

# Compacting Music Signatures for Efficient Music Retrieval

Bin Cui\*

Department of Computer Science  
Peking University  
bin.cui@pku.edu.cn

H. V. Jagadish

Department of EECS  
University of Michigan  
jag@eecs.umich.edu

Beng Chin Ooi Kian-Lee Tan

School of Computing  
National University of Singapore  
{ooibc,tankl}@comp.nus.edu.sg

## ABSTRACT

Music information retrieval is becoming very important with the ever-increasing growth of music content in digital libraries, peer-to-peer systems and the internet. While it is easy to quantize music into a discrete string representation, retrieval by content requires (approximate) sub-string matching, which is hard.

In this paper, we present a novel system, called *MUSIC*, that uses compact MUsic SIGnatures for efficient content-based music retrieval. The signature is computed as follows: (a) each music file is split into a set of (overlapping) segments; (b) similar segments are clustered together; the number of clusters corresponds to the number of dimensions; (c) for each music file, the number of its segments that fall into a cluster determines the key value in that dimension.

Most index structures for multimedia are only able to provide an initial filtering and return a set of candidate answers that must be further examined. For *MUSIC*, we have also designed a scoring function that permits a ranked answer set to be generated directly based only on the signatures. Our experimental results show that this scheme retains a high degree of accuracy while being very efficient.

## 1. INTRODUCTION

There has been a trend of growing availability of music in digital form due to the advancement of digital technology in the last two decades. Today, many commercial websites are offering music download services. There are also efforts in some organizations to build digital libraries with large volumes of music for education and research purposes. The enormous and growing availability of music data has created new challenges in managing such data. A traditional way to organize the music data is to use auxiliary text information, such as the song title, the artist's name, etc. However, the effectiveness of such text-based search heavily depends on the ability to specify meaningful keywords, which may not

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EDBT'08, March 25–30, 2008, Nantes, France.

Copyright 2008 ACM 978-1-59593-926-5/08/0003 ...\$5.00.

always be possible. People often need to search music by the musical content instead. For example, a musicologist may want to use a few bars of music score to find any similar music pieces in a database or to find out whether a composition is original; a layperson may just hum a tune and let the system identify the song by searching a melody database.

Except the meta data (text) of the music files, we can extract the content information from the music files. Generally, the lowest-level representation with which we are concerned is the event: the pitch, onset, and duration of every note in a music source. And hence, the music file can be represented by a sequence of notes. We can further analyse the content of music, and extract the semantic features to represent music, such as timbral and rhythm content [23, 29]. In this work, we mainly focus on the first case, i.e. representing music as a sequence of symbols, each symbol representing a musical sound. With this abstraction, we can express the music retrieval problem as a sequence/string matching problem: each musical object in the database is a sequence, and a query is also a sequence – our task is to find database sequences of which the query is a subsequence.

Substring matching is used widely, and several data structures, including trie (or suffix tree) based structures [17, 20, 28], have been shown to provide effective indexing. Unfortunately, in the music scenario, the substring matches we seek are not exact. Most trie-based structures are of no use if even one symbol is altered or missing.

An alternative structure that has been studied in recent years is the notion of a  $q$ -gram [6, 27]. Each database string is divided into segments of length  $q$ . The set of unique  $q$ -grams presented in a string is then recorded. Given a query, we seek database objects that include all (or most) of the  $q$ -grams in the query. While  $q$ -gram techniques can be valuable, they have their limits, primarily with respect to the choice of  $q$ . Too small a value of  $q$  leads to too insufficient filtering and too many false hits. Too large a value of  $q$  very quickly becomes computationally intractable (the number of distinct  $q$ -grams grows exponentially with the value of  $q$ ). Furthermore, approximate matching becomes difficult with large  $q$  values.

In this paper, we propose an adaptive  $q$ -gram system, called *MUSIC* (*MUSIC* *SIGNATURE*). To generate music signatures, we first split a music file into a set of segments using a sliding window. We then cluster all the segments obtained from the music database using the time warping distance between segments. Finally, by treating the number of clusters as the dimensionality, and the number of segments of a music file within a cluster as its key value for that

dimension, each music file can be represented by a single high-dimensional vector (i.e., the music signature). While *MUSIG*'s efficiency comes from its compact representation, its effectiveness (accuracy) arises because the music signature is capable of distinguishing differences between music especially when the number of dimensions is high (each music has its own melody, and each falls into some clusters, and hence different music will end up with different features over the high dimensional space). The *MUSIG* data structures are described in Sec. 3.

We also design a scoring function to determine the match score between query and music data (both may be different in length) using their music signatures. With this scoring function, *MUSIG* can return not just a list of possible matches, but also rank order them. This scoring function and query algorithm are described in Sec. 4. We compare the proposed scheme against existing time series match methods and *q*-gram methods. Our experimental study, described in Sec. 5, shows that the *MUSIG* mechanism can provide a much faster response time, e.g. 30 times faster than traditional *Sequence match* approach, while maintaining high retrieval accuracy.

## 2. PRELIMINARIES

In this section, we first provide some background knowledge on music features, and then review some related work.

### 2.1 Music Background

Music is the art of organizing sounds produced by instruments or human voice. A piece of music comprises a succession of musical sounds. There are many characteristics of possible interest in a musical object. The most important of these is melody, which is a pitch (or frequency) function of time. Figure 1 (a) illustrates a melody representation using a *pitch curve*. After discretization, we obtain a *pitch line*, which is a sequence of horizontal line segments in the pitch versus time plot. Two or more consecutive music notes with the same note value correspond to one line segment and its length corresponds to the total duration of all the notes. Figure 1(b) illustrates the pitch line corresponding to the pitch curve in Figure 1(a). Characteristics of pitch line are: the pitch value is exact in semitones; the time duration is standardized in multiples of a small time unit, such as an eighth note.

There is a considerable body of work devoted to the extraction of a pitch line from digitized audio. There are two kinds of music with respect to the complexity of pitch-line extraction: in *monophonic music*, no new note begins until the current note has finished sounding and sources are restricted to one-dimensional sequences; in *Polyphonic music*, a note may begin before a previous note finishes. For monophonic music, most researchers assume independence between the pitch and duration of a note. These features are not truly independent, but the simplification makes retrieval much easier. For polyphonic music, researchers have found that a polyphonic source can be reduced to a monophonic source by selecting at most one note at every time step. This monophonic sequence of notes can then be further deconstructed using the monophonic feature selection techniques. The challenges include identification of a dominant "voice" from a polyphonic sound wave, and time warping to adjust for small variations in tempo. While not yet perfect, practical, effective techniques [11, 16, 24] exist today to perform

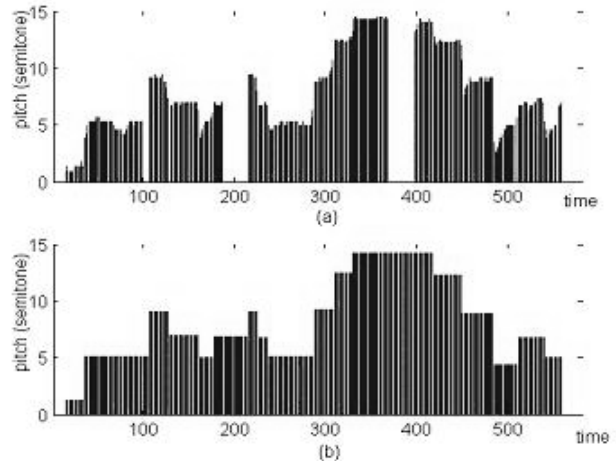


Figure 1: (a) Pitch curve for a hummed melody (b) Pitch line for pitch curve

this step. We build upon these in our current work.

