Audio Indexing for Efficient Music Information Retrieval*

Ioannis Karydis, Alexandros Nanopoulos, Apostolos N. Papadopoulos and Yannis Manolopoulos

Data Engineering Lab., Department of Informatics
Aristotle University, 54124, Thessaloniki, GREECE
E-mail: {karydis,alex,apostol,manolopo}@delab.csd.auth.gr

Abstract

This paper presents an algorithm that efficiently retrieves audio data similar to an audio query. The proposed method utilises a feature extraction method for acoustical music sequences. The extracted features are grouped by Minimum Bounding Rectangles (MBRs) and indexed by means of a spatial access method. We also present a novel false alarm resolution method that utilises a reverse order schema while calculating the distance of the query and results, in order to avoid costly operations. Performance evaluation results show that the proposed technique achieves considerable performance improvement in comparison to an existing method.

1 Introduction

The spread of digitized music as well as the development of digital music libraries gives impulse to the utilisation of retrieval methods other than the traditional metadata (title, composer, performer, genre, date, etc.) of a music object. As these information are not the music content itself but secondary features, content-based music information retrieval has greatly developed in recent years. Although, music retrieval based on humming is the most natural and spontaneous content-based music retrieval, retrieval can also be performed by providing a query music file or even by creating a query using a keyboard. Generally, content-based music retrieval requires an actual music piece in order to compare its content with the content of the music pieces already available in a database. The freedom of origin of the query object introduces the need for similarity searching due to possible errors in the query. In addition, the use of similarity searching is also underlined by the trend that songs are developed based on the variation of an original theme.

*This research is supported by the IRAKLITOS national program (2003-2005) funded by EPEAEK framework

One of the main challenges in Music Information Retrieval (MIR) is the choice of representation of the musical information within the system. A music object should be described by a set of its features. Numerous approaches exist on what features to retain and on how to select these features [19]. Music representation and consequently the feature selection can primarily be separated in two classes: the symbolic representation (MIDI format) and the acoustic representation (audio format - wav, mp3).

In the symbolic representation area, the MIDI representation is quite a common selection, while the set of features ranges from pitch [7], rhythm [2], changes in pitch [6] or even rhythm and pitch [21]. Recent research proposed that features could be represented in string format and accordingly presented string indices [8], [21], [2], [11]. Though, these approaches are not easily adopted for multiple features, lack data scalability for large music data [21] and generally string matching proves slower than numeric matching [10]. In order to address these inefficiencies [10] proposed a multi-feature numeric indexing structure that transforms music feature strings into numeric values.

As far as the acoustic representation is concerned the most common features are produced by time analysis([13], [14]), spectral analysis([13], [14], [9]) and wavelet analysis [18]. The coefficients collected from each of these analyses can be indexed in TV-Trees [17], locality-sensitive hashing schemes [22], S-Indexes [3]. In addition, [16] compares four different multidimensional indexing schemes for music data, the KD-Tree, the K-Tree, the Multidimensional Quick-sort and the Box Assisted Method. The authors conclude that KD-Tree is significantly more efficient than the other methods, especially for high-dimensional data. Finally, the authors in [20] utilise an M-Tree in which a selection of features is stored, claiming thus a 65% gain in space requirements.

In this paper, we focus on the problem of searching similar subsequences in music audio data using as features the first few DFT coefficients of the audio file (sequence). The resulting coefficients are stored in a spatial access structure

to decrease retrieval time. Key differences of audio data in comparison to existing approaches (mainly from the field of time-series analysis) require better testing as well as differentiated methods. For example, in common time-series applications, like stock-market analysis, query sequences have relatively short lengths, e.g., less than 1,000 elements. For music sequences, even in down-sampled raw audio files, a query would include at least three seconds thus producing a query sequence of approximately 60,000 elements. It is apparent thus, that the false alarm resolution of such a query could be computationally expensive. Moreover, compared to existing approaches in indexing music sequences for similarity searching, we are interested in an approach that will allow direct implementation in existing DBMSs. For this reason, we utilise index structures from the R-tree family, which have been implemented in several commercial and open-software DBMSs, e.g., Oracle and Postgres.

