acoustic features and textual features. Many emerging database applications such as image, time series and scientific databases, manipulate high-dimensional data. There is a long stream of research on solving the similarity search problem in high-dimensional space, and many indexes have been proposed [1], such as M-tree [3], VA-file [23] and iDistance [26]. These structures are usually designed for generic metric space, where object proximity is defined by a distance function, such as Euclidean distance. However, these methods cannot support retrieval of non-numerical data efficiently, such as text strings. There are two principal indexing methods for text databases: Inverted file [6, 16, 28] and Signature file [5]. An Inverted file builds an index entry for each term that appears anywhere in the database, and this index entry is represented by a list of the documents containing that term. To process a query, a vocabulary is used to map each query term to the address of its inverted list; the inverted lists are read from disk and are merged to return the answers. In Signature file indexes, each record is allocated a fixed-width signature, or bitstring of w bits. Each word that appears in the record is hashed a number of times to determine the bits in the signature that should be set. The queries are similarly hashed, then evaluated by comparing the query signature to each record signature. Each candidate record must be fetched and checked directly against the query to determine whether it is a false match or a true match.

3. MUSIC QUERY PROCESSING

In this section, we describe the $QueST$ music query system architecture, the related indexes and the query algorithms.

3.1 System Architecture

From our experience with music data repositories and music information retrieval, we observe two interesting phenomena. First, a music query is typically based on a subset of features, which could be either textual or content features. Second, the feature subsets used by a single query are typically not intermixed. Concretely, we mean that the subset of textual features used in a query is usually independent of the subset of content features used in the same query, and each feature typically has different characteristics, although we

need to integrate the results for complex queries that involve both textual and content features. Therefore, the $QueST$ Music Query System is designed with two objectives in mind. First, we want to provide efficient processing of complex queries to a music database system by indexing acoustic and textual features of music objects. Second, our music indexing techniques should not only speed up the processing efficiency of complex queries but also offer high accuracy compared to those offered by existing music retrieval systems. Figure 1 shows the overall structure of the $QueST$ system.

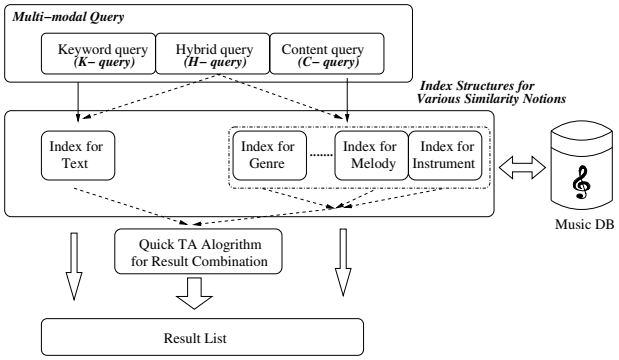


Figure 1: Overview of $QueST$ Music Query System

The system consists of the following components for keyword or content-based queries without resorting to any user feedback and manual query expansion. First, all music objects will pass through the musical feature extraction module before entering into the music databases. Various sets of features are extracted through analysis of the music contents, such as text descriptions and acoustic content. The text descriptions for a music object includes the title, artist, album, genre, year, etc. Inspired by work on music analysis [21, 13], we use multiple musical content features which can capture the acoustic characteristics of raw signal effectively. These features represent the timbral texture, rhythmic content, pitch-based signal feature and Wavelet Coefficient Histograms.

Second, we construct independent index for each musical representation. We propose a hybrid keyword index structure, which integrates the Inverted file [6, 16] and Signature file [5]. For content features, we introduce a concept of similarity notion to define more precisely the similarity between music objects, such as melody and genre. We apply a RBF neural network [8] to generate similarity oriented musical features. To further improve the computational efficiency, we use PCA [10] to reduce the acoustic feature dimensionality while retaining majority of the music information. Thus each similarity notion has its own separate high-dimensional index structure [26, 1].

Third, upon receiving a user query, the system first identifies whether it is a keyword based query or an acoustic content query or a hybrid query based on the query condition specified. Then the query is passed to the feature extraction module. We determine the set of features we can obtain from the query input and decide which index should be used for processing the query. For content based query with music sample input, we extract the acoustic content features and apply RBF neural network to tune the feature weights. If the query type is hybrid (comprising both textual and content features), we get the query answers via various indexes, and fuse the answers with a specially designed result fusion scheme QTA.

3.2 Index for Keyword Query

In keyword based music search, users generally search the database using different kinds of textual features, such as artist name, album name and genre. The typical query could be "find all $Jazz$ music performed by $Michael\ Jackson$". Therefore, the input query could be matched against multiple text attributes in the file, which could be title, artist or other information. Although classical indexing schemes, such as the Inverted file [6, 16, 28] and Signature file [5], can be applied to handle multiple attributes, they are either inefficient or ineffective (in terms of query false matches). In the Inverted file approach, each keyword has an associated list which contains the addresses to records having that keyword. Solving a partial match query in such a system involves scanning the lists associated with multiple keys and then determining which records appear in all of the lists. The technique works well in situations where the number of keys specified in the query is small, and where the query retrieves relatively small number of records. However, it can be expensive for multi-key query or for queries which are not very selective. When the query has several key values we have to perform several list-intersection operations. For example, let the inverted lists for the two keywords be "$Michael\ Jackson$" $= \{5, 8, 12, 28, 40, 55, 80\}$ and "1990" $= \{1, 8, 16, 23, 28, 36, 62, 70, 90, ...\}$. If we want to find the songs of $Michael\ Jackson$ in 1990, clearly the answers are their conjunction, i.e. music 8 and 28.

