Building a Platform for Performance Study of Various Music Information Retrieval Approaches

Jia-Lien Hsu and Arbee L.P. Chen

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.

alpchen@cs.nthu.edu.tw

ABSTRACT

In this paper, we describe the Ultima project which aims to construct a platform for evaluating various approaches of music information retrieval. Three approaches with the corresponding tree-based, list-based, and (*n*-gram+tree)-based index structures are implemented. A series of experiments has been carried out. With the support of the experiment results, we compare the performance of index construction and query processing of the three approaches and give a summary for efficient content-based music information retrieval.

1. Introduction

With the growth of music objects available, it is getting more attention on the research of constructing music information retrieval systems. To provide an efficient and effective content-based retrieval of music objects, various approaches have been proposed in which the music representations, index structures, query processing methods, and similarity measurement are key issues.

Regarding the issue of music representation, several approaches are introduced to model various features of music content, such as pitch, rhythm, interval, chord, and contour. To efficiently resolving user queries, different kinds of techniques are proposed, including string matching methods, dynamic programming methods, *n*-gram indexing methods, and list-based and tree-based indexing structures with the corresponding traversal procedures. Most researchers present their solutions to the key issues separately. However, the research work focusing on the quantitative and qualitative comparison of various techniques used in music information retrieval is still limited.

On the contrary, in the traditional information retrieval, the problems and techniques involved in the evaluation of retrieval systems and procedures have been investigated. The most common evaluation criteria have also been identified, such as precision and recall, response time, user effort, form of presentation, and collection coverage [18].

Due to the multi-faceted properties of music, there exist intrinsic difficulties for content-based music information retrieval (MIR). The framework of a formal MIR evaluation mechanism becomes necessary. The point is emphasized in [7], which states "a formalized set of MIR evaluation standards must become part of the MIR researcher toolkit" and "a set of music test databases of substantial size and varied content must be formed so that all MIR researchers can properly compare and contrast techniques under a variety of scenarios."

From the database point of view, we initiate the project of building a platform for the evaluation of music information retrieval systems. Considering the retrieval efficiency and effectiveness, we focus on the performance study of music representations, indexing and query processing which involve a wide range of techniques used in content-based music information retrieval.

The rest of this paper is organized as follows. In Section 2, we describe our project for evaluating music information retrieval approaches. The issues of system design, data set, query set generation, and efficiency and effectiveness study are also introduced in this section. The three approaches implemented in our platform are described in Section 3. We perform a series of experiments and illustrate the experiment results and performance study in Section 4. Section 5 concludes this paper and points out our future directions.

1.1 Related Work

Selfridge-Field [19] provides a survey of clarifying and resolving conceptual and representational issues in melodic comparison. Research work on MIR systems are introduced as follows. Ghias, *et al.* [10] propose an approach for modeling the content of music objects. A music object is transformed into a string which consists of three kinds of symbols, 'U', 'D', and 'S' which represent a note is higher than, lower than, or the same as its previous note, respectively. The problem of music data retrieval is then transformed into that of approximate string matching.

In [1][6], a system supporting the content-based navigation of music data is presented. A sliding window is applied to cut a music contour into sub-contours. All sub-contours are organized as an index structure for the navigation. Tseng [20] proposes a content-based retrieval model for music collections. The system uses a pitch profile encoding for music objects and an *n*-gram indexing for approximate matching. A framework is also proposed in which the music objects are organized as an *n*-gram structure for efficient searching [22]. Different techniques of local alignment and local common subsequences have also been applied for comparison. Similar techniques of *n*-gram indexing have also been employed in [8][24][25]. Furthermore, Downie and Nelson [8] provide an effectiveness evaluation of an *n*-gram based MIR system by using statistical analysis.

The work [17] focuses on music retrieval from a digital library in which dynamic programming is used to match melodic phrases. The issues of melody transcription and matching parameters are discussed and the trade-off between the matching criteria and retrieval effectiveness is shown. Also using dynamic programming, Lemstrom and Perttu [14] present a bit-parallel algorithm for

efficiently searching melodic excerpts. In the bit-parallel processing, the whole table for dynamic programming need not be created, and thus it leads to a better performance. Clausen, *et al.* [5] design a web-based tool for searching polyphonic music objects. The applied algorithm is a variant of the classic inverted file index for text retrieval. A prototype is implemented and its performance is investigated.

To develop a content-based MIR system, we have implemented a system called *Muse* [3][4][13]. In this system, various methods are applied for content-based music data retrieval. The rhythm, melody, and chords of a music object are treated as music feature strings and a data structure called *1D-List* is developed to efficiently perform approximate string matching [13]. Moreover, we consider music objects and music queries as sequences of chords [4] and *mubol* strings [3]. A tree-based index structure is developed for each approach to provide efficient matching capability. In [3], we propose an approach for retrieving music objects by rhythm. Instead of using only melody [1][4][6][10][13] or rhythm of music data, we consider both pitch and duration information plus the music contour, coded as *music segment*, to represent music objects [2]. Two index structures, called *one-dimensional augmented suffix tree* and *two-dimensional augmented suffix tree*, are proposed to speed up the query processing. By specifying the similarity thresholds, we provide the capability of approximate music information retrieval. When considering more than one feature of music objects for query processing, we propose multi-feature index structures [12]. With the multi-feature index, both exact and approximate search functions on various music features are provided.

2. The Ultima Project

The Ultima project is established with the goal to make a comprehensive and comparative assessment of various MIR approaches. Under the same environment and real data sets, a series of experiments can be performed to evaluate the efficiency and effectiveness of the MIR systems. Issues such as the threshold setting and the identification of most influential factors which dominates the system performance can be explored. Furthermore, heuristics for choosing appropriate representation schemes, indexing structures, and query processing methods when building an MIR system can be provided based on the performance study. The Ultima platform will be continuously maintained and served as the testbed whenever new approaches of content-based music information retrieval are proposed.

2.1 System Design and Implementation

The system is implemented as a web server, which runs on the machine of Intel Pentium III/800 with 1GB RAM on MS Windows 2000 by JDK 1.3. For posing queries at the client end, we provide the ways of humming songs, playing the piano keyword, uploading MIDI files, and using the computer keyboard and mouse. The server end consists of a mediator, four modules, and a data store, as shown in Figure 1. The mediator receives user queries and coordinates with other modules. The music objects and the corresponding information, such as title, composer, and genre, are organized as standard MIDI files and relational tables, respectively. The summarization module aims to resemble and visualize query results. The query generation module aims to generate parameterized user queries for performance evaluation, as discussed in Section 2.3. The implementations of the two modules are not finished yet. The report module aims to monitor and assess the performance of the system, such as the elapsed time of query processing, space of indices, and precision and recall of the retrieved results. The query processing module aims to resolve queries from the client end or the query generation module. The query processing module is designed as a "container" to which each query processing methods can be "plugged-in". Whenever a new method is proposed, it can be easily plugged into the module for performing experiments under the same environment. Currently, three methods are considered, *i.e.*, 1D-List [13], APS [2] and APM [3] which will be further discussed in Section 3.