The technical contributions of this paper are summarised as follows:

- The development of a novel algorithm that efficiently retrieves audio data similar to an audio query. The proposed algorithm addresses the characteristics that result from the nature of the examined problem, i.e., factors like the increased size of the sequences handled (as mentioned, such factors do not appear in work in related fields like the similarity search in time-series).
- The detailed experimental results which show the efficiency of the proposed algorithm and the performance gains compared to an existing baseline algorithm [4].

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 provides a complete account of the algorithm proposed in this paper. Additionally, Section 4 presents an efficient algorithm for false-alarm resolution. Subsequently, Section 5 presents and discusses the experimentation and results obtained. Finally, the paper is concluded in Section 6.

2 Related work

2.1 CBMIR Systems

Current related work on acoustic data - acoustic query Content-Based Music Information Retrieval (CBMIR) systems is rather limited. The author in [22] propose a spectral indexing algorithm for CBMIR. Its feature-extraction process attempts to identify distinctive notes or rhythmic patterns. The features are used to construct "characteristic sequences", which in the next step are indexed in a probabilistic scheme, the Locality-Sensitive Hashing (LSH). The LSH scheme allows both false positive and negative matches, that are compensated in a later step based on the uniformity in time of music tempo changes. Experimental results indicate high retrieval accuracy for different similarity types. In [20], the authors propose a CBMIR system that is mainly oriented towards servicing different types of queries. The acceptable query types include audio files, common music notation as well as Query-By-Humming (QBH). The MIDI format is used as an intermediate music object representation. The selection of features is called "representative melody" and is register into an M-tree structure, in which melodies are inserted based on their average length and pitch variation together with melody signatures representing the variation pattern. The used distance is a time-wrapping function. Preliminary results indicate 65% gain in space requirements when using the collection of features instead of the whole melodies.

As far as the work in [20] is concerned, it's main disadvantage is the assumption that the users' query must include at least one of the parts that they gather in order to create the "representative melodies". And as this might work for QBH, it might not for a random piece of a music file included in the index, especially for a small one. In addition, polyphonic music transcription is known to be very hard and poor performing([22], [15]). Regarding the work in [22], its feature selection mechanism is oriented towards identifying different types of similarity in music pairs. Additionally, the selected features can lead to false negatives, which have to be addressed in a post-processing step. Finally, [22] uses a specialized indexing mechanism. In contrast, we focus on a much simpler, but useful, model for searching similar subsequences, which is based on existing work on time-series analysis. Our approach does not introduce false negatives, according to the used similarity model, and, more importantly, it uses general purpose indexes (R-trees), which allow for a direct implementation in existing RDBMSs.

2.2 Multimedia Similarity Indexing

The GEneric Multimedia object INdexIng (GEMINI) approach [5] consists of a feature extraction function to map objects into points in f-dimensional space. Accordingly, a Spatial Access Method (SAM) method is used to accelerate search. The GEMINI approach is based on the following three key issues: (i) a fast, possibly allowing false alarms test, to discard the majority of non-qualifying objects, (ii) the use of a SAM to improve search performance, and (iii) the use of a false alarm resolution method.

One of the most popular spectral analysis for time sequences is based on the Discrete Fourier Transform (DFT). Retaining the first few coefficients as features leads to a truncation that under-estimates the distance of the sequences, thus introducing no false dismissals [4]. The popularity of this approach emerges from the fact that most real

sequences fall in the class of random walks, and colored noise in particular. For this kind of random walks, the first few coefficients of the DFT transform contain most of the energy of the sequence. This phenomenon is quite apparent in stock sequences, which can be considered as brown noise. Interestingly enough, it has been reported that it also holds for signals like audio data, which can be characterized as pink noise [4]. However, none of the existing approaches exploites the aforementioned issue.

Compared to the method proposed in [4], our approach differs in the following issues. Query sequences may have large lengths compared to the ones used in time-series data and stock-market sequences, that were examined in [4]. For this purpose, we developed a different mechanism for resolving false alarms, which takes into account the aforementioned factor. Moreover, [4] focuses on the indexing of one or some few sequences. In our approach, we index a database which may contain a large number of music sequences.

3 The Proposed Method

In this section we define the problem addressed by this research and present the proposed method in terms of the feature selection and extraction as well as the spatial access method utilised.