On the other hand, in the Signature file approach, the values of all the key fields are "squeezed" into a small bit-pattern which is stored in a separate file containing one signature for each record in the data file. The signature file includes one associated descriptor - a bit-string formed by superimposing (bitwise OR-ing) the codewords for all keywords of each data item. A codeword is a bit-string which is derived from the hashing value of a keyword. For a keyword k_i, we denote its corresponding codeword by $cw(k_i)$, where $cw()$ is a hash function. For an object m with 10 keywords, the superimposed descriptor $super(m)$ is given by: $super(m) = cw(k_1)\ OR\ cw(k_2)\ OR\ .....\ OR\ cw(k_{10})$.

The example shown in Figure 2 illustrates the procedure to OR the codewords (hashing value) for keywords to form the signature for a music object.

Keyword Type	Value	Codeword	Keyword Type	Value	Codeword
Title	Freedom	0001000101	Lyrics Writer	Lee Dik	1101000101
Artist	Jack	1001000101	Instrument	Drum	0111000101
Album	Deep Blue	0001000001	Publisher	BMG	0001110101
Gerne	Jazz	0001011101	Length	4 min	0101000101
Composer	Peter	0001111101	Year	2003	1001110101

Freedom	0001000101
Jack	1001000101
Deep Blue	0001000001
Jazz	0001011101
Peter	1101000101
Lee Dik	0001110101
Drum	0101000101
BMG	1001110101
4 min	0101000101
2003	1001110101
OR	
Signature	1101111101

Figure 2: An Example of Signature Generation

The descriptor for a partial or full match query Q can also be formed by OR-ing the codewords for all the $specified$ keywords, $super(Q) = OR\ cw(k_i),\ where\ k_i \in Q$.

If the keywords specified in a query Q is denoted by $k_1, k_2, \ldots, k_n$, we can define set $Spec(Q)$ of specified keyword numbers for a query Q as $Spec(Q) = \{i | k_i \in Q\}$.

It should be clear from the above two definitions that the bits in the query descriptor for a query Q will be a subset of the bits in the keyword descriptor of any matching object M. Thus, in order to determine which records satisfy a query, we scan the whole superimposed descriptor file, testing each descriptor against the query descriptor to determine whether the subset property holds or not. To answer a query with this approach, the files of signatures are first searched to select candidate records. The set of candidate records comprises all the records which satisfy the query but may also contain records which match at the signature level but do not satisfy

the query (false matches). False matches is the main weakness for the Signature file.

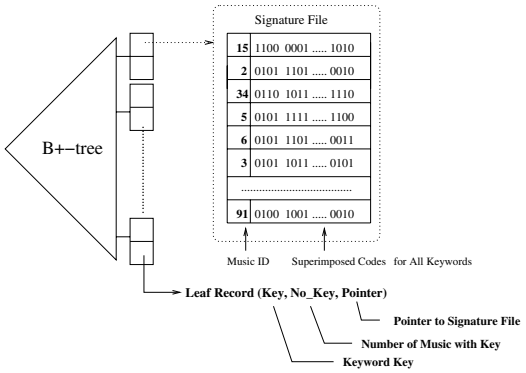


Figure 3: The Index Structure for Keyword Search

To remedy the weaknesses for both structures, we design a hybrid indexing scheme to provide speedy and accurate keyword based search. Figure 3 illustrates this hybrid indexing structure. In the first layer, a B$^+$-tree is built to index keys which are text strings. Note that the text descriptions are used to label music object in the music databases. Basically, the music can have multiple and various music descriptions. For ease of presentation, we assume the text information of the music is shown as follows:

$Music_{ID} : < title, artist, album, genre, composer, lyric\ writer, publisher, instrument, length, year >$

We note that there are other text descriptions that we could have used, such as CD track number, rating, file format, etc. Our representation can be easily extended to include these and other information and such extensions will not affect the usability of the system. We convert the $year$ and $length$ to text as well. Each leaf node of the B$^+$-tree contains $< key >$ and one pointer referring to an inverted list which is a contiguous list of the music containing the text key. Since this list could be very long, we use a signature file to organize these music objects using bit-strings for all keywords of each music item. The advantage of our scheme is that we only need to search the B$^+$-tree once compared with the Inverted file. Additionally, via the B$^+$-tree, we can prune away majority of the irrelevant music objects, and the signature file in the leaf level is sufficiently efficient to return the answers.