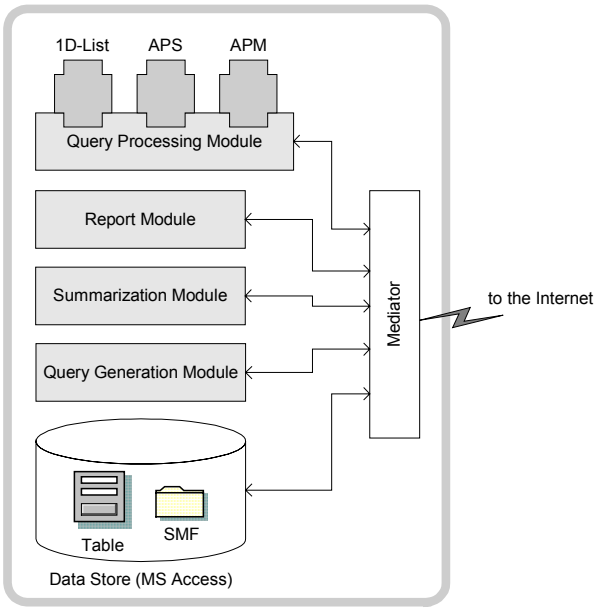


Figure 1: The function blocks of the server in the Ultima project.

2.2 Data Set

The testing data of music objects, from CWEB Technology, Inc., is a collection of 3500 single track and monophonic MIDI files. Most of them are pop music of Chinese and English songs in various genres.

The average object size is 328.05 notes. When coding these objects in the mubol and music segment representations, the average object size is 78.34 (mubol) and 272 (segment), respectively. Based on the statistics of the CWEB dataset, we estimate that one mubol corresponds to 4.19 notes, and one music segment corresponds to 1.21 notes. The *note count* is defined as the number of distinct notes appearing in a music object. According to the MIDI standard, the alphabet size is 128. Therefore, the note count of every melody string is between 1 and 128. For the CWEB data set, the average note count is 13.46. Due to the space limitation, the histograms of the object size and note count of the CWEB data set are skipped.

Moreover, when coding music objects by music segment, the distribution of segment pitch is shown in Figure 2. For the segment duration, most of its value is between 0 and 20 beats without any obvious clustering.

2.3 Query Set Generation

In the traditional information retrieval, there exist standard testing data, queries and the associated answers [9][23].

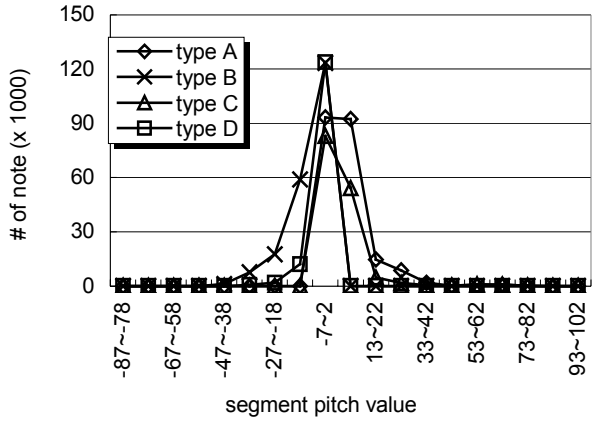


Figure 2: The distribution of segment pitch values of CWEB data set.

Therefore, a fair performance evaluation can be performed. However, there is no such kind of benchmarks dedicated to the MIR systems. In this project, we will also investigate a standard procedure for generating parameterized queries and the associated answers from a data set. With the variety of queries, the performance study will be more accurate.

2.4 Efficiency and Effectiveness Study

We design a series of experiments to evaluate the methods of indexing and query processing. Factors influencing the system performance are identified, such as query length, database size, and query approximation degree. The measurement of performance is based on memory usage, retrieved candidates and elapsed time for efficiency, and precision and recall for effectiveness.

3. Description of the Three Approaches

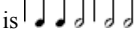
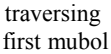
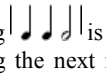
Instead of detailed procedures and algorithms, each approach is illustrated by an example to show the basic idea. The three approaches cover various methods of music representation, indexing, and query processing, as summarized in Table 1. Note that all the approaches support the functionality of exact, partial, and approximate matching. For simplicity, we only show an example of exact query processing.

Table 1: The representations and indexing structures of the three approaches.

Approach	Representation	Index structure
APM	Mubol (rhythm)	(*N*-gram+tree)-based
1D-List	Melody (pitch)	List-based
APS	Music segment (pitch+duration)	Suffix tree-based

3.1 The APM Approach

The rhythm information of music objects is coded as mubol strings. A mubol is a rhythmic pattern of a measure in a music object. A mubol string of a music object is the string of mubols which are determined by each measure of the music objects. For example, as shown in Figure 3(a), R1 is a mubol string of eight measures. The *n*-grams of R1, where $n = 1, 2, 3$, with the associated positions are listed in Figure 3(b). For example, the position "R1: 1,4,7" of the mubol in the first row of Figure 3(a) indicates the mubol appears in the first, forth, and seventh measure of R1. All the prefixes of an *n*-gram can be found in the $(n-1)$-grams, $(n-2)$-grams, …, and 1-grams. To efficiently process queries, the *n*-grams of mubol strings are organized as a tree structure, called L-tree. The tree height h of L-tree is the maximal n, i.e., $h = 3$ in our example. As shown in Figure 3(c), the nodes in level 1 of the tree indicate the first mubols of 1-grams, the nodes in level 2 indicate the second mubols of 2-grams, etc. Note that there are two kinds of links in L-tree, namely, solid link and dotted link. The internal nodes are connected with solid links, while the leaf nodes with the associated information are indicated by dotted links.

The exact query processing is performed by a tree traversal of the L-tree. Suppose the query is  for exact searching. The query will be processed by traversing the L-tree in level-wise manner. When processing the first mubol of the query at level 1 of the L-tree, the node containing  is matched and its children will be reached for processing the next mubol. When processing the second mubol at level 2, those children (only one node in this example) will be compared to the second mubol. Since the node containing  is matched to the second mubol of the query string, the two children of this node will be reached for further processing. Moreover, all the mubols of the query string have been processed and only (R1:2,5) is the answer.

The L-tree is a (*n*-gram+tree)-based index structure. In the approach of *n*-gram indexing, if the length of the query string is larger than n, the false match may happen. For the L-tree of tree height h, if the query length is larger than h, the query will be divided into subqueries and the intermediate answers with respective to each subquery will be merged and confirmed by the join processing.