3.1 Problem statement

The problem definition is as follows: Let D be a collection of n musical sequences, i.e., $D = \{D_i\}, 1 \leq i \leq n$. Given a musical sequence Q, find all $D_i \in D$ where each such D_i contains at least one subsequence S_j of length $|S_j| = |Q|$ and $||S_j - Q|| \leq \epsilon$ (ϵ is user-defined). We use the Euclidean distance between of Q and S_j.

Example. Let the collection of sequences D be the one depicted in Figure 1, containing three music sequences. For a given query subsequence Q (also depicted in the figure) and for $\epsilon = \sqrt{5}$, we find a match in D_1. The corresponding subsequence is depicted within a solid rectangle. Notice that D_1 also contains another subsequence of length three (the one with elements 23, 17, and 31), which matches Q. However, since we need only to report that D_1 contains a match, the first found match suffices. □

It should be noted that in this work, to obtain musical sequences, wav audio files are used. Initially, the files are down-sampled to 22050 Hz, transformed to single-channel audio and 8-bit representation of each sample. Then their header is removed in order to retain only the audio information part. Thus, we consider a musical sequence to be a

D1: 0 0 12 25 18 32 12 23 17 31
D2: 0 1 13 12 28 35 19 58 92 14
D3: 2 5 67 96 55 44 28 128 116 35

Q: 24 16 32

Figure 1. Example of subsequence similarity matching.

sequence of integers ranging from 0 to 255, that describes the signal amplitude of the music file.

3.2 Feature Extraction

The selection of appropriate features is considered very important in multimedia information retrieval. Meaningful features help in the effective representation of the objects and enable the use of indexing schemes for efficient query processing.

We apply the feature extraction process proposed in [4], since the problem we are dealing with is similar to subsequence matching in time-series. Therefore, the original audio sequence is transformed to a number of multidimensional points by applying a sliding window to the audio data and by applying the Discrete Fourier Transform (DFT) to each part. Therefore, each audio sequence produces a set of multi-dimensional points. The dimensionality of the transformed space depends on the number of DFT coefficients that will be used for the representation. By keeping the first few DFT coefficients the size of the original audio sequence is reduced significantly. Moreover, much of the audio sequence energy is concentrated in the first few DFT coefficients [4] and therefore adequate representation is achieved.

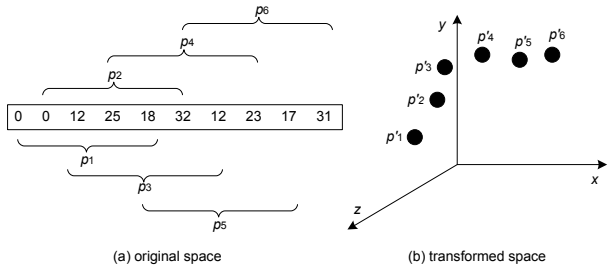

Figure 2. Feature extraction process.

An example of the aforementioned transformation technique is illustrated in Figure 2. A sliding window of size five is applied to the original audio sequence. Each point p_i defined by the sliding window is transformed to a point p'_i

in the 3-d space by applying the DFT and keeping only the first coefficients of the transformation. It has been proved in [4] that no false dismissals are introduced by using this transformation technique, due to the fact that the distance in the transformed space is lower-bounded. However, false alarms are possible and they must be resolved. The false alarm resolution method is analysed in Section 4.

3.3 Indexing

The audio representation illustrated in the previous section cannot guarantee efficient query processing. Therefore, the transformed audio sequences must be organized by means of an indexing scheme, towards increased processing efficiency.

Due to the fact that each audio sequence is represented by a set of multi-dimensional points, a multi-dimensional access method can be used to organize the data. However, indexing directly the multi-dimensional points will lead to huge storage consumption because each audio sequence can generate thousands of multi-dimensional points. To attack the problem we apply an approach similar to the one proposed in [4] which performs a grouping of the multi-dimensional points to minimum bounding rectangles. Therefore, we take advantage of the fact that consecutive multi-dimensional points are expected to be close in the transformed space. An example of the packing process is illustrated in Figure 3.