Input: Hybrid indexing scheme, Query keyword list **Output:** Answer list	
1	Pick the keyword k_i with largest cardinality
2	Search the B$^+$-tree to find signature file referred by k_i
3	Construct superimposed codeword cw with query keyword list
4	Return music ID whose descriptor satisfies subset property

Figure 4: Algorithm for Keyword Search

With the above scheme, the basic search algorithm, shown in Figure 4, is quite straightforward. From the keyword input, the user can pick up keyword k_i which is in the field with the largest cardinality in line 1. In an Inverted file index, each distinct text feature in the database is held in a textual key list and each key refers to the music list containing the certain keyword. Clearly the cardinality of feature field $artist$ is much larger than $genre$ which is typically only 30, while there could be thousands of artists in the music database. For a query with $artist$ and $genre$ as input, we first select $artist$ to find a shorter candidate list, and hence has better pruning efficiency. In Line 2, we find the appropriate signature file sf with the B$^+$-tree in the first layer. The query descriptor is constructed with $super()$ and then used to determine the final answer list.

3.3 Index for Content Query

In this section, we discuss how to support content based query by music sample. In other words, the query input can be a song, a piece of music, or just a music piece hummed by a user, and the user wants to find similar music from the database. After analyzing the content of the input music object, we can extract the intrinsic semantic features of the music, such as rhythmic and pitch-based signal content. We can search the music database with those acoustic characteristics of the given a query sample. By extracting the acoustic features of the query and data, and mapping them to a high-dimensional space, we can find similar music using a single high-dimensional index structure.

However, a user may want to find the music under different similarity such as same genre or similar melody. Using the original acoustic feature space, the music system only returns the same answer set for various similarity demands, and such answers may not be optimal in many cases. Therefore, we introduce a novel scheme to map the original acoustic features to different spaces according to the respective similarity requirement.

3.3.1 Similarity Notion Oriented Signature Generation

Our similarity notion oriented signature generation is based on Radial Basis Function (RBF) neural network processing. RBF neural networks derive their strengths from a "model-free" processing of data and a high degree of freedom associated with their architecture [8]. It has been widely applied for function approximation, pattern classification, data compression and so on. The goal of the method is to obtain different NN parameters respectively using different training sets for various similarity notions, and thus we can map the raw musical signal into different acoustic feature spaces nonlinearly.

In our framework, an RBF classifier is used as the mechanism to conduct complex nonlinear feature mapping from the original high dimensional feature space to a new feature space. This has several advantages: (1) Adaptiveness - it can represent different similarity notions via learning processes with various training sets; (2) Compactness - it can generates featuers with lower dimensionality, as we can set fewer number of neurons in the hidden layer, which is applied for indexing, and (3) Efficiency - In contrast to multilayer perceptron neural network, linear least square based method is used to determine the weights between the hidden and output layers. This can make the network faster and free of local minimal. In our implementation, the RBF neural network contains three layers, including the input layer, the hidden layer and the output layer. The output layer contains class labels and is used for training. The neuron values in the hidden layer are extracted as music descriptor for content based query. The number of neurons in the hidden layer is configured to be smaller than the one in the input layer which is equal to the dimensionality of the acoustic features.

In the RBF training stage, training samples are selected with random sampling to cover each subclass for particular similarity notion. Based on the current setting, we use 20% of the whole data size as learning examples which sufficiently cover all subcategories of data. After the network training is completed, feature extraction can be achieved by feeding the musical feature vectors into the net-

work and taking the vectors computed in the hidden units as the lower dimension representations. These lower dimension vectors can be used for effective similarity-based search with certain similarity notion, e.g. search by genre or melody similarity to an input music.

3.3.2 Preprocessing with PCA

The RBF neural network can effectively convert the original acoustic features into lower dimensional subspaces with respect to different similarity notions. However, the high dimensionality of acoustic features may degrade the efficiency of neural network. First, it incurs high training cost as the computational cost is proportional to the feature dimensionality. Second, it may reduce the accuracy of the training effect, as the points in the high-dimensional space are sparse and hardly distinguishable. Third, it requires larger topological structure for neural network.

To alleviate the above problems, we try to reduce the dimensionality of the acoustic features before they are fed into the RBF neural network. In our work, we adopt Principal Component Analysis (PCA) [10] as a "pre-processing" step to optimally reduce the dimensionality of the inputs to the neural network. In this way, we can speed up the training and processing time. PCA is a widely used method to reduce the dimensionality of high-dimensional data. Using PCA, most of the information in the original space is condensed into fewer dimensions along which the variances of data distribution are the largest.

To generate the new features, we first reduce the dimensionality of the input raw feature vector with PCA, which is applied to all music in the database rather than only to the training samples. This has an advantage in that the covariance matrix for each type of single feature vectors contains the global variance of music in the database. The number of principal components to be used is determined by the cut-off value ψ. There is no formal method to define this cut-off value. In this study, the cut-off value ψ is set to 99 so the minimum variance that is retained after the PCA dimensions reduction is at least 99%.

3.3.3 Indexing the Semantic Oriented Feature Spaces

Once the similarity notion oriented feature sets have been extracted, we can simply adopt a high-dimensional index structure for each similarity notion, as music objects are feature-transformed into points of a vector space with a fixed dimension d. There is a stream of research on solving the similarity problem in high-dimensional space, and many indexes have been proposed for this purpose [1, 3, 23, 26]. In this study, the iDistance is employed due to its simplicity and superior performance as reported in [26]. However, any other high-dimensional index structure can be adopted, and it will not affect the efficiency of our scheme.