3.2 The 1D-List Approach

In this approach, music objects are coded as melody strings. For example, there are two music objects M1 and M2 in the database. The melody strings of M1 and M2 are "so-mi-mi-fa-re-re-do-re-mi-fa-so-so-so" and "do-mi-so-so-re-mi-fa-fa-do-re-re-mi", respectively.

To support efficient string matching, melody strings are organized as linked lists, as shown in Figure 4(a). For the notes of the same pitch in the melody strings, they are linked in an increasing order. Each node in the linked lists is of the form (*x*:*y*) which denotes the *y*-th note of the melody string of the *x*-th music object in the database.

When a query Q = "do-re-mi" is specified, the lists involved in Q are retrieved with the two dummy nodes *start* and *end* as shown in Figure 4(b). Then, the exact query processing goes as follows. Let A_x be the first element of node A, and A_y be the second element of node A. For each pair of nodes (A, B) taken from two adjacent linked lists, if $A_x = B_x$ and $A_y+1 = B_y$, we build an exact link from A to B, as shown in Figure 4(c). Also, we build an exact link from *start* to node F of the first list if F has an outgoing link, and from node L of the last list to *end* if L has an incoming link. By traversing the exact links from *start* to *end*, each path indicates a substring appearing in the melody string of the database and will be considered as a result. In our example shown in Figure 4(c), there exists only one path, which is denoted in bold-faced links, *i.e.*, "*start*-(1:7)-(1:8)-(1:9)-*end*".

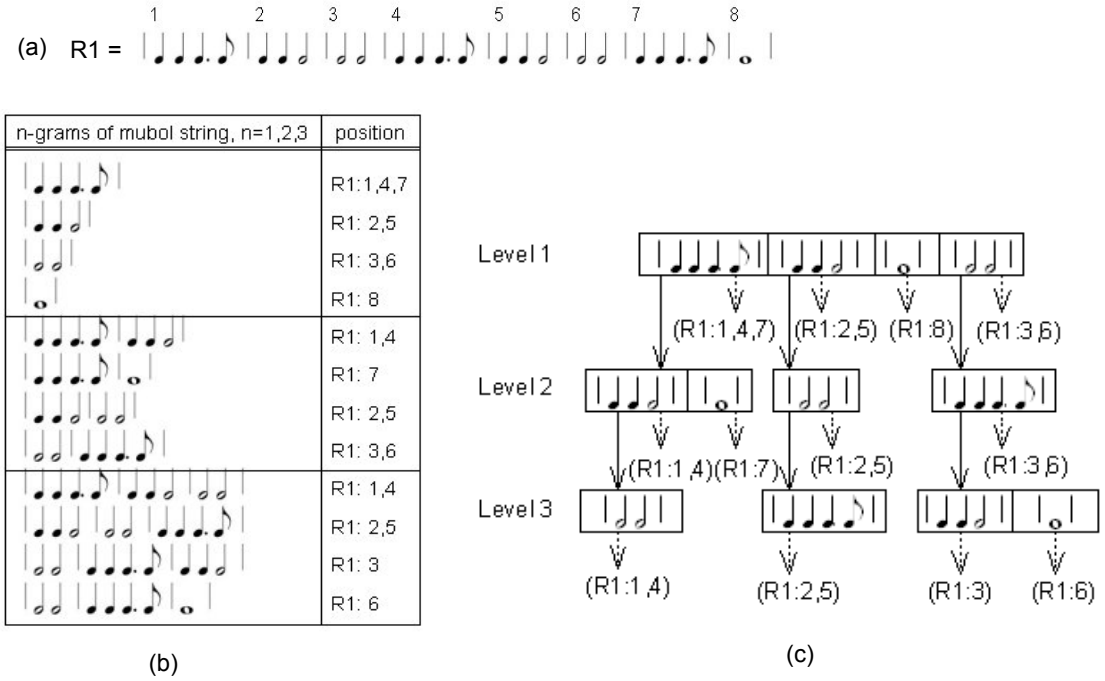


Figure 3: (a) A sample mubol string R1. (b) The table of *n*-grams associated with the corresponding positions. (c) The L-tree of the mubol string R1.

3.3 The APS Approach

For better readability, the representation, indexing, and query processing are separately described as follows.

3.3.1 Representation of Music Objects

Taking into account of music contour with note duration and pitch, the APS approach represents music objects by sequences of music segments. A *music segment* is a triplet which consists of the *segment type* and the associated duration and pitch information. There are four segment types defined to model the music contour, *i.e.*,  (type A),  (type B),  (type C), and  (type D). Define the *segment base* as the horizontal part of a music segment. The beat information of a music segment is represented by the *segment duration* which is the number of beats in the corresponding segment base. The pitch information of a music segment is represented by the *segment pitch* which is the *note number* in the MIDI standard of the corresponding segment base minus the note number of the segment base of the previous segment base. For example, for the piece of music shown in Figure 5, the corresponding representation as a sequence of music segments is shown in Figure 6. The music segment (A, 1, +1) indicates that it is a type A segment with the segment duration and segment pitch being 1 and +1, respectively. When coding by music segments, the first music segment and the last music segment are ignored due to lack of information to assign the segment type. Therefore, the music object of Figure 5 is represented by the sequence of (B,3,-3) (A,1,+1) (D,3,-3) (B,1,-2) (C,1,+2) (C,1,+2) (C,1,+1). In priori to describing the dedicated indexing structures for APS, we introduce a data structure named *suffix tree*. A suffix tree is originally developed for substring matching [11][15].

For example, Figure 7 shows the suffix tree of the string S = ”ABCAB”. Each leaf node (denoted by a box) corresponds to a substring starting at the position indicated in the node in S, and each link is labeled with a symbol α, where $\alpha \in \Sigma \cup \{\$\}$, Σ is the alphabet of S and ‘$’ is a special symbol denoting end-of-string. As a result, all the suffixes, *i.e.*, “ABCAB”, “BCAB”, “CAB”, “AB”, and “B”, are organized in the tree. For a query string, the matching processing is a typical tree traversal. For example, suppose that the query string is “AB”. We follow the leftmost path to the black node, and all leaf nodes descending from the black node are the results, *i.e.*, the first and the forth position.

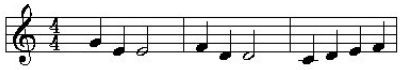


Figure 5: A piece of music.