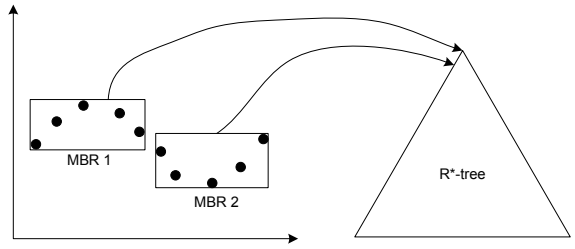

Figure 3. Packing and indexing scheme.

The number of produced Minimum Bounding Rectangles (MBRs) is significantly less than the number of multi-dimensional points. Therefore, MBRs can be organized by means of an R*-tree [1] or any other multi-dimensional access method. We focus on the R*-tree access method because it has been consistently used in many applications and has been already implemented by commercial database vendors.

3.4 Similarity Range Search

The user query is composed of a query audio sequence Q and a distance threshold e. The similarity query processing method is composed of three major steps which are briefly described in the sequel:

Step 1: The query audio sequence Q is transformed by means of the aforementioned transformation technique. If Q is larger than the sliding window size w, then it is split to k parts q_1, q_2, ..., q_k, where $k = \lceil \frac{|Q|}{w} \rceil$.

Step 2: The query parts determined in the previous step are used to search the R*-tree index. The result of this step is a set of audio sequences that *may satisfy* the query constraints.

Step 3: The last step involves the refinement of the answers returned by the index. This is performed by a novel false alarm resolution algorithm which is studied in detail in the subsequent section.

4 False Alarm Resolution

To resolve a false alarm, we must retrieve the corresponding subsequence and examine its actual distance against the query sequence. The algorithm in [4] uses a direct false-alarm resolution mechanism. Whenever an MBR is found to satisfy the range query, its subsequence is fetched and examined against the query sequence. For the context examined by our approach, the aforementioned method is inefficient due to two reasons:

i. Query sequences are much larger compared to the ones in the context of [4] (stock-market data). Therefore, the cost of naively resolving each false alarm can become the bottleneck of the entire searching operation.

ii. In [4] only one data sequence is examined, whereas we handle many music sequences. If we used the direct approach of [4], the music sequences would be examined in random order (the one that the range query produces), resulting to a scattering effect while accessing disk pages that contain the sequences. No locality in access will be preserved and a buffer cannot be utilised effectively (a phenomenon that in database terminology is called thrashing).

To overcome the latter issue (ii), we do not directly examine each possible match. Instead, we collect information for all possible matches (namely, the starting and ending position of each corresponding subsequence and the ID of the music sequence from which the subsequence originates). Then, we resolve false alarms in a post-processing step, by first grouping the possible matches for each music sequence separately, and by sorting in each group the ranges according to their starting positions. With this method, we try to avoid the random scattering when accessing the music sequences.

To address the former issue (i), we propose the examination of pages in a reverse order when resolving a false alarm. For instance, assume that we have to resolve a false alarm generated from an MBR that corresponds to subsequences in the range $[l, r]$. For each position $l \leq i \leq r$, there may exist a subsequence of length $|Q|$ that matches the query sequence Q. Even if the range $[l, r]$ is relatively small, the fact that we have to examine subsequences of a very large length $|Q|$, produces the problem that we have to address. A straightforward approach would examine all these subsequences and report those that produce a match. A trivial optimization is to terminate the examination of each subsequence at the moment that the actual distance becomes greater than the user-defined threshold of similarity (since the examination of the rest of the subsequence cannot reduce the distance). However, this results only into CPU time savings. I/O is not reduced, because all subsequences starting at all position $l \leq i \leq r$ have to be examined. Since each such subsequence is of length $|Q|$, a large number of disk pages have to be fetched.