3.4 Hybrid Query

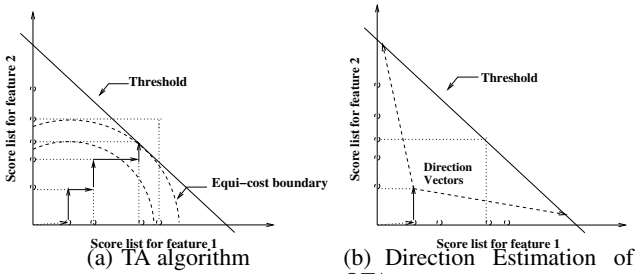
As introduced previously, the user may provide a music sample and some keywords as a query. In some cases, the user may give one sample but require multiple similarity notions, e.g. similar genre and instrument. For simplicity, we also treat this as a hybrid query. We do not provide specific index structure for such queries, but refer to the keyword and acoustic feature indexes separately and then merge the answers.

To achieve efficient combination of scores from different result lists from indexes representing various similarity notions, a new cost-aware algorithm, called QTA (Quick Threshold Algorithm) is introduced. Note that since all objects in the result list from keyword index has the same score, the QTA is only used to merge score lists from different index trees constructed with musical content features. The QTA is based on the well known TA (Threshold Algorithm) algorithm [4]. To facilitate the understanding of QTA, Figure 5 gives the pseudo code of the TA algorithm, which returns the top-k objects when it runs k times. The main difference between our QTA algorithm and the original TA lies in the scheme to select the next partial score list to facilitate sorted access. Rather than picking the score list in a round-robin style, QTA selects the next list to evaluate by optimizing access costs with the same termination condition.

Input: d result lists for d different features
Output: Object for top K query

Initiation: rset $\Leftarrow \emptyset$, $tr \Leftarrow 0$
1 $next\_list \Leftarrow opt\_List()$
2 $obj_{next} \Leftarrow$ get object in $next\_list$ with sorted access
3 **If** obj_{next} has not been accessed
4 calculate aggregated score of obj_{next}
5 insert obj_{next} into rset
6 **If** rset $\neq \emptyset$
7 $obj_{max} \Leftarrow$ object with max. aggregated score
8 **If** $tr <$ obj_{max}'s aggregated score
9 return obj_{max} and delete it from rset
10 **If** all lists received at least one sorted access
11 $tr \Leftarrow$ add up all partial scores from each list
12 with sorted access

Figure 5: Threshold Algorithm (TA) for Combining Score Lists



(a) TA algorithm (b) Direction Estimation of QTA

Figure 6: Execution Path for TA and QTA

Before we introduce the detailed procedure for QTA, we show how the TA algorithm performs in Figure 6(a). We can see that associated query execution path is like a staircase (feature score list is selected in a round-robin style) and KNN objects will be returned when the threshold boundary is hit by the "walk". Note that for TA, the highest partial score accessed most recently in feature list i is the score value in dimension i of Figure 8(a). A sorted access to score list i leads to an increase in the partial score. In general, this process can be seen as a position change in a multidimensional space and will be terminated when the aggregated score of K objects (K is predefined) is larger than or equal to certain threshold boundary. Note that, in TA, it is equal to the summation of partial score for all the lists and corresponds to a diagonal line graphically in 2 D plate. The whole procedure is inefficient in terms of sorted access cost. To achieve this, the upper bounding cost to estimate the access cost is first introduced below:

$$UC(l_1, ..., l_k) = \sum_{i=1}^{k} (AC_{l_i}(C_{l_i}(l_1, ..., l_{l_k})) + \sum_{p=1, p \neq i_i}^{d} RC_p) \quad (1)$$

where $l_1, ..., l_k$ and l_i represent the execution path and the score list with sorted access in the i-th iteration. $AC_{l_k}(w)$ denotes the sorted access cost to obtain the w-th biggest partial score from list k and RC_p.

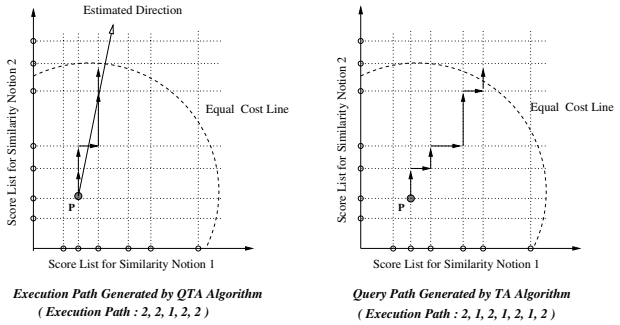


Figure 7: Example of Query Path

The basic goal for the QTA algorithm is to reach the threshold boundary with minimal access and computational cost. To achieve this target, the first step is to estimate the threshold boundary and then determine in which direction to forward to reach the boundary with the minimum cost. As an example shown in Figure 7, the query step is 3 for QTA which is much fewer that TA whose step is 7. Note that, "$2, 2$" in QTA can be treated as one step.