The techniques described in this paper can be used with any string of symbols, and in particular, any choice of music represented as a string. For concreteness, we will focus on the pitch line representation in all our experiments and examples.

### 2.2 Related Work

Content-based music retrieval has attracted much research interest recently [8, 12, 13, 14, 15, 18, 21, 22, 23, 3, 2]. Several content-based retrieval techniques have been proposed in the literature, and they can be classified into two categories, i.e. Acoustic and Symbolic retrieval. The first category converts music data into indexable items, typically points in a high-dimensional vector space that represent music features, such as timbral texture, rhythmic content and pitch content [23, 29]. However, this approach cannot effectively support query by a short piece of music or humming, because the feature of the whole music piece may differ much from its short segment. The second class adopts symbolic representation based on the musical scores and notes in the score to keep track of musical information such as tone, pitch and duration. In this case, the problem of music retrieval can be transformed into approximate string matching or time sequence/series data matching [9, 10, 15, 18, 25, 26, 30, 31]. Unfortunately, these algorithms 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. To the best of our knowledge, there is no efficient index structure for long sequences. It is this problem that we address in this paper.

We mention here a small selection of papers on music retrieval that are closest to ours. The authors of [5] presented a similarity metric for continuous pitch contour using the crossing area between 2 pitch time sequences. However, how to achieve an optimal matching of 2 time sequences considering key transposition and tempo change is not addressed. In [9], a method was proposed to compare the continuous pitch contour of a hummed query with the melodies in the database. To tolerate tempo variations, this method uses

dynamic time warping distance for the comparison. To deal with key shifting, it uses a heuristic to estimate the key transposition by doing multiple dynamic time warping computations. A shortcoming of this technique is the heavy computation requirement. And the key transposition estimation by multiple trials is very inefficient. As a result, this method considered matching only at the beginning of a song in retrieval. A continuous dynamic programming method was proposed in [18] to compute the accumulated distance between a query time sequence and a target time sequence. This method handles the key transposition by assuming a correct start frame in the time sequence and distance measurement is based on that start frame. In [31], the authors treated both the melodies in the music databases and the user humming input as time series. Such an approach can integrate many database indexing techniques into an acoustic query, improving the quality of such system over the traditional string database approaches. They proposed a special searching technique called k-Local Dynamic Time Warping (DTW) that is invariant to shifting, time scaling and local time warping. A lower-bound distance for dynamic time warping was presented. The paper also introduces a framework for existing dimensionality reduction transformations, such as Discrete Fourier Transform and Discrete Wavelet Transform, to allow time warping. The algorithm only deals with whole sequence matching where the time series have same length, and  $\epsilon$ -range query is used on the index structure. However, the method cannot handle queries that are shorter than the represented segments in the database. Overall, the time series approach has been shown to be robust and effective for music retrieval using acoustic input, such as for query-by-humming. However, the computation cost can be enormous for a practical music retrieval system with a large database.

### 2.3 Problem Definition

We are given a large database of music pieces, each represented as a pitch line. We are given a query object, also as a pitch line, typically corresponding to a small fraction of a music piece – the match could be not necessarily at the beginning of the music, but anywhere in the piece. A typical example of such applications is the query that has been hummed by users.

Our task is to find quickly music pieces from the database that have a fragment, anywhere in the piece, similar to the query. The notion of similarity is not mathematically specified, but rather is in “the ear of the listener,” following the tradition in the information/multimedia retrieval rather than database community. In particular, “exact” matches must be found, compensating for musical errors made by the user in humming the query.

## 3. THE DESIGN OF *MUSIG*

### 3.1 Segment the Music

A single piece of music can be very long, often comprising hundreds of symbols. Consequently, effective manipulation of an entire piece of music is difficult. One solution is to split the music file into short segments. These segments can be semantically determined. However, this requires semantic knowledge, which may not be available to the segmenting program.

An alternative is that the segments can be determined

mechanically, as a sequence of a certain number of symbols. Furthermore, there is no reason for such mechanically created segments to be disjoint. Instead, it is preferable (as we can demonstrate experimentally) to have the segments overlap – each successive segment is shifted over by a specified window slide parameter. Figure 2 illustrates the segmentation process using a sliding window.

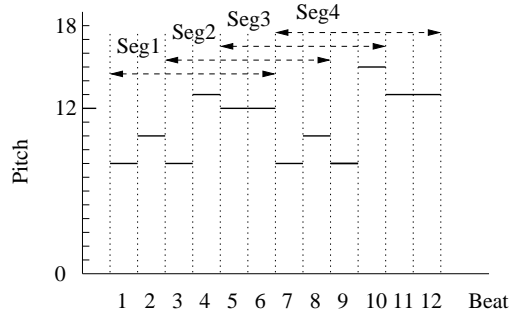


Figure 2: Segment the music with sliding window

The length of the window (6 in our example) and the window sliding step size (2 in our example) are both parameters to be chosen carefully. The smaller the window sliding step size, the larger the number of segments obtained, and hence the larger the representation, with associated storage and processing costs. In return, the smaller step size permits better matching with poor alignment of start position affecting query match less.

The length of the window has only a small effect on the number of segments (only at the ends of the long musical entry in the database). Here the issue is one of over or under specification. Since our retrieval does not consider the relative order of the matched windows, if the length is too small, we will get too many spurious matches. On the other hand, with too long a window approximate matching becomes difficult, as the query music can be much shorter than original music in the database. We will carefully study choices of these parameter values in the experimental section below.

### 3.2 The Case for Signature Method

If we treat each beat as one dimension and the pitch value as key of the corresponding dimension, each segment created above can be represented by a multi-dimensional vector. In the example, the window sliding step size is 2 beats, and hence the music piece can be represented by four segments, (8, 10, 8, 13, 12, 12), (8, 13, 12, 12, 8, 10), (12, 12, 8, 10, 8, 15) and (8, 10, 8, 15, 13, 13); in other words, 4 points in a 6-dimensional space. It is possible for us to conduct music retrieval using multi-dimensional index structures. For each segment of a music query, we conduct a sub-query to search the similar melody in the music database, and then merge the query answers. Such an approach has been explored in [31, 4]. However, this approach has three major drawbacks:

- It incurs a high computational cost. Because each music melody in the database has been split into many points, the database size can be large. For example, the average length of music in our experiment is about 400 beats. Assume we set the window size to be 8 beats and the window sliding step size to be 4 beats, each music will be represented by 99 8-dimensional points.

Additionally, since the music query is also segmented, there will be more distance computations and data accesses due to the existence of sub-queries.

- It is difficult to control the size of each query, as the query consists of several sub-queries. Note that, the query melody is typically longer than the specified window size. If the window size is large, it degrades the query performance due to the dimensionality curse. On the other hand, if we choose small window size, there is a higher possibility that more candidates would be retrieved for each sub-query to improve the accuracy of music retrieval (since it is more likely for similar short music portions to exist between two different music).
- As a music query may start anywhere in a music melody, the segments of database may not be consistent with the start of query melody. Therefore, some potential answers may be excluded. To minimize this, we can use small window sliding step size but it introduces more sub-queries (and hence higher processing cost) as well as higher storage overhead.

To address difficulties such as the above,  $q$ -gram techniques have been suggested. For each musical object, we can record a vector of the  $q$ -grams present in it. Given a query, we can develop a vector of the  $q$ -grams in the query. Then we can find vectors in the database that completely cover the query vector – these are data objects with all the right  $q$ -grams.