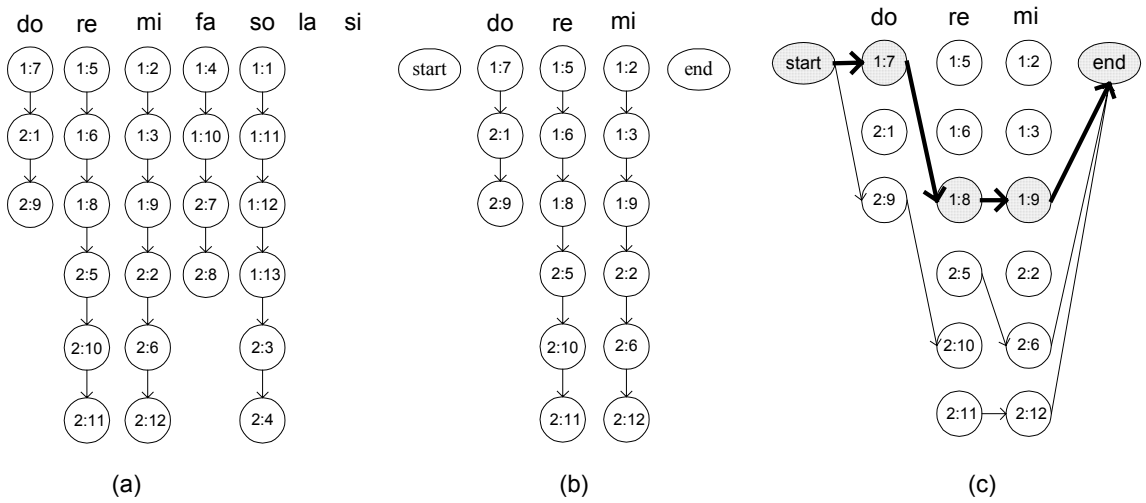


Figure 4: (a) The index structure of M1 and M2 for the 1D-List approach. (b) An example of exact query Q = "do-re-mi". (c) The exact link and result of the query Q.

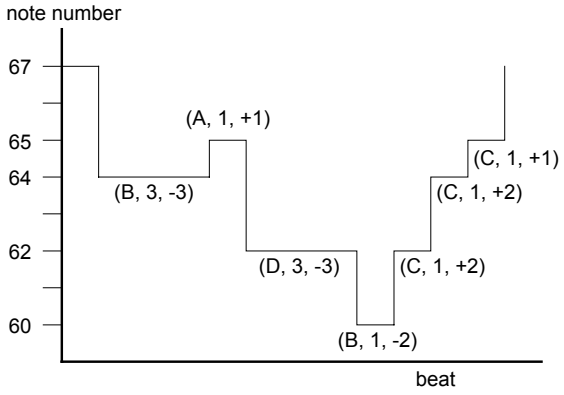


Figure 6: The corresponding sequence of music segments of the music object in Figure 5.

3.3.2 Index Structures for Sequences of Music Segments

In the following, we introduce two index structures for efficiently processing queries of music segments, *i.e.*, the one-dimensional and two-dimensional augmented suffix trees.

A one-dimensional augmented suffix tree (1-D AST, in short) is a suffix tree with the segment duration information being added to the edges. First, a suffix tree based on the sequences of the segment types is constructed. Each edge of the suffix tree refers to a symbol appearing in one or more positions in the sequence. For example, let the sequence of music segments be (A,2,+1) (B,5,-1) (C,1,+1) (A,3,+1) (B,3,-2). Using only the segment types, the suffix tree can be constructed as shown in Figure 7. The bold-faced edge in Figure 7 refers to the 'B' in the second and fifth position. Since the corresponding segment durations are 5 and 3, we attach the range of segment duration <*min*, *max*> = <3, 5> to the edge. This range can be used to filter out some results which cannot be answers during query processing. Figure 8 shows an example of a 1-D AST.

To exploit the filtering effect, the range <*min, max*> should be as compact as possible. For a given population of segment durations, such as {1, 2, 2, 3, 7, 8, 8}, two ranges <1, 3> and <7, 8> are better than one range <1, 8>. Thus, the edge should be split into two edges labeled with <1, 3> and <7, 8>. This method is called *dynamic splitting*. In some cases, however, if it is hard to find compact ranges from a given population, we may apply *static splitting* method by splitting a range into some predefined smaller ranges which can be obtained from the statistics of data set.

The 2-D AST is an extension of the 1-D AST by attaching both segment duration and segment pitch information to the edge.

3.3.3 Query Processing

The query processing for the augmented suffix tree is called the *thresholding-based matching*, which is able to deal with both exact and approximate queries [2]. The approximation degree of the query is specified by means of thresholds. The exact queries can be considered as a special case with the thresholds being set to zero. For ease of illustration, we only show the processing of exact queries in the following.

Based on the 1-D AST in Figure 8(b), given the query Q = (A,1,–) (C,2,–) (A,5,–), we find the music objects whose sequences of music segments contain Q. When processing queries against a 1-D AST, the segment pitch in the queries is not needed and denoted by '–'.

The tree traversal starts from the root node and goes as follows. When processing the first music segment (A,1,–), the edge A<1, 1> is matched such that we reach the node N_1. Then, when processing the second music segment (C,2,–), the edge C<1, 3> is satisfied because the duration of the music segment is covered by the range of the edge. For the last music segment (A,5,–), although the segment type of the two edges from node N_2 is matched, the two edges are filtered out because the duration of the music segment is not covered by any ranges of the edges. Therefore, the processing terminates without any answer. Note that the results derived from this tree traversal are not necessary the answers to the query. Further verification of the results is required.

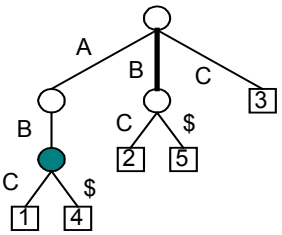


Figure 7: The suffix tree of the string S = "ABCAB".

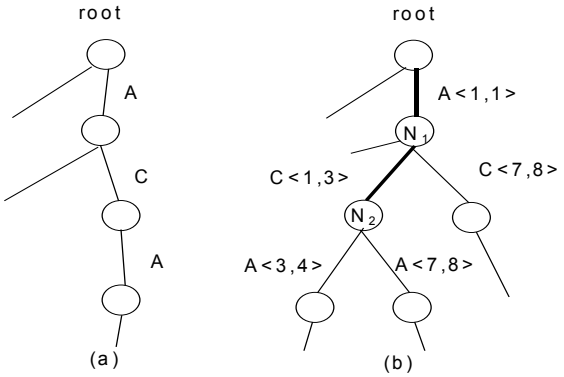


Figure 8: (a) An example of suffix tree. (b) The 1-D augmented suffix tree.

4. The Efficiency Study

In this section, we show the experiment results on the efficiency of the three approaches described in Section 3. For the APS approach, both 1-D AST and 2-D AST are implemented. For comparison, we also construct a suffix tree, denoted as ST, based on the segment types of music segment sequences.

4.1 Index Construction

The elapsed time and memory usage for constructing indices of the three approaches are illustrated as follows. For the APM approach, the tree height of L-tree is set to 6 in our experiments. As shown in Figure 9 and Figure 10, both the elapsed time and the memory usage of 1D-List are less than those of L-tree. This is because the construction of 1D-List is a simple process of transforming the melody strings to the linked lists, and the number of nodes in the 1D-List is linear to the database size.