Input:	Current position p in d dimensional space
Output:	Path direction di to hit threshold boundary with minimum cost
Initiation: $d\_set \Leftarrow \emptyset$	
1	tr_e = max. aggregated score in current rset
2	d_set = candidate path direction generation
3	**For each** direction di in d_set
4	Estimate the position in threshold boundary with threshold tr_e
5	Estimate cost with equation 6 to hit the target position
6	Return the di with minimal cost

Figure 8: Execution Direction Estimation

The algorithm for direction estimation is shown in Figure 8. In this study, we use the maximum aggregated score of all objects in $rset$ as an estimated value of tr, denoted by tr_e(line 1). After that, M uniformly distributed vectors are generated. A graphic view is shown in Figure 6(b). In order to make the boundary position estimation, we simply use binary search to discover the point along the position, where the aggregated score becomes tr_e(line 4). Once the position in the threshold boundary is fixed, the upper cost for reaching this position needs to be estimated using equation 1. The estimator is polynomial extrapolation for AC and RC. The direction with the lowest upper bounding cost will be picked at the end. Once the direction is determined, the optimal sort-access sequence can be easily generated - the basic idea is to select one path combination that is as close to the path as possible.

3.5 Conducting Music Query

We have discussed each component of the proposed $QueST$ system. Now we are ready to describe the overall procedure for music query processing. Figure 9 shows the algorithm of a general search. We first analyse the query input, and determine the type of the query the user wants. In lines 3-4, we determine the keyword with highest preference and conduct the search on our hybrid Inverted-and-Signature file structure. If the query is a content query, we first extract acoustic features from the music sample, then we use PCA to reduce the dimensionality, and we use RBF neural network to tune the weights for a certain similarity notion. After the music sample is mapped to a point in the high-dimensional space, we use the traditional similarity search to get the similar music objects in the database (lines 6-8). For the hybrid query as shown in lines 10-12, we conduct keyword search and content search separately, and merge the answers using the proposed QTA before they are returned to the user.

Input:	System architecture, query input
Output:	Answer list
1	Process query input
2	**If** Query is keyword query
3	Determine the keyword
4	Search via text index
5	**Elseif** Query is the content query
6	Extract acoustic features
7	Convert to a certain similarity space via RBF NN
8	Access the index w.r.t. the similarity notion
9	**Else**
10	Get answer from keyword index
11	Extract and transform acoustic features, access multiple indexes
12	Merge the answer by QTA

Figure 9: Search Algorithm of Music System

4. PERFORMANCE EVALUATION

In this section, we present an experimental study to evaluate the proposed music system $QueST$. Given a music query, either keyword or content or hybrid query, we find the matching music objects in the database using the proposed indexing methods. We have implemented our music retrieval system $QueST$, and conducted experiments to evaluate the effectiveness of $QueST$ by comparing it with some traditional approaches. As performance metrics, we use response time, and precision which is the ratio of the number of relevant records retrieved to the total number of records retrieved. The relevance of music is determined by human for a certain query, because the indexes try to find the most similar objects in terms of Euclidean distance. On the other hand, recall and precision are typical performance metrics for information retrieval system. However, manually identifying all the similar music objects for each music in a large music database is very time consuming, and almost infeasible in reality. As a result, we only show the performance on precision, and with the various numbers of similar music objects returned; our results are enough to compare the effectiveness of different schemes. For each experiment, we run 100 different queries, and report their average performance for system evaluation. The experiments have been conducted on a computer with P4 CPU (2.5GHz), 1GB RAM, and running Microsoft Windows XP Professional Operating System.

The real dataset used in the experimental study is constructed from compact disks and MP3 compressed audio files. The major parameters are shown in Table 1. For example, the genre can have thirty different types including Classical, Country, R&B, Hip-hop, Jazz, Rock, Metal, Blue, Dance, Caribinal and Pop etc.

Dataset Parameters	Details
Number of music	3000
Number of Artists	400
Number of Albums	500
Number of Genres	30
Publishers	20
Years	1970 - 2003

Table 1: Some Parameters of the Dataset

4.1 Performance on Keyword Query

In the first set of experiments, we study the performance of $QueST$ on keyword search. Note that, only keywords are used in this experiment. We first use the real dataset and a mixture of operations with different numbers of keywords to evaluate the efficiency of the keyword search structure in $QueST$. In this study, we define four types of queries with respect to the numbers of keyword input, i.e. Type I to IV. For example, a type III query can be "find $Michael\ Jackson$'s songs in 1990 with genre $Rock$" (three keywords).