These  $q$ -gram vectors do indeed constitute a signature technique. However, the length of the signature vector is equal to the number of possible  $q$ -grams, which in turn grows exponentially with  $q$ . As such, the signature is practical only for small segments (with small values of  $q$ ).

In the case of text strings, a value of  $q = 3$  is usually quite adequate. However, a music segment is likely to have many repeated notes (since we discretize at a fine granularity of beat, most notes are held for longer than the minimum beat size). Also, musical phrases very frequently use immediately neighboring notes. So considering very short segments is not very useful. This leads us to seek a scalable signature technique.

### 3.3 Generate the Music Signature

To generate the signatures for entries in a music database, we first organize the individual segments obtained from the database into clusters of similar segments. The segments corresponding to a single piece of music will appear in multiple clusters.

With this clustering done, we can compute the signature for a piece of music as follows: (a) Let the number of dimensions be the number of clusters, and cluster  $i$  corresponds to dimension  $i$ . (b) Let the key value of the  $i$ -th dimension be the number of segments of the musical piece that belong to the cluster. Suppose a music file has 10 segments after segmentation, the whole music dataset is partitioned into five clusters, and the music signature S1 is (1, 2, 5, 0, 2). As shown in Figure 3, the signature S1 means that 1 segment of the music melody falls into cluster 1, 2 segments in clusters 2 and 5, and 5 segments in cluster 3, while there is no melody segment in cluster 4. Figure 4 shows the algorithm for generating music signatures for a music database.

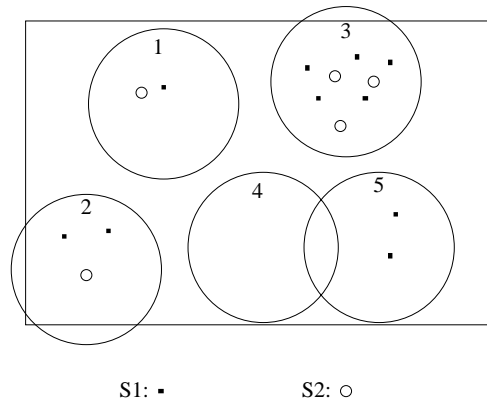


Figure 3: Clustering the segments

#### Algorithm Gen\_sig(dataset)

Input: music dataset

Output: music signatures

1. initialize signatures;
2. convert each music file into pitch lines;
3. set window size and window sliding step size;
4. for each pitch line
  5. split the pitch lines into segments;
  6. segment\_ID = music\_ID;
7. each music is represented by a set of segments;
8. cluster the segments using time warping distance;
9. for all the segments
  10. if segment belongs to cluster  $i$
  11. signature[segment\_ID].[i]++;

Figure 4: Algorithm for generating music signatures

The algorithm is quite straightforward. In line 1, we initialize the music signatures. Then all the music data is converted into pitch line format. In lines 3-7, we segment the pitch lines, and represent each music as a set of segments. Each segment of music has the same ID number as the music. Next, we split all the segments into clusters, and each segment belongs to the nearest cluster. We employ the K-means clustering scheme [7] to generate the clusters. Finally, for all the segments, we count the number of segments in each cluster (lines 9-11). Therefore, after the processing, we can get the signatures for all the music files.

Different from the traditional Euclidean distance metric used in K-means, we adopt time warping distance to effectively capture the difference between music segments. Time warping is an algorithm for measuring similarity between two sequences which may vary in time or speed [19]. The implementation of computing the time warping distance can be visualized as a string matching style dynamic program with quadratic complexity. The time warping distance between two sequences  $\vec{X}$  and  $\vec{Y}$  is defined as follows, where  $H()$  is the first element of sequence and  $R()$  is the rest of sequence:

$$D_{warp}(\vec{X}, \vec{Y}) = D(H(\vec{X}), H(\vec{Y})) + \min \begin{cases} D_{warp}(\vec{X}, R(\vec{Y})) \\ D_{warp}(R(\vec{X}), \vec{Y}) \\ D_{warp}(R(\vec{X}), R(\vec{Y})) \end{cases}$$

Two performers rendering the same piece of music will have small differences in their respective renditions. The discretization and quantization can also introduce small differences. In consequence, the problem we need to address is not one of substring exact match, but rather one of substring approximate match. Clustering similar segments together provides an avenue to allow for approximation. The specific similarity functions used for this clustering do not affect any other part of the *MUSIG* system. Hence these functions can be made as sophisticated as desired. The number of clusters to choose is another optimization parameter. The larger this number, the greater the storage and processing cost, and also the less forgiving the index structure is to approximations. On the other hand, too low a number will not provide sufficient discrimination.

Please note that there are two very distinct data spaces being considered, with very different properties and semantics. In the first data space, each music is represented by a set of segments. The second data space is a multi-dimensional space which has a segment cluster as a dimension, and a piece of music as a point in this space. The first space is used for segment clustering, while the second space is the one used at query time.

## 4. QUERY ALGORITHM

### 4.1 Signature as Filter

Having created a signature for each piece of music in the database, we would like to be able to use it to eliminate most of the database very quickly in response to a query, and zoom in on a few promising candidates.

To accomplish this, we extract the music signature for the query at hand, using the same set of clusters as for the music database. This signature must now be compared against the signatures in the database. Since we would like to find database objects for which the given query is a substring, it must be the case that all the segments in the query are also in the matching database object. We can rapidly eliminate any database objects for which this is not the case. The remainder are the match candidates to be returned.

There are a couple of issues to consider in this regard. First, there is a potential phase matching problem. Suppose we are using a window sliding step size of 2, but the query matches perfectly after shifting one position. In such a case, no segment may perfectly match at all, although time warping distance may alleviate this suffering. To address this possibility, we must generate as many versions of the query as the window sliding step size, each version shifted by one position more than the previous. The results to be returned are the union of the matches from the various versions. While our implementation examines all query versions, for ease of presentation, in the rest of this section, we shall only discuss how a single query version is processed.

The second issue is one of approximation. Due to errors/choices made in the rendition and/or in the discretization, we may desire to report approximate matches. To a large extent, this issue of approximation is already taken

care of at the time of cluster creation. Two segments that are similar are likely to be in the same cluster: small differences between query and database sequences will thus get masked. If desired, we may choose to further loosen the approximation threshold for query. While such a permissive notion will catch more approximate matches, it may also let through many mismatches. So this relaxation should be used with care.

### 4.2 Match Score

Traditionally, multimedia index structures have only been used as a rough cut first stage in the retrieval process to return candidates for further examination. This is not out of choice, but rather because the index structures can only be devised for simplified representations, so that more expensive sophisticated processing can be carried out on promising candidates.

In the case of *MUSIG*, we have the compact representation of the music and there is really no need to have such a two-stage query processing strategy. Rather, we can directly support a music information retrieval system. The only additional thing required then is to have a scoring function that can be used to rank the returned results. We introduce such a scoring function next.