The reverse-order examination scheme works as follows. When we have to examine a subsequence that starts at position i in its corresponding sequence D_c, we do not fetch the page containing this first element. Instead, we find the page (denoted as R) that contains the last element that has to be examined, that is, the $|Q| + i - 1$ element of D_c. Then, we first examine the partial distance between elements of R and the corresponding elements in Q, which is properly aligned so as if it is examined against the subsequence starting at position i. If the partial distance is larger than the user-defined ϵ, then we do not examine the rest elements of the subsequence. By moving to the next position, i.e., $i + 1$, we can still first examine the partial distance between the corresponding elements in Q and the ones of the subsequence starting at position $i + 1$. Thus, we avoid reading another page, as long as we examine subsequences that contain elements stored in R. In case the partial distance is not greater than ϵ, then we have a partial match. We fetch the page that contains the first position of the currently examined subsequence and we compute the actual distance between Q and the subsequence, until a full match is found or the computed distance exceeds ϵ. Assuming that each disk page can store N elements, we use a buffer than can hold $\lceil |Q|/N \rceil$ pages, so as to avoid re-reading the intermediate pages.[1] Conclusively, in case of a full match, all intermediate pages are read (a fact that cannot be avoided), whereas in other cases a large number of page reads can be avoided, leading to significant savings in I/O time.

Before presenting the algorithm, we note the following. Assume that we must examine the subsequence that starts at position i in sequence D_c. Also assume a numbering of pages in D_c: the first one has ID 0 and the final one $\lfloor |D_c|/N \rfloor$. Then, the page that contains the last element has ID (denoted as rpID) equal to $\lfloor \frac{i+|Q|-1}{N} \rfloor$. The offset of the first element in this last page is denoted as f and is equal to rpID $\times N$. By using these notations, Figure 4 illustrates the alignment of Q when testing a partial match. The elements involved in the calculation of the partial distance are shown in gray color.

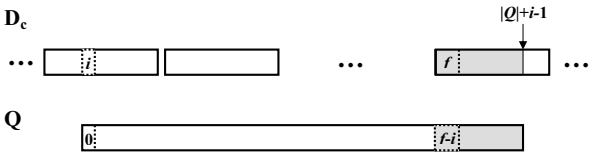

Figure 4. Example of partial matching using the reverse page scheme.

Now we move on to describe the algorithm that resolves false alarms, which is given by procedure RFA (Resolve False Alarms) in Figure 5. RFA operates within a loop that examines all positions $l \leq i \leq r$, where $[l, r]$ is the range that needs to be examined. The decision whether to compute the partial distance or to proceed in a normal distance calculation is determined by the value of variable rMode (stands for reverse mode). Whenever a partial match is found, rMode becomes true (it becomes again false, when a normal match fails). During the calculation of partial or normal distances, elements of D_c sequence have to be examined. The algorithm examines when a new page has to be fetched, since these elements may be stored in several contiguous pages. Notice that fetching is done through a buffer with $|Q|/N$ pages, so as to avoid re-reading of the same pages when it is not necessary.

5 Performance Evaluation

In support of the efficiency of the proposed algorithm, this section presents a number of experiments that have been performed. A concise description of the experimentation platform and data sets is also given followed by a performance analysis based on experimental comparison of the baseline approach, i.e., the ST-Index [4], and the proposed approach, the MS-Index (stands for Music Subsequence match Index).

All algorithms described have been implemented and performed on a personal computer with 3.01GHz Intel Pentium IV processor, 1GByte RAM, operating system MS Windows XP, while the developing package utilised was MS Visual C++ version 6. The performance measure was

[1] For fair comparison, we also provide the same buffer to the method of [4]. Also, we use for it the trivial optimization of early termination of distance calculation.

```
procedure RFA(D_c, l, r, Q, ε, N, wSize)
begin

forceRead = false, rMode = true
rpID = ⌊(i+|Q|−1)/N⌋
f = rpID × N

for (i = l ; i < min{r, |D_i| − |Q|}; i++)
    if (forceRead == true)
        /*a partial match was found earlier in reverse page*/
        fetch page that contains the i-th element of D_c
        forceRead = false
    else if (⌊i/N⌋ == rpID)
        /* rPID will be tested as a normal page*/
        rMode = false
    else if (⌊i/N⌋ > rpID)
        /*a new reverse page must be found*/
        rpID = ⌊(i+|Q|−1)/N⌋
        f = rpID × N
        rMode = true

    if (rMode == false)
        s = 0
        for (j = 0); j < |Q|; j++)
            fetch page containing the (i + j)-th element of D_c
            s += (D_c[i + j] − Q[j])^2
        if (√s ≤ ε)
            output match
        else
            rMode = true
            forceRead = false
    else
        s' = 0
        for (j = 0; j < |Q| − f + i; j++)
            s' += (D_c[f + j] − Q[f − i + j])^2
        if (√s' ≤ ε)
            rMode = false
            forceRead = true
            i − − /*re-examine i-th element for full match*/
end
```

Figure 5. The algorithm that resolves false alarms.

the wall-clock time measured in milliseconds.