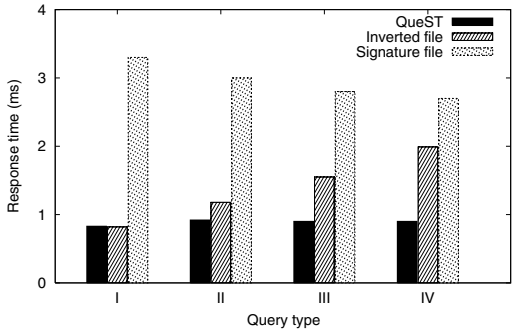


Figure 10: Performance on Keyword Search

We compare $QueST$ against Inverted file and Signature file approaches under various query types. For the Signature file, we set 128 bits for each music object. Figure 10 shows the response time for different keyword searches. The Signature file is significantly worse than the other two methods. This is because it has to scan the whole signature file to get the candidates and then filter out the false matches. Moreover, the processing cost is also higher. Unlike the other two methods, the performance of Signature file is improved when the query has more keywords, although it still performs worse than the two competitors. This is because the Signature file contains more information about keywords. Also we use "OR" function to generate the signature and it introduces more false matches when fewer keywords are provided. For the Type I query, the Inverted file is a bit faster than $QueST$. This is because basically both approaches adopt the same structure in the first tier, i.e. B^+-tree for text string. However, $QueST$ stores more information in the leaf level, e.g. the signatures for the keywords. In such a case, $QueST$ may introduce more disk I/O. However, when the query has more keywords input, $QueST$ clearly shows better performance, about 100% better than Inverted file for Type IV query. In the Inverted file, the B^+-tree has to be accessed multiple times to get the individual list for each keyword, and then the lists have to be merged to get the final answers. Therefore, the more keywords there are, the more steps are needed to complete the result set generation and thus the longer the response time. However, the performance of our $QueST$ structure is not affected by the query type significantly. The main reason is that the performance of keyword based query plays an important role to determine the final performance. We always select the keyword with the highest cardinality, which results in the shortest answer list. Additionally, we use the signature file in the leaf level to prune the answers with respect to the rest of the keywords, and the processing of the signature file is very efficient. Thus $QueST$ only needs to traverse the B^+-tree in all the cases, and hence resulting in faster response.

4.2 Performance on Content Query

In this section, we present the results of experiments to verify the effectiveness of $QueST$ for content based music retrieval. Music retrieval can be informally defined as: the user submits a query music clip and the system retrieves a list of music objects that are most similar to it; the list of "matching" music is displayed in non-ascending order of the degree of similarity. However, the meaning of music similarity can be defined boardly. To efficiently meet the user demands, we carefully define various similarity notions, which is most commonly used in real life applications. Each notion of similarity corresponds to one kind of content query and music descriptor. As we described previously, we first extract all the acoustic features with 116 dimensions from the dataset and use the trained RBF neural network to map the features to multiple high-dimensional space with respect to a certain similarity notion. In this study, four popular types of similarity are used for testing:

- **Type V:** find music that has similar genre from the database.
- **Type VI:** find music performed by the same artist from the database.
- **Type VII:** find music with the same instrument from the database.
- **Type VIII:** find music that has similar melody from the database.

In this experiment, we randomly pick 100 music objects as query examples, and return up to 50 objects in terms of similarity measurement. We compare our $QueST$ with three methods, i.e. *MARSYAS* which represents 69 dimensional feature sets including timbral, rhythm and pitch features [21], *DWCHs* which represents 47 dimensional Daubechies Wavelet Coefficient Histograms[13], and $InMAF$ [18] which linearly concatenates the reduced dimensionality of *MARSYAS* and *DWCHs* feature sets.

One of our conjectures is that it is possible to obtain effective retrieval from low-dimensional feature space if these vectors are carefully constructed. In our framework, we build indexing vectors from high-dimensional "raw" feature vectors via a hybrid architecture consisting of PCA and neural network, which can effectively reduce the size of the final music descriptors. Furthermore, by incorporating human musical perception, more discriminating information could be squeezed into a smaller size of feature vector which leads to superior performance for similarity search for various similarity notions.

The experimental study confirms our claim. Figure 11 summaries the query effectiveness of *MARSYAS*, *DWCHs*, *InMAF* and our proposed $QueST$ for the four different query types. As shown, *MARSYAS* and *DWCHs* are the worst in terms of precision rate. *DWCHs* achieves marginally better performance than *MARSYAS*, because it captures both the local and global physical characteristics of the music signal. *InMAF* performs better by concatenating *MARSYAS* and *DWCHs* feature sets, as it captures more detailed acoustic features of the music objects. However, the linearly cumulative information of *InMAF* cannot discriminate objects due to the

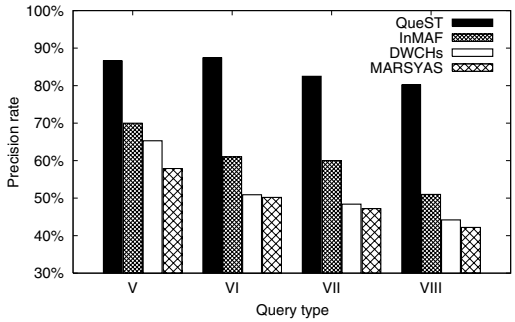


Figure 11: Precision Comparison of Content Query. Dimensionalities of DWCHs and MARSYAS are 47 and 69 respectively.

dimensionality curse. The increase of precision is not proportional to the information included. Although, the raw features adopted in $QueST$ is the same as *InMAF*, the experimental results clearly demonstrate that $QueST$ significantly outperforms the other three methods. For example, Figure 11 shows that, comparing to *InMAF*, $QueST$ improves the retrieval precision from 60% to 85% on average. Overall, around 40% improvement can be observed against the competitors for all kinds of query types. Our proposed $QueST$ has three advantages. First, it adopts both *MARSYAS* and *DWCHs* feature sets, and hence captures the majority of the acoustic features of the music objects. Second, we alleviate the dimensionality curse of the acoustic features. After PCA processing, we can reduce around 25% of the dimensions while retaining 99% of the information in the raw data. In RBF neural network, we set the number of neurons in the hidden layer 20, which is used to index the acoustic features. Note that, we can use any number of neurons in the hidden layer with different neural network configurations. However, we found 20 is almost optimal in our experiments, because it is not very high in terms of dimensionality, furthermore it does not introduce much information loss compared with lower neuron numbers. Third, we use NN to tune the weight of each dimension for different similarity notions. Therefore, the generated multiple sets of features can incorporate human perception more precisely, and yield satisfactory results for all types of similarity.