Given two music signatures, data  $S1$  and query  $S2$ , with  $D$  dimensions, the overall match score  $MS$ , is the accumulated score over the whole dimensionality.

$$MS = \sum_{i=1}^D score_i$$

where

$$score_i = \begin{cases} S2_i & S1_i \geq S2_i \ \& \ S2_i > 0 \\ -S1_i & S1_i \geq 0 \ \& \ S2_i = 0 \\ -\infty & else \end{cases}$$

There are three different cases that need to be considered when we compare the values of  $S1_i$  and  $S2_i$ :

1. Both  $S1_i$  and  $S2_i$  are larger than 0 and  $S1_i$  is greater than or equal to  $S2_i$ . This means that both query and database have segments that belong to the  $i^{th}$  cluster. The  $S2_i$  segments in the query are all matched, so we increase the score by  $S2_i$ . There may be additional repetitions of the same segments in the database entry, since  $S1_i$  could be larger, but these additional repetitions do not contribute to the score.
2.  $S1_i$  is larger than or equal to 0 and  $S2_i$  is equal to 0. This case is for a segment that occurs in the database entry, but not in the query string. From a strict substring match perspective, this is not a problem – the target database string can contain all sorts of additional material, but should still be returned as long as it also includes the query string. However, it is well-established in information retrieval that the more irrelevant information a document contains, the lower it should be ranked. Otherwise long documents, which could include “everything”, will always match every query. Therefore a mismatch penalty is appropriate.

Furthermore, we note that music has a repetitive nature – the “important” phrases (or “motifs”) are repeated many many times, and these are the ones most

likely to be queried. As such, it is appropriate to have the mismatch penalty be a decrease of the score by  $S1_i$ . If this is some unimportant part of the musical piece that wasn't included in the query, the penalty is negligible. However, if it is a crucial part, then the fact that it is missing in the query should attract a heavy penalty.

3. In the last case,  $S1_i$  is less than  $S2_i$ . We can verify that  $S2$  is not the answer and prune it away. This is because  $S2_i$  cannot be greater than  $S1_i$  if  $S2$  is the sub-melody of  $S1$ . In our implementation, we use a sufficiently large value (a value that is larger than the maximum value that any of the dimensions can take) to represent  $-\infty$ . This penalty may appear to be overly harsh. However, our experiments show that it leads to better results than milder penalties for this case.

### 4.3 Matching Algorithm

#### Algorithm search(signatures, query melody)

Input: Music signatures, query melody

Output: Music IDs with K highest match scores

1. initialize answer list;
2. convert the query melody into pitch lines;
3. split the pitch line into segments;
4. for each segment
  5. find nearest cluster i;
  6. signature[i]++;
7. for each signature in database
  8. compute the match score using scoring function;
  9. if  $MS$  is larger than K-th candidate
  10. adjust answer list;

Figure 5: The search algorithm of *MUSIG*

Figure 5 shows the basic query algorithm of our proposed *MUSIG* scheme. Given a music query, either a sub-melody or hummed tune, we want to find the music, which exactly matches the query, from the database. After the music data processing, we have the signatures of all music files, and signature generation information, such as dimensionality, cluster centers, window size and window sliding step size. Note that, we apply the same parameters for music data and music query. We first initialize the answer list in line 1. In lines 2-6, we transform the query melody into signature format. Then, we compute the match score between the music data and query using the scoring function (line 8), and adjust the answer list if necessary. Finally, we return to the user the top ranked K answers with highest match score, i.e. the music which most probably match the query melody.

### 4.4 Probability Analysis

In this section we argue analytically that the probability of a signature match is usually low if there isn't a true match. Consider a music data object  $a$  being compared against some query music  $b$ . These objects have  $X$  and  $Y$  segments respectively, with  $X > Y$  typically. Suppose we have  $D$  dimensions after segment clustering, that all segments are randomly distributed among the clusters and the  $b$  is not a sub-sequence of  $a$ . If  $a_i \geq b_i \forall i \in [1...D]$ , where  $a_i$

and  $b_i$  are the number of segments in cluster  $i$  respectively, the data  $a$  will be retrieved wrongly according to the current matching score function. Let the probability of this kind of mis-match be  $P$ .

Given the dimensionality  $D$  and a sequence with segments  $n$ , we can have any number of segments between  $[0...n]$  on a certain dimension. Let  $p_i$  represent the probability that the dimension has  $i$  segments, we have

$$p_i = \frac{C_n^i \cdot (D-1)^{n-i}}{D^n}$$

If the data  $a$  and query  $b$  are mis-matched,  $a_i$  has to be larger than  $b_i$ , e.g. for one dimension  $d$  of sequence  $b$ , if the number of segments is  $n$ , the probability that  $a_i > n$  is  $\sum_{i=n}^X p_i^a$ , where  $p^a$  represents the probability on the sequence  $a$ . Therefore, we have

$$\begin{aligned} P &= p_{y=0}^b \cdot (p_0^a \cdot P(X, Y, D-1) + p_1^a \cdot P(X-1, Y, D-1) \\ &\quad + \dots + p_X^a \cdot P(X-X, Y, D-1)) \\ &\quad + p_1^b \cdot (p_1^a \cdot P(X-1, Y-1, D-1) + \dots \\ &\quad + p_X^a \cdot P(X-X, Y-1, D-1)) \\ &\quad + \dots \\ &\quad + p_Y^b \cdot (p_Y^a \cdot P(X-Y, Y-Y, D-1) + \dots \\ &\quad + p_X^a \cdot P(X-X, Y-Y, D-1)) \\ &= \sum_{i=0}^Y (p_i^b \cdot (\sum_{j=i}^X p_j^a \cdot P(X-j, Y-i, D-1))) \end{aligned}$$

Simplifying the equation, we have

$$P(X, Y, D) = \begin{cases} 0 & : X < Y \\ 1 & : D = 1 \\ \sum_{i=0}^Y (p_i^b \cdot (\sum_{j=i}^X p_j^a \cdot P(X-j, Y-i, D-1))) & : \text{Otherwise} \end{cases}$$

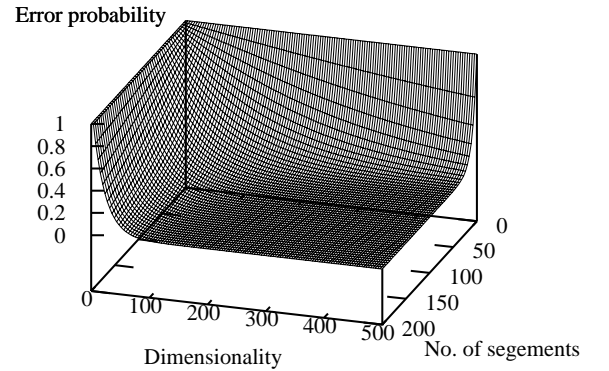


Figure 6: The error probability

Using the default parameter values from the experimental study below, we have  $D = 400$ ,  $X = 393$  and  $Y = 43$ . Using the above formula, we can compute the probability  $P$ , i.e.  $P \approx 6 * 10^{-8}$ . Clearly, the error rate of the matching score function is very low for randomly distributed data. Figure 6 shows the error probability with respect to the change of  $D$

and  $Y$ . We fixed  $X$  using the average music length in the database.

The figure clearly shows the effect of dimensionality and query length, i.e. when the dimensionality and the query length increase, the error probability decreases. One promising finding is that the scheme already yields satisfactory performance for small query length if we provide sufficiently high dimensionality. For example, if we set the dimensionality 400, the error probability is less than 1% when the query length is larger than 15, which means we can provide high precision even for a short piece of query music.

## 5. AN EXPERIMENTAL STUDY

In this section, we present an experimental study to evaluate the proposed content-based music retrieval method, *MUSIG*. Given a query melody, either a portion of the original music in the database or a hummed melody, we try to find the original music that matches the query. For each experiment, we ran 100 different queries, and report their average for all performance numbers. There are two performance numbers that are of primary interest – one is response time, which measures the efficiency of the proposed technique; the other is accuracy, which measures the accuracy of the proposed technique. Accuracy is the percentage of times (out of the 100 test runs) that the correct answer is included in the returned result set. Note that in all our queries there is precisely one exact match perfect answer. The probability of its being included in the returned result is a function of the result set size – the larger the returned result, the higher this probability. The experiments have been conducted on a PC system with P4 CPU (1.8GHz), 512MB RAM, and Microsoft Windows XP Professional Operating System.