The data sets employed for the experiments included solely real music objects. Experiments have been conducted on 300 audio files including more than 13 hours of music. The audio files originated from CD-Audio that were ripped in wav format. These music objects include classical works, modern international pop, rock, instrumental music as well as different types of Greek music. Queries were made by retaining 1-10 seconds of the audio files included in the database. Henceforth, for purposes of more clear representation, results illustrate the relative execution times between the MS-index and ST-index.

In the first experiment we consider the retrieval time of both approaches with relation to the window size of the DFT procedure. The DFT window is a sliding window placed on every offset of the original music sequence, while for each such placement the first three DFT coefficients are kept. The vector of DFT coefficients is then fed to the MBR creation unit. The results on (relative) execution time are illustrated in Figure 6.

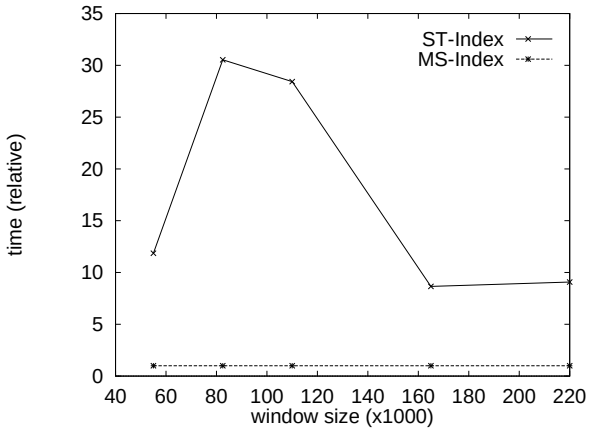

Figure 6. Relative retrieval time for different DFT window sizes

Considering the absolute times, we would notice that as expected, the execution time for both algorithms decreases with increasing window sizes. This is due to two factors. First of all, by increasing the window size, we achieve a reduction in the size of each DFT coefficient sequence, and for mean size of the MBRs, we obtain less MBRs. Thus, the index performance is ameliorated, reducing the CPU cost. In addition, there are much fewer candidates that the false alarm resolution needs to process, thus reducing the I/O cost.

However, focusing on the relative execution times (Figure 6), we notice that the results for ST-Index are tunable with respect to the window size. Within the limits tested the worse performance was achieved for $w = 80,000$, while for standard value used in all other comparative experiments ($w = 220,000$) its performance is very close to the best received. Nevertheless, MS-Index clearly outperforms the ST-Index by a factor of more than 8, while in some cases the difference is even more than 30.

In our next experiment we study the retrieval time of both approaches with relation to the different values of the *p* parameter. The *p* parameter corresponds to the estimated size of the range query. The results are depicted in Figure 7.

We have to notice that both methods perform better for increased *p* and large query size. This is mainly due to the fact that for higher *p* less MBRs are created, that is we obtain lengthier sub-trails within each MBR. Accordingly, the MBRs are better located in the feature space with less overlapping. This leads more efficient range query results and significantly fewer candidates to be resolved in each