The suffix tree-like data structures including the augmented suffix trees in the APS approach suffer from the space consumption. It is not reasonable to construct a full and complete augmented suffix tree just for handling the rare cases of extremely long-length queries. On the contrary, an augmented suffix tree with longer tree heights is beneficial to the efficiency of query processing. In our experiments, the tree height of augmented suffix tree is set to 4, 6, 8, 10, and 12. We construct three indices of APS, *i.e.*, ST, 1-D AST, and 2-D AST. As described in Section 3.3, we apply the static splitting method to divide the domain of duration into three ranges and the domain of pitch into two ranges. Obviously, the elapsed time and memory usage of the three indices ST, 1-D AST, and 2-D AST are increasing. We only show the construction of the 2-D AST in Figure 11 and Figure 12, where 'h_4' indicates the tree height of four, 'h_6' indicates tree height of six, and so on. The elapsed time and memory usage in the cases of smaller tree heights are much less than the cases of larger tree heights.

4.2 Exact Query Processing

In the following, we discuss the efficiency of processing exact queries for the APM, 1D-Lst, and APS approaches. Factors of query length, number of objects, and tree height of indices of APS will be investigated.

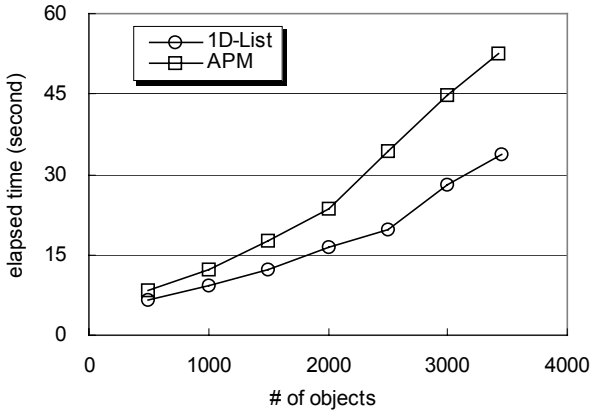


Figure 9: Elapsed time vs. # of objects for index construction of APM (L-tree, h=6) and 1D-List.

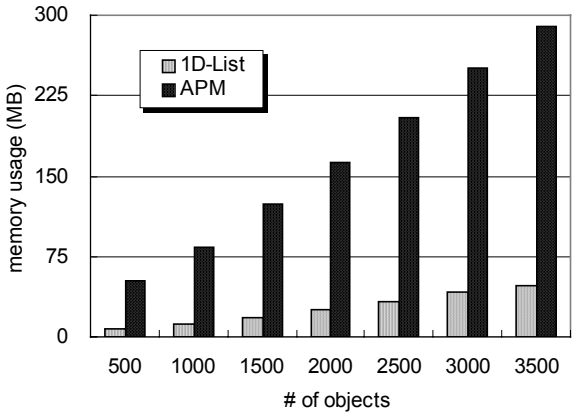


Figure 10: Memory usage vs. # of objects for index construction of APM (L-tree, h=6) and 1D-List.

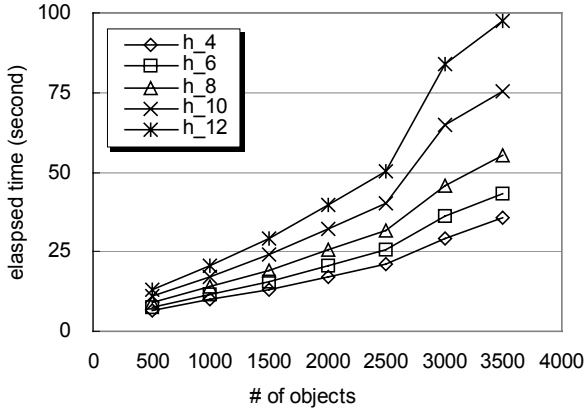


Figure 11: Elapsed time vs. # of objects for index construction of APS (2-D AST).

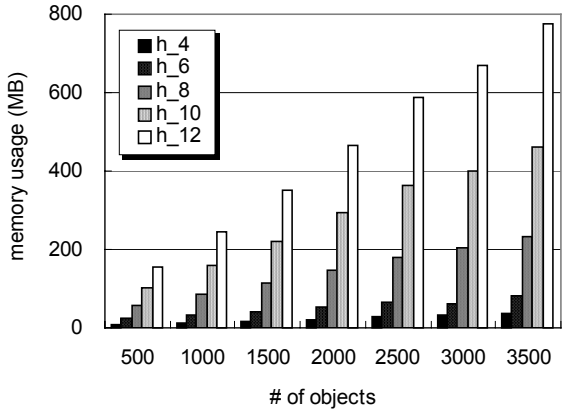


Figure 12: Memory usage vs. # of objects for index construction of APS (2-D AST).

For the APM and 1D-List approaches, the factors of query length and number of objects are explored, as shown in Figure 13, Figure 14, and Figure 15. Figure 13 shows the scalability of two approaches. The query length for 1D-List, denoted by |Qn|, is of twelve notes. Accordingly, the query length for APM, denoted by |Qm|, is of three mubols. Compared to the APM approach, the 1D-List approach scales well as the number of music objects.

Figure 14 and Figure 15 show the elapsed time versus query length of APM and 1D-List, where 'obj_0.5K' indicates five hundred music objects in the experiment, 'obj_1.0K' indicates one thousand objects, and so on. For the APM approach, as shown in Figure 14, the elapsed time decreases rapidly when processing queries of length from one to six. As processing queries of length seven, the elapsed time rises up substantially. In the experiment setting, the tree height of L-tree is six. As in Section 3.1, if the query is of length seven, it will be divided into two subqueries. As a result, two times of the L-tree traversal are required. In addition, the join processing also contributes extra elapsed time. For the queries of length from seven to thirteen, similar behavior can be observed. When processing queries of length from seven to twelve, the elapsed time decreases. As processing the queries of length thirteen, the elapsed time rises up again, and so on. For the 1D-Lsit approach, Figure 15 shows the elapsed time versus the query length. The elapsed time increases slightly for query lengths ranging from 1 to 10, and remains almost the same for longer queries. Since only the lists involved in the query are retrieved for building exact links, the elapsed time is linear to the query length.

The elapsed time consists of the time for building links and traversing links. When dealing with shorter queries, the number of lists to be processed is small and the elapsed time increases slightly. When dealing with longer queries, although the number of lists to be processed increases, the number of answers to the query dramatically reduces such that the elapsed time remains almost the same. In our experiment, the number of answers is less than 2 for the query of lengths ranging from 16 to 64.

For APS, factors of query length, number of objects, and tree height of the three indices are explored as follows.

Figure 16 shows the scalability of APS with 1-D AST and 2-D AST of tree height of eight. The APS with ST is not included because of a much larger elapsed time under the same condition. Compared to the 1-D AST, the 2-D AST performs well as the number of objects increases.