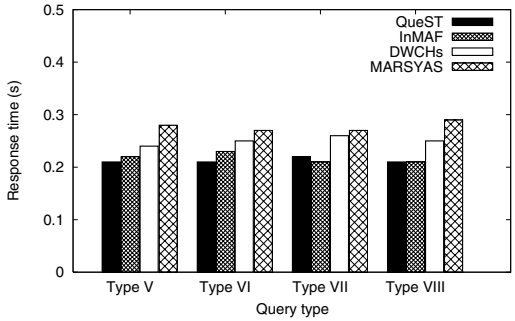


Figure 12: Response Time for Different Query Types.

Now, a large database size and/or a high-dimensional feature vector can make query processing based on any access method very inefficient in terms of searching time. On the contrary, a small but well discriminative feature vector could provide not only superior retrieval accuracy but also fast response time. To further illustrate the performance advantage of using our method, we compute the response time for the various methods including the time to generate feature set and search over a high-dimensional index. For fair comparison, we build the same index, i.e. iDistance [26], for all the methods, however any other high-dimensional index can be used in our system. Results presented in Figure 12 indicate the efficiency of our proposed method based on the evaluation of 4 types of content queries. $QueST$ is the fastest among four competitor. Although $QueST$ adopts all the acoustic features, it reduces the overall dimensions to only 20 via PCA transformation and neural network training, which is originally 116 dimensions. We observe that the improvement is not so significant, because a major portion of the response time is on the feature extraction and processing. Overall, our approach is promising for content query even in terms of response time.

4.3 Performance on Hybrid query

Having evaluated our framework's performance on keyword query and content query, we proceed to examine its performance for hybrid query. In the following study, we proceed to examine its performance for hybrid query. Firstly we present the query definitions based on previously defined similarity notions and possible keywords as follows. Note that, if the query consists of one keyword and one similarity notion, we can simply merge the answers as the answer list of keyword query have the same scores, e.g. all the songs performed by a certain singer. Therefore, we ignore such query in this study as the performance is the same as that for content query.

- **Type IX:** the query consists of one keyword and two similarity notions, e.g. "find music that has similar $melody$ and $singer$ to the given example and the genre is $Jazz$".
- **Type X:** the query consists of one keyword and three similarity notions, e.g. "find music that has similar $melody$, $instrument$ and $genre$ to the given example and produced in year 2000".

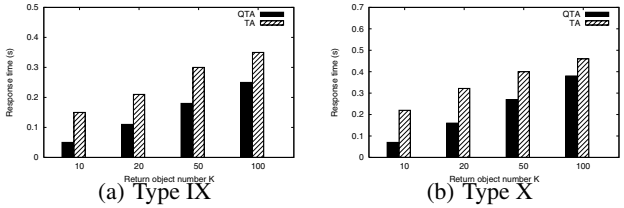


Figure 13: Performance Comparison of Fusion Algorithm

The queries are constructed by randomly selecting different similarity notions and keywords. The number of objects returned, K, is set to be 10, 20, 50 and 100. Figure 13 plots the query response time for both the query types as a function of retrieved object numbers K. In all cases, QTA always gives the lower cost due to significant reduction of sorted access. Compared to TA, QTA can significantly reduce the I/O cost as it estimates the path after an object is returned. When K increases, both algorithms have to access more objects, which introduces more disk accesses and computation. Additionally, the computational costs for query path estimation increase as more objects need to be returned. Furthermore, the type X query needs longer response time compared with type IX query. The reason is that we have to merge the answers from three candidate lists,

which requires more I/O and computational cost. However, the response time for QTA is still substantially lower than that of the TA algorithm. Overall, the QTA can serve as a highly efficient technique for merging result lists.

In the last experiment, we select the QTA algorithm as the fusion scheme in $QueST$, and compare the precision with the other three mechanisms. Note that, for approaches *DWCHs*, *MARSYAS* and *InMAF*, we only need to merge the answer lists from the keyword and content query respectively, as they simply return one answer list regardless of multiple similarity notions. The results are shown in the Figure 14, and the query is a mixture of types IX and X. Clearly, our method shows better performance that the other three competitors, about 40% higher in terms of precision. First, its optimized feature for various similarity notions can reflect human musical perception more precisely. Second, the QTA fusion algorithm can combine the optimal answers from multiple answer lists. Therefore, the proposed $QueST$ is superior for hybrid query against *DWCHs*, *MARSYAS* and *InMAF*.