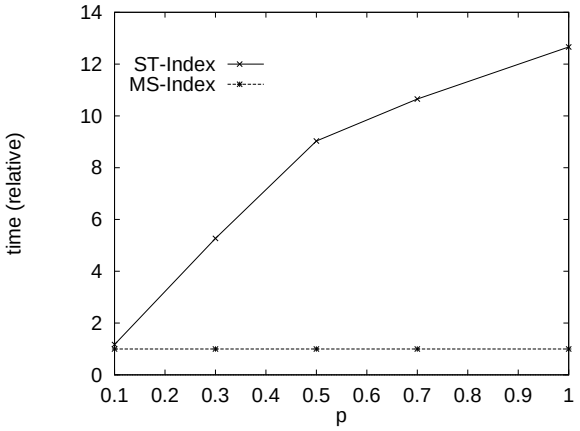

Figure 7. Relative retrieval time for different values of the p parameter

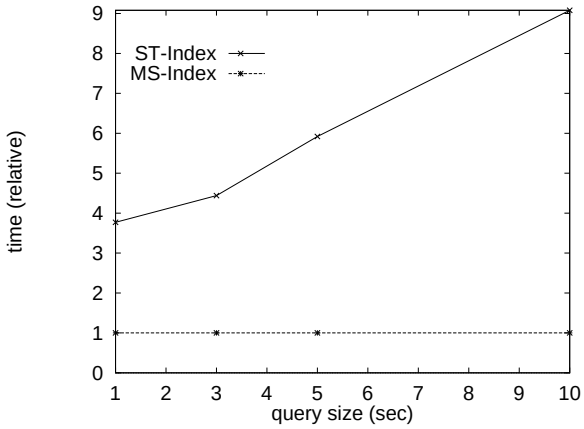

Figure 8. Relative retrieval time for different query sizes.

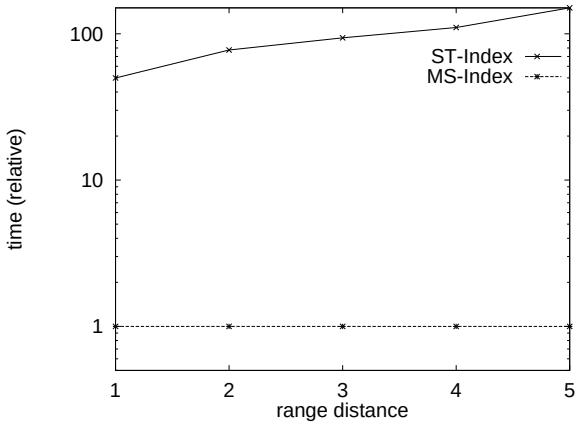

Figure 9. Relative retrieval time for different query range sizes.

false alarm resolution, thus significantly reducing the I/O cost. However, considering the relative performance of both methods, we see that MS-Index proves to be significantly better, especially for greater p.

Figure 8 presents the third experiment, were we study the relative retrieval time for different query sizes. The query size is measured in seconds and since the window size of the DFT cannot be greater than the size of the query sequence, we have also varied the window size proportionally. Should the DFT window size had been left unaffected, then the result would be misleading as its small size would produce a large number of sub-queries, that would increase the time of larger queries. As previously, the performance of the MS-Index is at least 4 times better than the ST-Index, while for greater queries becomes almost 9 times faster. The relative superiority of the MS-Index is due to the fact the larger query size uses larger DFT window and the reverse order schema becomes more effective by pruning more intermediate pages.

In the final experiment, we examine the relative retrieval time for different query range sizes (user-defined ϵ parameter). The results are given in Figure 9 (notice that the vertical axis is in logarithmic scale). In this case, as range increases, the number of hits returned requiring false alarm resolution increases. Accordingly, the number of false alarms increases. The ST-Index, by resolving each hit directly, does not take full advantage of the buffering scheme available. This is due to the absence of locality within files, on which the buffer relies. On the other hand, the MS-Index resolves hits for each file separately by accumulating them. Thus, the locality within the buffer is preserved.

6 Conclusions

We have presented a feature extraction method for acoustical music sequences. The extracted features are grouped by Minimum Bounding Rectangles (MBRs) and indexed by means of a spatial access method. Given a range query and some results, we have presented a false alarm resolution method that utilises a reverse order schema while calculating the Euclidean distance of the query and results, in order to avoid costly calculations. Comparative evaluation to an already existing algorithm showed significantly reduction in execution times.

Further work could be oriented towards the fine-tuning of the various parameters used by the algorithm, in order to achieve maximum efficiency. Additionally, further refinement could be applied to the feature extraction method as well as the sub-sequence matching technique, by applying the methods proposed in [12]. More advanced signal meth-

ods could lead to more musically meaningful representation of the music sequence, increasing thus, the performance at the level of the results returned from the SAM.

Currently, the proposed method supports only range searches, thus an interesting future work is the inclusion of nearest-neighbor searching. Finally, the method could benefit from a ranking system that would classify the true results of a k-NN search according to their similarity to the query, thus allowing search for similarity in different versions of the same audio sequence.