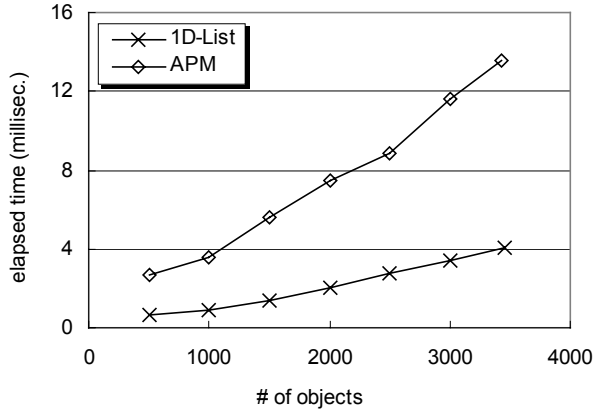


Figure 13: Elapsed time vs. # of objects for query processing of APM (|Qm|=3, L-tree, h=6) and 1D-List (|Qn|=12).

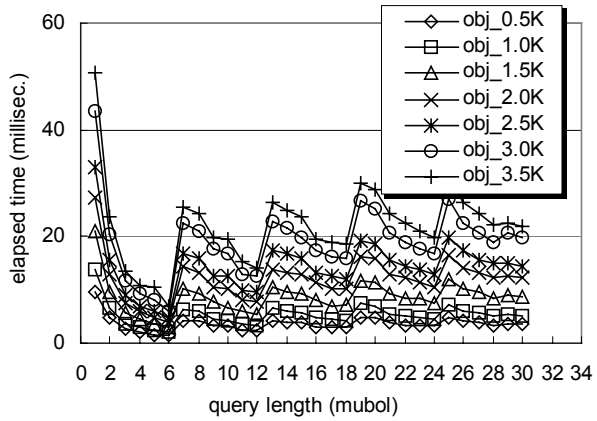


Figure 14: Elapsed time vs. query length for query processing of APM (L-tree, h=6).

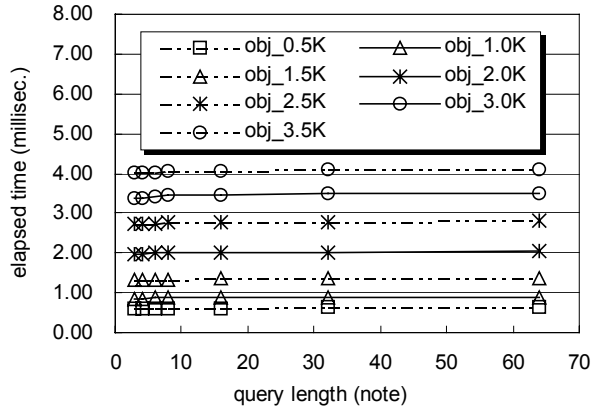


Figure 15: Elapsed time vs. query length for query processing of 1D-List.

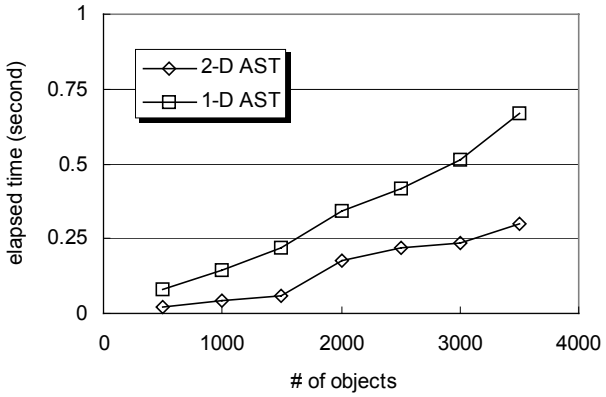


Figure 16: Elapsed time vs. # of objects for query processing of APS (1-D AST and 2-D AST, |Qs|=8, h=8).

Figure 17, Figure 18, and Figure 19 show the elapsed time versus query length for ST, 1-D AST, and 2-D AST, respectively. The curves in Figure 19 have a similar trend to the curves in Figure 14. However, for shorter queries ranging from one to eight music segments, such kind of trend is not obvious in APS. Two reasons are given as follows. In APM, leaf nodes are regarded as results, while leaf nodes of APS are just candidates for further confirmation. In addition, the number of leaf nodes retrieved in APS is much more than the one in APM. For example, after the tree traversal, there are four leaf nodes for four-mubol queries in APM, while there are 16968, 5429, 105 nodes for four-segment queries in APS with the index of ST, 1-D AST, and 2-D AST of tree height twelve, respectively. Post processing of a large number of candidates results in extra computation which smoothes the curves.

The total elapsed time of query processing in APS consists of three parts, *i.e.*, tree traversal, joining processing (if the query length is longer than the tree height), and post processing (for similarity computation). Among the three parts, the post processing consumes most of the elapsed time. For example, with the 2-D AST of tree height ten, the total elapsed time of processing a ten-segment query is 811 milliseconds, where 10 milliseconds for tree traversal and 801 milliseconds for computing similarity. When processing queries whose length is longer than the tree height, the query will be divided into subqueries. The number of candidates will be reduced after the joining processing. However, our database of 3500 music objects is only of moderate size. No matter what the tree height is, the number of candidates does not change much. Therefore, the difference of the performance with various tree heights is not obvious in our experiments, as shown in Figure 17, Figure 18, and Figure 19. We believe that, when dealing with much more music objects in databases, the influence of tree height will be revealed.

For comparison, we show the elapsed time for different indices in Figure 20. The performance gain of 2-D AST is obvious because of substantial edge pruning and candidate reduction.

In the following, we show the filtering effect of APS by applying 1-D AST and 2-D AST. The number of candidates is the number of leaf nodes retrieved after tree traversal. The filtering effect is measured by the *candidate reduction rate* (*CRR*), which is defined as the ratio of the number of reduced candidates using 1-D AST or 2-D AST to the number of candidates using ST.

$$CRR = \frac{N_{ST} - N_{AST}}{N_{ST}}$$

where N_{ST} denotes the number of candidates by applying ST and N_{AST} denotes the one by 1-D AST or 2-D AST.

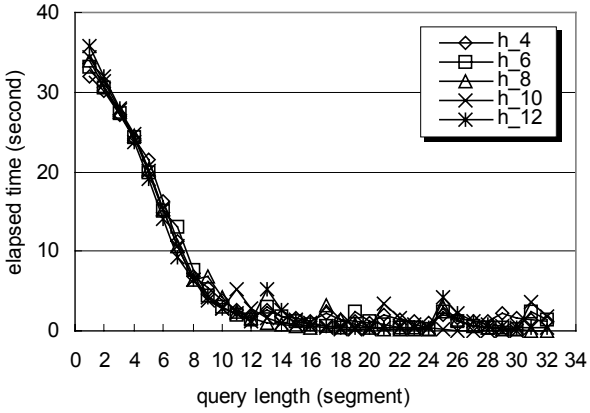


Figure 17: Elapsed time vs. query length for query processing of APS (ST, 3.5K objects).