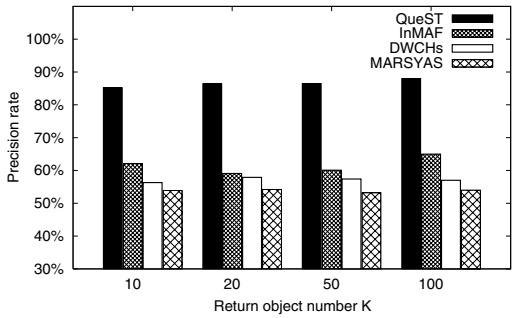


Figure 14: Precision Comparison for Hybrid Query.

5. CONCLUSIONS

We have presented the design of the $QueST$ music query system to support various queries to a large music database by acoustic and textual features. $QueST$ builds multiple index structures for different query types. Concretely, we use a hybrid structure of Inverted file and Signature file for supporting efficient keyword search, and create separate feature indexes for independent subsets of content features of music objects with respect to the similarity notion using PCA and RBF neural network. Our experimental results showed that our proposed schemes can support various queries efficiently in a music database system, while offering a high degree of accuracy, compared to existing music retrieval algorithms.

The work on $QueST$ continues along several directions. We plan to extend the current techniques to support music query on small mobile devices or in a P2P environment. We are also interested in investigating the various performance issues of the current framework when we apply it to image, video and general audio data. Developing analytical cost models that can accurately estimate the query costs in terms of space and time complexity is another interesting research direction.

6. REFERENCES

[1] C. Bohm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. In *ACM Computing Surveys*, pages 322–373, 2001.

[2] A. Chen, M. Chang, J. chen, J. L. Hsu, C. H. Hsu, and S. Huak. Query by music segments: an efficient approach for song retrieval. *The Proc. ICME*, 2000.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 24th VLDB Conference*, pages 194–205, 1997.

[4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. ACM Symp. PODS*, pages 102–113, 2001.

[5] C. Faloutsos. Signatures files. *Information retrieval: database structures and algorithms*, pages 44–65, 1992.

[6] E. Fox, D. Harman, R. Baeza-Yates, and W. Lee. Inverted files. *Information retrieval: database structures and algorithms*, pages 28–43, 1992.

[7] A. Ghias, H. Logan nd D. Chamberlin, and B. C. Smith. Query by humming: Musical information retrieval in an audio database. In *Proc. ACM Conference on Multimedia*, pages 231–236, 1995.

[8] Simon Haykin. *Neural Networks - A Comprehensive Foundation*. Prentice Hall, 1999.

[9] S. R. Jang, H. R. Lee, and J. C. Chen. Super mbox: an efficient/effective content-based music retrieval system. In *Proc. ACM Multimedia*, pages 636–637.

[10] I. T. Jolliffe. *Principle Component Analysis*. Springer-Verlag, 1986.

[11] I. H. Kang and G. Kim. Query type classification for web document retrieval. In *Proc. ACM Conference on SIGIR*, pages 64–71, 2003.

[12] R. L. Kline and E. P. Glinert. Approximate matching algorithms for music information retrieval using vocal input. In *ACM Multimedia*, pages 130–139, 2003.

[13] T. Li, M. Ogihara, and Q. Li. A comparative study on content-based music genre classification. In *Proc. ACM SIGIR Conference*, pages 282–289, 2003.

[14] C. C. Liu, A. J. L. Hsu, and A. L. P. Chen. Efficient theme and non-trivial repeating pattern discovering in music databases. In *Proc. 15th ICDE Conference*, pages 14–21, 1999.

[15] N. Liu, Y. Wu, and A. Chen. Efficient k-nn search in polyphonic music databases using a lower bounding mechanism. In *Proc. Multimedia Information Retrieval*, pages 163–170, 2003.

[16] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Information Systems*, 14(4):349–379, 1996.

[17] G. Peters, C. Anthony, and M. Schwartz. Song search and retrieval by tapping. *The American Association for Artificial Intelligence*, 2005.

[18] J. Shen, J. Shepherd, and A. H. Ngu. Integrating hetegrogeneous features for efficient content based music retrieval. In *Proc. 13th CIKM Conference*, 2004.

[19] D. Tao, H. Liu, and X. Tang. K-box: a query-by-singing based music retrieval system. In *ACM Multimedia*, pages 464–467, 2004.

[20] R. Typke, R. C. Veltkamp, and F. Wiering. Searching notated polyphonic music using transportation distances. In *Proc. ACM Multimedia*, pages 128–135, 2004.

[21] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. In *IEEE Transactions on Speech and Audio Processing*, pages 293–302, 2002.

[22] A. Uitdenbogerd and J. Zobel. Music ranking techniques evaluated. In *Australasian Computer Science Conference*, pages 275–283, 2002.

[23] R. Weber, H. J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th VLDB Conference*, pages 194–205, 1998.

[24] R. Yan, J. Yang, and A. G. Hauptmann. Learning query-calss dependent weights in automatic video retrieval. In *Proc. ACM Conference on Multimedia*, pages 548–555, 2004.

[25] C. Yang. Efficient acoustic index for music retrieval with varians degrees of similarity. In *Proc. ACM Conference on Multimedia*, 2002.

[26] C. Yu, B. C. Ooi, K. L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to knn processing. In *Proc. 27th VLDB Conference*, pages 421–430, 2001.

[27] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings ACM SIGMOD Conference*, pages 181–192, 2003.

[28] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Trans. Database Systems*, 23(4):453–490, 1998.