The music dataset used in the experimentation are music files downloaded from the Internet [1] or from CD collections, such as mp3, wav, MIDI formats. There are 3500 music files with a total duration of 246 hours. To construct the index and conduct query, we first extract the melody, i.e. pitch information, from the raw music data [24, 25]. To construct the pitch line for the melody, each note is converted to a set of horizontal line segments. The height of a line segment corresponds to the note value (absolute pitch) and the length of the line segment corresponds to note duration.

### 5.1 Parameter Tuning

The *MUSIG* scheme has three major parameters: the window size, window sliding step size and the dimensionality of signature. In the first set of experiments, we tune these three parameters. The window size  $W$  is varied from 4 to 32 beats (quarter note), the window sliding step size (WS) is varied from 1 to 4 beats, and the dimensionality of signature (D) is varied from 100 to 500. All the experiment parameters are listed in Table 1. In the experiments for parameter tuning, we fix the length of music query as 1/8 of the original music, and we use a result set size (or answer rank threshold) of 10. The bold numbers are default values used in the experiments.

#### 5.1.1 Effect of Window Size

First, we explore how the performance is affected by different values of window size. Figure 7 (a) shows the accuracy of the *MUSIG* scheme and the time to generate the music signatures for different window sizes.

Window size (W)	4, <b>8</b> , 16, 32
Window sliding step size (WS)	<b>1</b> , 2, 3, 4
Signature dimensionality (D)	100, 200, 300, <b>400</b> , 500
Length of music query (L)	$\frac{1}{16}$ , $\frac{1}{8}$ , $\frac{1}{4}$ , $\frac{1}{2}$
Answer rank	1, 5, <b>10</b> , 20

Table 1: Experiment Parameters

We observe that the performance of the *MUSIG* scheme improves with relatively small window size. As shown in Figure 7 (a), when the window size is larger than 16, the larger the window size the worse the performance. To understand this, consider that the average musical piece in our database is 400 beats long. Because the music query is only 1/8 length of the original size, it is only about 50 beats. When the window size is large, there are few segments after segmentation, e.g. 19 segments with window size 32, compared to 43 with window size 8. Also, a window of size 32 includes the majority of the query sequence within a single segment. Hence, the signature of the music query cannot represent the melody effectively. On the other hand, the performance degenerates when the size is 4 or less. When the segment is short, there is a greater possibility that two different pieces of music may share some portions with similar melodies. Therefore, the signature of the music query cannot efficiently distinguish the difference from that of the music in the database. The optimal window size, 8, which is slightly better than 16, is a compromise of these factors. The accuracy can be 98% in top-10 returns, i.e. in about 98% retrieval processes, the correct music is found within the top ten ranks.

Figure 7 (b) shows the total time to generate the music signatures from the entire collection of music in the database. The time includes the melody segmentation, clustering and signature generation. Among them, the cost of clustering dominates the overall cost, because K-means clustering algorithm is very expensive computationally and has cost proportional to both segment length and data size. Here, the window size is the length of segments for clustering. Clearly, we can see that the signature generation is most expensive when window size is equal to 32. The query response time is quite small, around 0.25s, and also approximately constant since we fixed the dimensionality of signature at 400.

#### 5.1.2 Effect of Window Sliding Step Size

In this experiment, we vary the value of window sliding step size. As we mentioned previously, the query melody is typically just a small portion of the whole melody and may begin anywhere in the original melody. If the window sliding step size is larger than 1, there could be some segments mismatched, therefore we generate as many versions of the query as the window sliding step size, each version shifted by one position more than the previous, and we return the union of the top matches from the various versions as the results. Figure 8 (a) shows the accuracy for the *MUSIG* scheme. The results clearly demonstrate the superiority when window sliding step size is 1. When the window sliding step size is small, more segments are generated for the music query, i.e. the query has more feature representatives. In the signature comparison with music data, *MUSIG* can constitute the distinction effectively between

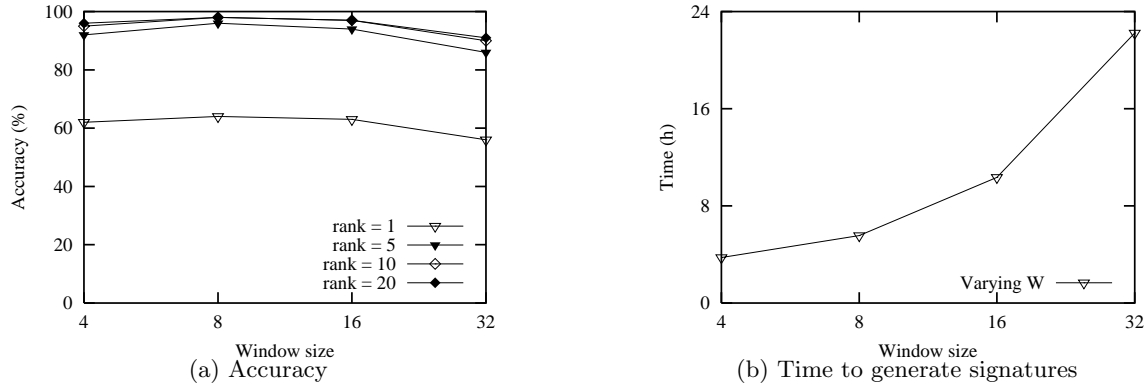


Figure 7: Performance with different window sizes

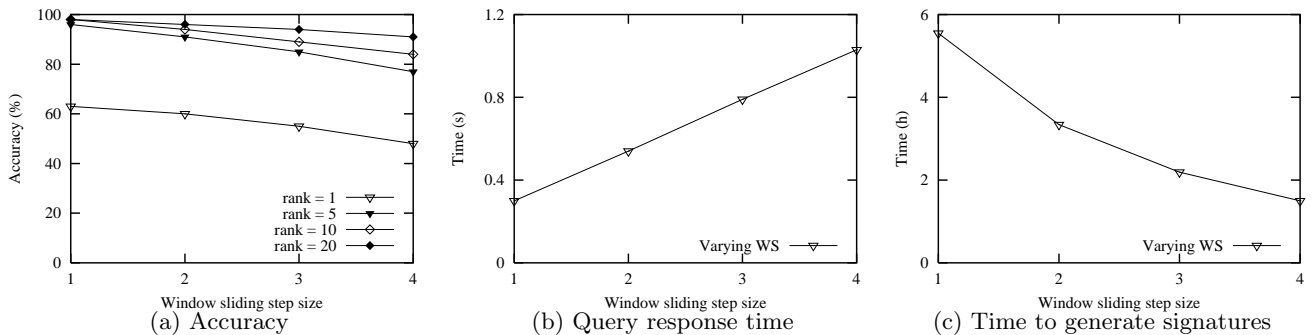


Figure 8: Performance with different sliding window sizes

query and data. On the contrary, the larger window sliding step sizes generate fewer segments, and hence insufficient discrimination. Figure 8 (b) shows that the query response time is almost proportional to the window sliding size, because we have to cluster different versions of queries and search database respectively.

Although the small window sliding step size introduces higher cost to generate the music signatures as shown in Figure 8 (c), we still prefer to set the window sliding step size 1. Because we only need to generate the music signature for the music database once, and the query response time is significantly superior when the window sliding step size is 1.