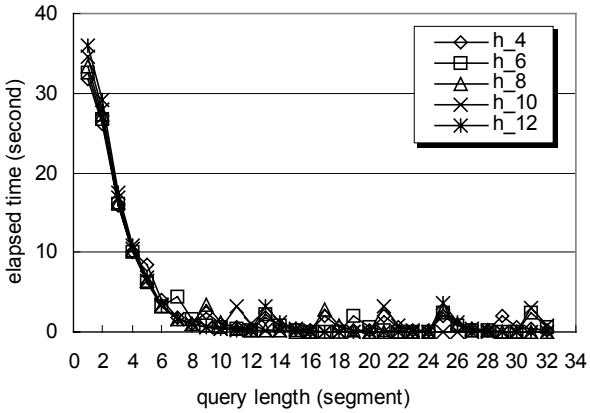


Figure 18: Elapsed time vs. query length for query processing of APS (1-D AST, 3.5K objects).

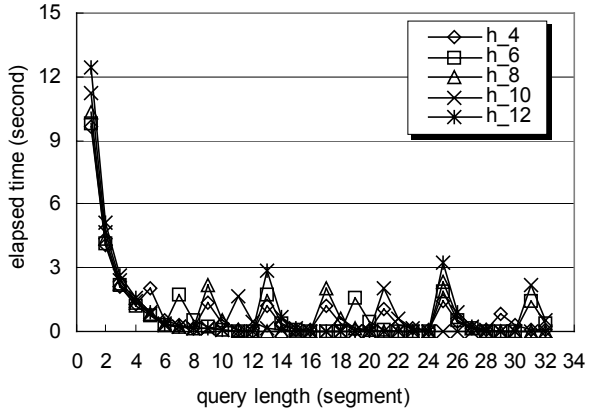


Figure 19: Elapsed time vs. query length for query processing of APS (2-D AST, 3.5K objects).

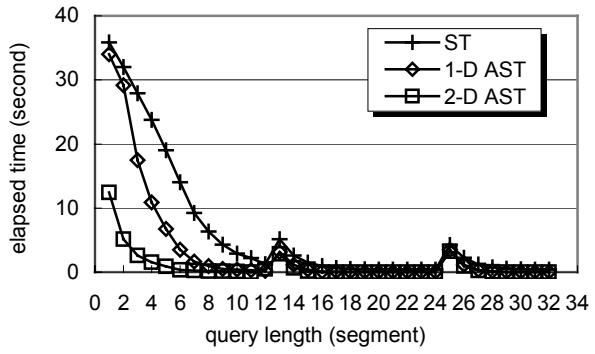


Figure 20: Elapsed time vs. query length for comparison of query processing of APS using various indices (h=12, 3.5K objects).

Higher reduction rates suggest better filtering effects. As shown in Figure 21, there are two kinds of curves with respect to the corresponding y-axis. The 'ST', '1-D AST', and '2-D AST' indicate the number of candidates applying the corresponding indices. The 'R1D' and 'R2D' indicate the CRR of the corresponding indices.

For the 1-D AST, the CRR increases as the query length is less than 14, while the ratio decreases as the query length ranges from 15 to 32. For the 2-D AST, since there are much fewer candidates, the CRR for the query lengths ranging from 1 to 24 is at least 80%. For the longer queries, the CRR is decreased to 67%.

For shorter queries, the APS approaches with 1-D AST and 2-D AST get benefits through attaching the beat and pitch information. However, for longer queries, all the methods have fewer candidates such that the filtering effect decreases slightly. For example, as the query lengths range from 24 to 32, the number of candidates for ST, 1-D AST, and 2-D AST is 3, 1, and 1, respectively. In general, the filtering effect of 2-D AST is better than that of 1-D AST. Moreover, a significant reduction of the candidates can be achieved using our approaches as the query length is less than 14.

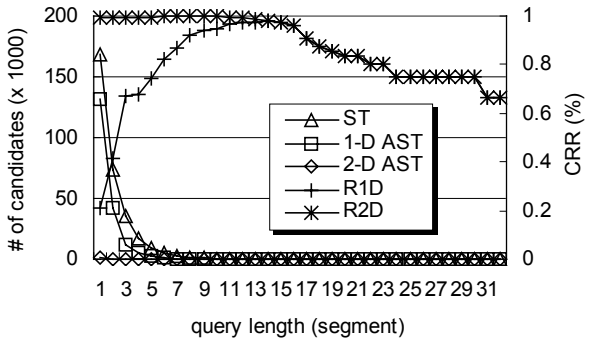


Figure 21: Reduction rate vs. query length for comparison of query processing of APS using various indices (h=12, 3.5K objects).

4.3 Summary of the Experiment Results

Following the comprehensive illustrations of the performance with respect to each approach, we summarize the experiment results for a comparison in Table 2. Four sets of query lengths for query processing are selected, *i.e.*, 1, 2, 3, and 4 mubols for APM, 4, 7, 10, and 12 notes for 1D-List, and 4, 8, 12, and 14 music segments for APS.

For reference, we also implement the string matching methods, namely, STR_MAT_n, and STR_MAT_ms. STR_MAT_n is a standard string matching method using the `indexOF` function in Java, which can be used to compare melody strings. On the other hand, STR_MAT_ms is for comparing sequences of music segments, which match segment types, followed by a checking for segment duration and segment pitch.

We summarize the experiment results as follows.

First, the 1D-List approach is superior in terms of indexing and query processing. However, the melody string of 1D-List approach is coded as the string of pitch values (*i.e.*, the note number in MIDI standard). If the MIR system is designed for end users and the query approximation is one of major concerns, 1D-List may not result in good effectiveness. If it is the case of exact searching from the bibliographic catalog, the 1D-List approach is suggested.

Second, the APM outperforms the APS family. Two reasons are given as follows. The APS family needs an extra cost for post processing. In addition, the average number of branches of a tree node in L-tree is much more than that of AST. It results in fewer candidates of APM. Therefore, the elapsed time of APS family is more than that of APM.

Third, constructing indices for the APS family is not always beneficial to query processing, especially when the query length is less than four music segments. For longer query lengths, the performance of 2-D AST is impressive, as shown in Figure 20 and Table 2. In addition, the performance difference between the 2-D AST with various tree heights is limited, as shown in Figure 17, Figure 18, and Figure 19. Therefore, for the APS family, we suggest using the 2-D AST of smaller tree heights. This is because the index size of 2-D AST substantially reduces when the tree height is smaller. For example, as shown in Figure 12, the index size of 2-D AST of tree height 12 is 774.46 MB, while that of tree height 10 is 461.57 MB.