References

[1] N. Beckmann, H.P. Kriegel and B. Seeger: "The R*-tree: an Efficient and Robust Method for Points and Rectangles", *Proceedings of the ACM International Conference on Management of Data (SIGMOD'90)*, pp.322-331, Atlantic City, NJ, 1990.

[2] J.C. C. Chen and A.L.P. Chen: "Query by Rhythm An Approach for Song retrieval in Music Databases", *Workshop Research Issues in Data Engineering*, pp.139-146, 1998.

[3] D. Dervos, P. Linardis and Y. Manolopoulos: "S-index: a Hybrid Structure for Text Retrieval", *Proceedings ADBIS*, pp.204-209, 1997.

[4] C. Faloutsos, M. Ranganathan and Y. Manolopoulos: "Fast subsequence matching in time-series databases", *Proceedings of the ACM SIGMOD international conference on Management of data*, pp.419-429, 1994.

[5] C. Faloutsos: "*Searching Multimedia Databases by Content*", Kluwer Academic Publishers, 1996.

[6] A. Ghias, J. Logan, D. Chamberlin and B. C. Smith: "Query by Humming: Musical Information Retrieval in an Audio Database", *ACM Multimedia*, pp.231-236, 1995.

[7] J.L. Hsu, C.C. Liu and A.L.P. Chen: "Discovering Non-Trivial Repeating Patterns in Music Data", *IEEE Transactions on Multimedia*, Vol.3, No.3, pp.311-325, 2001.

[8] J. Hsu, C. Liu and A.L.P. Chen: "Efficient Repeating Pattern Finding in Music Databases", *Proceedings ACM CIKM*, 1998.

[9] B. Kostek and A. Wieczorkowska: "Parametric Representation of Musical Sounds", *Archive of Acoustics*, pp.3-26, 1997.

[10] Y.L. Lo and S.J. Chen: "Multi-Featured Indexing for Music data", *IEEE TICDCSW*, 2003.

[11] C.C. Liu, J.L. Hsu and A.L.P. Chen: "An Apprximate string MAtching Algorithm for Content-Based Music Data Retrieval", *IEEE Multimedia Computing and Systems*, pp.451-456, 1999.

[12] Y. Moon, K. Whang and W. Han: "A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows" *In Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pp. 382-393, 2002.

[13] C. Papaodysseus, G. Roussopoulos, D. Fragoulis, Th. Panagopoulos, C. Alexiou: "A new approach to the automatic recognition of musical recordings", *Jounal of Acoustical Engineering Society*, Vol. 49, No 1/2, pp.23-35, 2001.

[14] M. Paraskevas and J. Mourjopoulos: "A Statistical Study of the Variability and Features of Audio Signals", *Audio Engineering Society*, 1996.

[15] J. Pickens: "Harmonic Modeling for Polyphonic Music Retrieval", PhD Thesis, University of Massachusetts at Amherst, 2004.

[16] J. Reiss, J.-J. Aucouturier and M. Sandler: "Efficient multidimensional searching routines for music information retrieval", *2nd ISMIR*, pp.163-171, 2001.

[17] V. S. Subrahmanian: "Multimedia Database systems", Morgan Kaufmann Publishers, San Francisco, 1998.

[18] A. Wieczorkowska: "Musical Sound Classification based on Wavelet Analysis", *Fundamenta Informaticae*, Vol. 47, No. 1/2, pp.175-188, 2001.

[19] A. Wieczorkowska and Z. Ras: "Audio Content Description in Sound Databases", *Web Intelligence: Research and Development*, pp.175-183, 2001.

[20] J.-Y. Won,J.-H. Lee, K. Ku, J. Park and Y.-S. Kim: "A Content-Based Music Retrieval System Using Representative Melody Index from Music Databases", *To appear in ADBIS 2004*, 2004.

[21] W. Lee and A. L. P. Chen: "Efficient Multi-Feature Index Structures for Music Information Retrieval", *SPIE*, pp. 177-188, 2000.

[22] C. Yang: "Efficient Acoustic Index for Music Retrieval with Various Degrees of Similarity". *ACM Multimedia*, pp.584-591, 2002.