### 5.1.3 Effect of Signature Dimensionality

Here we present one representative set of experiments that studies the effect of signature dimensionality, i.e. we vary the dimensionality from 100 to 500. Figure 9 (a) shows the accuracy with varying answer rank for the *MUSIG* scheme. We observe that as the dimensionality increases, the accuracy increases. When the dimensionality of signature is high, which means that more clusters are generated for melody segments, each cluster has smaller size and music signature can represent the music feature more correctly. For example, when the cluster size is large, two different segments may be contained in the same cluster, thus *MUSIG* cannot distinguish between these two melodies only using the signatures. But if we further split the cluster and two segments are separated into two clusters, then *MUSIG* can differentiate between them easily. As shown in the figure,

the accuracy for top-10 return is increased from 73% to 98% when we increase the dimensionality from 100 to 500. Note that, the performance of *MUSIG* becomes stable when dimensionality is larger than 400 in terms of precision. The query response time and signature generation time are presented in Figure 9 (b) & (c). As we said previously, the query time and signature generation time are proportional to the dimensionality, and the cost is highest when dimensionality is 500.

In the following experiments, we shall compare the *MUSIG* method with other sub-sequence match approaches. Note that, the optimal parameters, such as window size and signature dimensionality, may vary for different datasets. However, for clarity of presentation, we use the optimal parameters determined above as default, i.e. window size is 8, window sliding step size is 1 and signature dimensionality is 400. We set dimensionality as 400, because it yields similar accuracy but better query performance compared with 500 dimensions.

## 5.2 Compare with Other Structures

In this section, we evaluate the various schemes with different query inputs. We compare against  $q$ -gram method [6, 27], and we set  $q$  equal to 3. We also compare our method with sequence similarity matching methods [18, 30, 31]. With the melodies properly transposed, a sequence similarity measure can be used for melody matching, and time warping distance can be a good similarity measurement. The distance is computed by accumulating the local

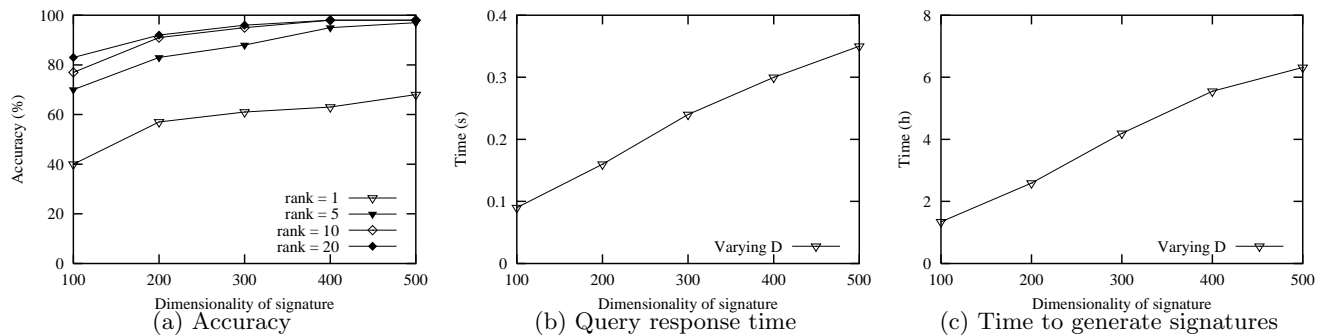


Figure 9: Performance with different signature dimensionalities

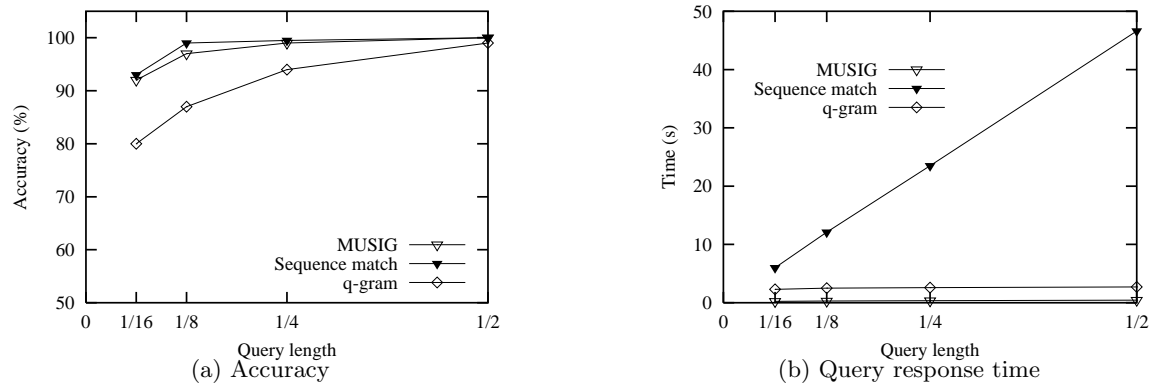


Figure 10: Effect of query length

pitch distance between two sequences using a dynamic programming algorithm. We name this kind of approach as *Sequence match*.

### 5.2.1 Effect of Query Length

In this experiment, we compare the *MUSIG*, *q*-gram and *Sequence match* approaches with varying lengths of music query, which are  $\frac{1}{16}$ ,  $\frac{1}{8}$ ,  $\frac{1}{4}$  and  $\frac{1}{2}$  of the original music respectively.

Figure 10 (a) shows the accuracy of the three schemes. We can see that *MUSIG* can almost perform as well as *Sequence match*. Especially when the query length is large, e.g.  $> 1/4$ , the accuracy is higher than 99% for both methods. However, when the query length is short, the performance is degraded for both methods. This is because for a short query, there is a higher possibility that the music query is similar to more music pieces in the database. The query length affects the performance of *MUSIG* more significantly. In *MUSIG*, we split the music into segments to generate the signature. When the music query is short, we have fewer segments, and hence the music signature cannot represent the music feature effectively. The music signature introduces some information loss due to its compact representation compared with sequence representation of the melody. However, it is only about 1% worse than *Sequence match* approach even when the query is  $1/16$  of the music. Clearly, *q*-gram performs worse than the other two methods. This is expected as the value of *q* is relatively small, and segments

of different music may share the same *q*-grams. Hence the *q*-gram cannot distinguish the difference between two music efficiently. Although the *q*-gram scheme can provide better performance when we choose larger *q*, such a choice is impractical because the cost will increase exponentially.

Figure 10 (b) shows the comparison in terms of query response time. As shown, *MUSIG* can reduce the time cost to only about 1/30 that of the *Sequence match* method. All the query operations of *MUSIG* are very efficient in terms of CPU cost. Additionally, the signature representation is very compact, and the I/O cost is also low. Considering 2660 music files with signature dimensionality 400, the space required is only 4.1 MBytes. The *Sequence match* method is computationally expensive. First, it needs to scan all the transformed music sequence for comparison. Although in [31], the authors proposed to enhance the *Sequence match* approaches by using R\*-tree as a filter, it is not suitable for music melody matching. This is because different pieces of music in the database may have different lengths, and generally the average length of the music is large, e.g. around 400 beats long for the music data used in this paper. Additionally, the query is usually a sub-sequence match. It is not feasible to get correct filtering condition from a short melody for the R\*-tree to prune the data properly. Second, the music query can begin anywhere in the original music, greatly increasing the number of sub-sequence matches to be performed. Note that the starting position of query does not affect the query performance of the *MUSIG* mecha-