5. Conclusion

In this paper, we describe the Ultima project which aims at building a platform for evaluating the performance of various approaches for music information retrieval. The issues of system design, query set generation, and performance study are discussed. The list-based, tree-based, (*n*-gram+tree)-based approaches are considered. Concerning the efficiency study, a series of experiments are conducted. The factors of database size, query length, tree height are investigated. We also provide a comparative study and summarization of the three approaches.

Future work include a performance evaluation of retrieval effectiveness among these approaches. Also, various input methods, the summarization module, and the query generation module will be implemented. The dynamic programming-based approaches, which are not covered in this project yet, will be considered in the next stage. While more and more polyphonic music retrieval methods are proposed, we also plan to extend our project to build a database of polyphonic music objects for evaluating these methods.

Table 2: The comparison of various approaches.

Approach (|DB| = 3500)	Index		Exact query processing[1][2] (millisec.)				
	Size (MB)	Time (sec.)	|Qm| = 1 mubol |Qs| = 4 notes |Qn| = 4 segments	|Qm| = 2 |Qs| = 7 |Qn| = 8	|Qm| = 3 |Qs| = 10 |Qn| = 12	|Qm| = 4 |Qs| = 12 |Qn| = 14	In average
APM (L-tree, h=6)	289.0	52.5	50.6	23.8	13.6	10.1	24.5
1D-List	48.3	33.7	4.0	4.0	4.1	4.0	4.0
STR_MAT_n	N/A	N/A	861.0	852.0	852.0	851.0	854.0
APS (ST, h=12)	48.3	39.9	23767.0	9239.0	2899.0	1271.0	9294.0
APS (1-D AST, h=12)	290.7	54.0	10882.0	1630.0	416.0	195.0	3280.1
APS (2-D AST, h=12)	774.5	90.0	1570.0	244.0	96.0	9.0	479.8
STR_MAT_ms	N/A	N/A	2974.0	2814.0	2794.0	2814.0	2849.0

Note:
[1] Qn, Qm, Qs indicate that queries are coded as melody strings for the 1D-List approach, mubol strings for the APM approach, and sequences of music segments for the APS approach, respectively.
[2] |Qn|, |Qm|, and |Qs| indicate the length of queries in note, mubol, and music segment, respectively.

Acknowledgment

We would like to thank the CWEB Technology, Inc., for sharing us the data set used in our experiments.

References:

[1] Blackburn, S. & DeRoure, D. (1998). A tool for content-based navigation of music. In Proc. of ACM Multimedia.

[2] Chen, A. L. P., Chang, M., Chen, J., Hsu, J. L., Hsu, C. H. & Hua, S. Y. S. (2000). Query by music segments: An efficient approach for song retrieval. In Proc. of IEEE Intl. Conf. on Multimedia and Expo (ICME). New York.

[3] Chen, J. C. C. & Chen, A. L. P. (1998). Query by rhythm: An approach for song retrieval in music databases. In Proc. of the 8th Intl. Workshop on Research Issues in Data Engineering, (pp. 139-146).

[4] Chou, T. C., Chen, A. L. P., & Liu, C. C. (1996). Music databases: Indexing techniques and implementation. In Proc. of IEEE Intl. Workshop on Multimedia Data Base Management System.

[5] Clausen, M., Engelbrecht, R., Mayer, D. & Smith, J. (2000). PROMS: A web-based tool for searching in polyphonic music.

[6] DeRoure, D. & Blackburn, S. (2000). Content-based navigation of music using melodic pitch contours. Multimedia Systems, 8(3), Springer. (pp. 190-200).

[7] Downie, S. (2000). Thinking about formal MIR system evaluation: Some prompting thoughts. Available on http://www.lis.uiuc.edu/~jdownie/mir_papers/downie_mir_eval.html.

[8] Downie, S. & Nelson, M. (2000). Evaluation of a simple and effective music information retrieval method. In Proc. of ACM SIGIR, (pp. 73-80).

[9] Frakes, W. B. & Baeza-Yates, R. (1992). Information retrieval: Data structures and algorithms, Prentice-Hall.

[10] Ghias, A., Logan, H., Chamberlin, D., & Smith, B. C. (1995). Query by humming: Musical information retrieval in an audio database. In Proc. of ACM Multimedia, (pp. 231-236).

[11] Gusfield, D. (1997). Algorithms on strings, trees, and sequences. Cambridge University Press.

[12] Lee, W. & Chen, A. L. P. (2000). Efficient multi-feature index structures for music data retrieval. In Proc. of SPIE Conference on Storage and Retrieval for Image and Video Databases.

[13] Liu, C. C., Hsu, J. L., & Chen, A. L. P. (1999). An approximate string matching algorithm for content-based music data retrieval. In Proc. of Intl. Conference on Multimedia Computing and Systems (ICMCS'99).

[14] Lemstrom, K. & Perttu, S. (2000). SEMEX: An efficient music retrieval prototype. In Proc. of Intl. Symposium on Music Information Retrieval.

[15] McCreight, E. M. (1976). A space economical suffix tree construction algorithm. Journal of Assoc. Comput. Mach., 23, 262-272.

[16] MIDI Manufactures Association (MMA), MIDI 1.0 Specification, http://www.midi.org/.

[17] McNab, R. J., Smith, L. S., Witten, I. H., & Henderson, C. L. (2000). Tune retrieval in the multimedia library. Multimedia Tools and Applications, 10(2/3), Kluwer Academic Publishers.

[18] Salton, G. & McGill, M. (1983). Introduction to modern information retrieval. MaGraw-Hill Book Company.

[19] Selfridge-Field, E. (1998). Conceptual and representational issues in melodic comparison. In Hewlett, W. B. & Selfridge-Field E. (Ed.), Melodic similarity: Concepts, procedures, and applications (Computing in Musicology: 11), The MIT Press.

[20] Tseng, Y. H. (1999). Content-based retrieval for music collections. In Proc. of ACM SIGIR.

[21] Uitdenbogerd, A. & Zobel, J. (1998). Manipulation of music for melody matching. In Proc. of the 6th ACM Intl. Multimedia Conference, (pp. 235-240).

[22] Uitdenbogerd, A. & Zobel, J. (1999). Melodic matching techniques for large music databases. In Proc. of the 7th ACM Intl. Multimedia Conference, (pp. 57-66).

[23] Witten, I. H., Moffat, A., & Bell, T. C. (1994). Managing gigabytes: compressing and indexing documents and images, International Thomson Publishing company.

[24] Yanase, T. & Takasu, A. (1999). Phrase based feature extraction for musical information retrieval. In Proc. of IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing.

[25] Yip, C. L. & Kao, B. (2000). A study on *n*-gram indexing of musical features. In Proc. of IEEE ICME.