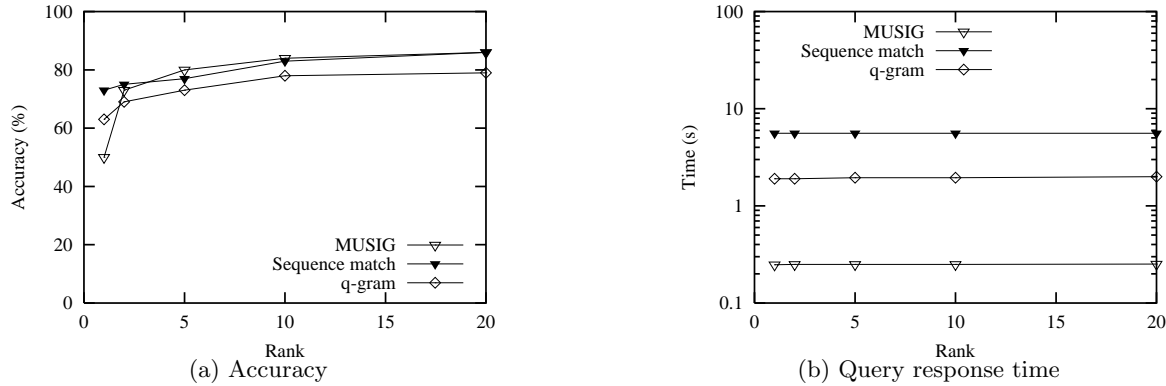


Figure 11: Performance with hummed query

nism. The  $q$ -gram also runs much slower than *MUSIG*. Although it can generate  $q$ -grams of the music query faster, the comparison incurs a much higher cost because of the many  $q$ -grams in the database.

### 5.2.2 Performance with Hummed Query

Thus far, all experiments reported were with the query from the database itself, and possibly being perturbed. In this section, we report on experiments performed with query melodies that were generated externally. Five subjects, 3 male and 2 female, were asked to hum a snippet each of a hundred different melodies in the database. None of the subjects is a professional singer. The time duration of a hummed snippet was about 15 - 20 seconds. The humming voices are recorded through microphone using 44100 Hz 16 bit waveform (PCM) format. Pitch extraction is the first step in doing music retrieval using hummed melody. The pitch extraction result for a melody query is a sequence of pitch values where each value corresponds to a frame of size 2048 samples (46 millisecond). Then we conduct pitch curve aggregation to generate pitch line with time duration standardized in multiple of a small time unit, such as a quarter note. We analyze the signal energy and detect vowels in the humming to estimate the tempo of hummed melody [11].

Figure 11 shows the performance for the hummed query. All techniques performed worse than the results shown in section 5.2.1. There are at least two reasons for this. First, the hummed query is much shorter ( $< 1/10$ ) compared with the music in the database, where the average music duration is 4.3 minutes. Second, some hummed queries had very poor quality, due to inability of the subject to carry a tune. Nevertheless, the performance of *MUSIG* is quite good, with accuracy even exceeding that of *Sequence match* in many cases. This is because the clustering scheme of *MUSIG* can mask some faults in a hummed tune, as long as the error melody does not get switched over to a different cluster. On the other hand, the clustering of *MUSIG* also introduces some information loss, and different music files may share same signature, which explains the poorer performance for the first three ranking positions.

Additionally, *MUSIG* is superior over *Sequence match* approach in terms of the query response time for hummed queries as well. It can get the answers around 30 times faster than *Sequence match* method because of its efficient process over the music signatures.

### 5.2.3 Dataset with Noise

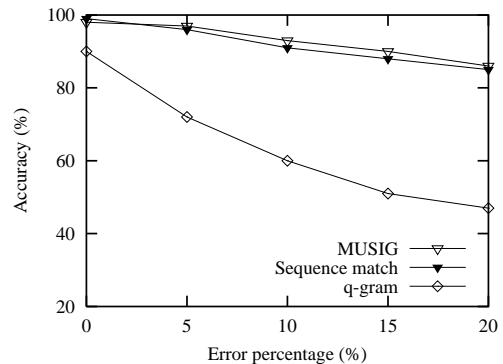


Figure 12: Effect of noisy data

Different renditions of the same musical piece can have subtle differences in pitch and tempo. Amateur singers may produce less subtle errors in pitch. The pre-processing techniques that extract pitch lines from acoustic signals can often mask such small variations. However, larger variations do get through, the pre-processing itself introduces the possibility of quantization error. As such, it is important for any music retrieval technique to deal effectively with noise.

To model noise, we randomly perturb some fraction of the notes in every query, e.g. note insertion/deletion and note value change. Figure 12 shows the results, with the percentage of notes perturbed listed along the X-axis. We find that all the methods yield worse performance when the data has more noise. The  $q$ -gram technique is particularly hard hit, because each pitch error affects several  $q$ -grams of the music. *MUSIG* and *Sequence match* do much better, still shoeing an accuracy of over 85% at a 20% error rate. The low error in *MUSIG* is on account of the clusters not being affected too much by noise.

### 5.2.4 Effect of Scalability

To test the scalability of our method, we seek a larger music database. Due to the limited number of music files, we extend our existing music database with synthetic pieces. We generate 100,000 random walk data sequences which are normalized into the domain of pitch value. Figure 13(a) shows the accuracy of three approaches. We can see that all

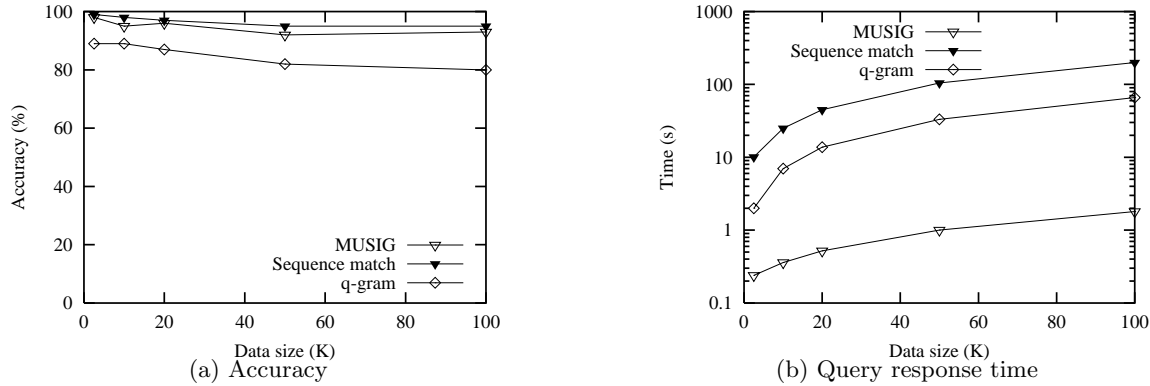


Figure 13: Effect of Scalability

the methods degenerate as the data size increases, because there is higher possibility that different data sequences share the similar segment. Furthermore, the larger dataset may affect the clustering effectiveness of *MUSIG*. However, the *MUSIG* mechanism still yields good performance, i.e., = 93% accuracy for large data size of 100,000. Although it is still 2% worse than *Sequence match* in terms of accuracy, its significant gain in response time over *Sequence match* (as shown in Figure 13 (b)) makes it a promising approach.

### 5.2.5 Effect of Insertion

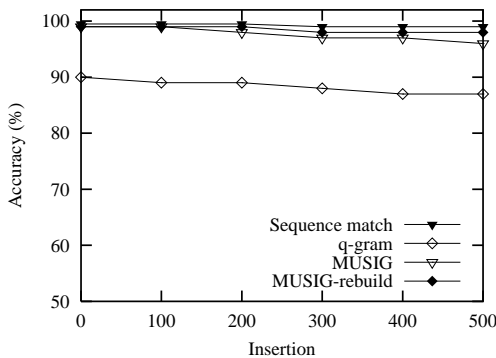


Figure 14: Effect of insertion

In this experiment, we study the effect of insertion on these schemes. The delete operations yield similar performance and we omit it. We use the music query with 1/8 length of the original melody. For *MUSIG*, we evaluate two versions: *MUSIG* represents the version that we use the existing cluster center information to generate signature for new music; and *MUSIG-rebuild* represents the version that always rebuild the tree upon insertion.

We randomly select the data from the music dataset, and first generate the signatures with 2000 music. Subsequently, we insert up to 500 more new music. We record the response time and accuracy of top-10 answers after 100 newly inserted melody. The two *MUSIG* versions run much faster than *Sequence match* and *q-gram* approaches, and we only show the accuracy in Figure 14.

First, we observe that both *MUSIG* and *Sequence match*

still yield good performance with new music inserted into database with  $> 97\%$  accuracy. Second, as more points are inserted, the performance of *MUSIG* degenerates as the newly inserted data affects the precision of cluster centers. However, the degradation of performance is marginal. Third, it is clear that the rebuilding strategy, *MUSIG-rebuild*, can reduce the performance degradation. The results show *MUSIG*'s robustness with respect to insertions in the sense that it can take sufficiently large number of insertions. Additionally, we can regenerate the signatures offline while not interfering with the other queries. This makes *MUSIG* a promising candidate even for dynamic datasets.

### 5.2.6 Performance on Polyphonic Data

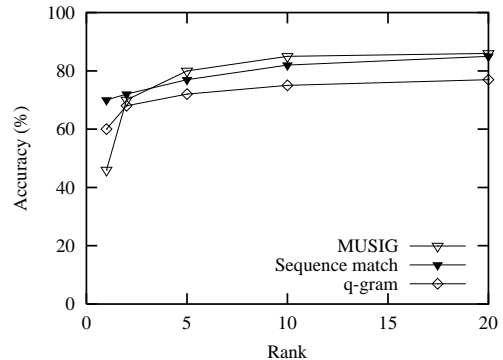


Figure 15: Performance on polyphonic data

As we introduced previously, polyphony poses more challenges to effective music retrieval. However, we can simplify the processing by pulling out an entire monophonic note sequence equal to the length of the polyphonic source. One approach is that the note with the highest pitch at any given time step is extracted. Since this approach yields most decent results [24, 25], we adopt the highest pitch in our experiments. We use hummed query to test the performance on the polyphonic data which consists of 1000 MP3 files.

Figure 15 shows the performance on polyphonic data. All the performances degrade slightly compared with the monophonic data, e.g.  $< 85\%$ , because extracting single pitch for each note introduces some information loss. However, the

performance of *MUSIG* is still satisfactory, with accuracy even exceeding that of *Sequence match* when the number of returned answers is larger than 5. This is because the clustering scheme of *MUSIG* can mask some faults in the pitch lines during polyphonic pitch extraction.

## 6. CONCLUSION

In this paper, we described a novel system, called *MUSIG*, that uses compact MUsic SIGnatures for efficient content-based music retrieval. To generate the music signature, we adopted a clustering approach that adapts to the distribution of the data in the database. We also designed a scoring function to determine the match score between a music query and a music data based on their music signatures. Conceptually, the *MUSIG* scheme combines the idea of  $q$ -grams with the idea of vector quantization. It appears to be particularly well suited to applications for substring matching where database strings have significant repetitions in them. Our experimental results show that the *MUSIG* scheme is very efficient while retaining a high degree of accuracy as compared to existing sequence match and  $q$ -gram techniques, even after a significant number of updates.

## 7. REFERENCES

- [1] Download karaoke files and players. In <http://www.schok.co.uk/kb/downloads.htm>, 2005.
- [2] B. Cui, L. Liu, C. Pu, J. Shen, and K. L. Tan. Quest: querying music databases by acoustic and textual features. In *Proc. ACM Multimedia*, 2007.
- [3] B. Cui, J. Shen, G. Cong, H. T. Shen, and C. Yu. Exploring composite acoustic features for efficient music similarity query. In *Proc. ACM Multimedia*, 2006.
- [4] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of SIGMOD Conference*, pages 419–429, 1994.
- [5] C. Francu and C. G. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *Proc. International Conference on Multimedia and Expo (II)*, pages 889–, 2000.
- [6] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. 27th VLDB Conference*, pages 491–500, 2001.
- [7] J. Hartigan and M. Wong. A K-means clustering algorithm. In *Applied Statistics, Volume. 28*, pages 100–108, 1979.
- [8] J. Hsu, A. Chen, H. Chen, and N. Liu. The effectiveness study of various music information retrieval approaches. In *Proc. 11th CIKM Conference*, pages 422–429, 2002.
- [9] S. R. Jang and H. R. Lee. Hierarchical filtering method for content-based music retrieval via acoustic input. In *Proc. ACM Multimedia*, pages 401–410, 2001.
- [10] 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.
- [11] K. Jensen and T. H. Andersen. Real-time beat estimation using feature extraction. In *Proc. the Computer Music Modeling and Retrieval Symposium*, 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] 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*, 1999.
- [14] C. C. Liu and Po-Jun Tsai. Tcontent-based retrieval of mp3 music objects. In *Proc. 10th CIKM Conference*, 2001.
- [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] M. Marolt. On finding melodic lines in audio recordings. In *Proc. 7th Int. Conference on Digital Audio Effects*, 2004.
- [17] E.M. McCreight. A space-economical suffix tree construction algorithm. In *Journal of the ACM*, 1976.
- [18] T. Nishimura, H. Hashiguchi, J. Takita, J. X. Zhang, M. Goto, and R. Oka. Music signal spotting retrieval by a humming query using start frame feature dependent continuous dynamic programming. In *Proc. International Symposium on Music Information Retrieval*, pages 301–307, 2001.
- [19] L. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.
- [20] H. Shang and T. H. Merrettal. Tries for approximate string matching. In *IEEE Transactions on Knowledge and Data Engineering*, pages 540–547, 1996.
- [21] D. Tao, H. Liu, and X. Tang. K-box: a query-by-singing based music retrieval system. In *ACM Multimedia*, pages 464–467, 2004.
- [22] R. Typke, R. C. Veltkamp, and F. Wiering. Searching notated polyphonic music using transportation distances. In *Proc. ACM Multimedia*, pages 128–135, 2004.
- [23] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. In *IEEE Transactions on Speech and Audio Processing*, pages 293–302, 2002.
- [24] A. Uitdenbogerd and J. Zobel. Manipulation of music for melody matching. In *Proc. ACM Multimedia*, 1998.
- [25] A. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music databases. In *Proc. ACM Multimedia*, 1999.
- [26] A. Uitdenbogerd and J. Zobel. Music ranking techniques evaluated. In *Australasian Computer Science Conference*, pages 275–283, 2002.
- [27] E. Ukkonen. Approximate string matching with  $q$ -grams and maximal matches. In *Theoretical Computer Science*, 1992.
- [28] E. Ukkonen. On-line construction of suffix trees. In *Algorithmica*, pages 14(3):249–260, 1995.
- [29] C. Yang. Efficient acoustic index for music retrieval with various degrees of similarity. In *Proc. ACM Multimedia*, 2002.
- [30] Y. Zhu and M. Kankanhalli. Music scale modeling for melody matching. In *Proc. ACM Multimedia*, 2003.
- [31] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proc. of SIGMOD Conference*, pages 181–192, 2003.