Economical Structure for Multi-feature Music Indexing[1]

Yu-Lung Lo, and Chun-Hsiung Wang

***Abstract*—The management of large collections of music data in a multimedia database has received much attention in the past few years. In the most of current works, the researchers extract the features, such as melodies, rhythms and chords, from the music data and develop indices that will help to retrieve the relevant music quickly. Several reports have pointed out that these features of music can be transformed and represented in the forms of music feature strings or numeric values such that the indices can be created for music retrievals. However, there is only a small number of existing approaches introduced multi-feature index structures for music queries while most of others are for developing single feature indices. The existing music multi-feature index structures are memory consuming and lack of scalability. In this paper, we will propose an improved version of index structure which is a memory saving approach for multi-feature music indexing. Our experimental results show that the new approach outperforms existing multi-feature index schemes.**

***Index Terms* — multimedia database, music database, multi-feature index, content-based retrieval.**

I. Introduction

As the explosive growth of the multimedia applications, there is more and more non-alphabet data needed being processed now. These data, unlike the conventional numeric or character types of data, includes images, voices, films, documents and so on. For manipulating these new data types, most of proposed papers are contented-based retrieval to process voices, films, and documents [4][5][6][7][13]. Recently, as the rapid progress in digital representations of music data, how to efficiently manage music data is getting more attentions. There are more and more investigations increasingly attractive in retrieving the music collections such as the Query by Rhythm by Chen, et al. [2], Query by Music Segments by Chen, et al. [3], Multi-Feature Index Structures by Lee, et al. [12], Non-Trivial Repeating Pattern Discovering by Liu, et al. [14], Approximate String Matching Algorithm by Liu, et al. [15], Key Melody Extraction and N-note Indexing by Tseng [20], Melodic Matching Techniques by Uitdenbogerd, et al. [21], Numeric Indexing for Music Data by Lo, et al. [16][17] and more in [1][9][13][16] [18].

In the researches of music content-based retrieval, many approaches extract the features, such as key melodies, rhythms, and chords, from the music objects and develop indices that will help to retrieve the relevant music efficiently [11][14][18]. Several reports have also pointed out that these features of music can be transformed and represented in the forms of music feature strings [2][3][9][12][13][18][20] or numeric values [16][17] such that the indices can be created for music retrievals. We also can combine these different features to support various types of queries. However, there is only a small number of existing approaches introduced multi-feature index structures for music retrievals while most of researches are for developing single feature indices. The existing music multi-feature string index structures are memory consuming and lack of scalability, whereas the numeric indexing approaches for music data are inflexible for varied of query lengths and difficult to support for fault tolerance searching. To address the drawbacks of current multi-feature indices for music data retrieval, in this paper, we will propose a space saving multi-feature index structure for music query searching. Our study is also shown that the proposed index structure is more economic in memory need than existing multi-feature indexing approach though it is without query restriction.

The remaining of this paper is organized as follows: in section 2, we specify the string indexing and numeric indexing for music data retrievals. After that, the existing music multi-feature indexing schemes are discussed in section 3. The section 4 introduces our proposed new index structure for music data, and then we show our experimental results in section 5. Finally, a conclusion is given in the last section.

II. Indexing For Music Data Retrieval

Currently, there are two structures of string indexing and numeric indexing for music data retrieval. We specify these two indexing schemes in this section.

A. String Indexing for Music Data

A suffix tree is a tree-like index structure representing all suffixes of a string and provided solutions for string matching problems [19][22][23]. It consists of following characteristics:

1) A suffix tree, constructed from a string with length of *m* symbols, consists of *m* leaf nodes. These leaf nodes can be numbered from 1 to *m*.
2) Any two branches from a non-leaf node should be labeled with different symbols.
3) The number of each leaf node points out the start position of the sub-string which consists of the symbols labeled from the root to this leaf node of the tree.

For example, if there is a string - “ababc”, the suffix tree constructed from this string can be shown in Fig. 1. Recently,

[1] This work was supported by National Science Council of ROC Grant NSC 95-2221-E-324-039.

Yu-Lung Lo is with the Chaoyang University of Technology, Wufong Township Taichung County, 41349 Taiwan. Phone: (04)2332-3000 ext. 7121; fax: (04)2374-2337; e-mail: yllo@ cyut.edu.tw.

Chun-Hsiung Wang is with the Chaoyang University of Technology, Wufong Township Taichung County, 41349 Taiwan. E-mail: s9514612@cyut.edu.tw.

the suffix tree is also used as indices for music feature strings to help searching in music database [3][12].

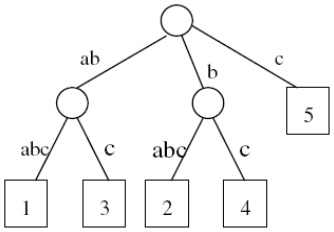

Fig. 1. An example of suffix tree for music string - "ababc"

B. *Numeric Indexing for Music Data*

In 2000, Jagadish, et al. proposed numeric mapping which maps a string into a real value [10]. Later on, a numeric indexing technique for music data was proposed by Lo, et al. [16][17]. If the music data can be represented by a numeric value, the R-tree and many other numeric index structures can be used to construct the index for music data. For translating music data into numeric value, let's assume that the music symbols, 'a', 'b', 'c', …, 'm', also can map into integer values 0, 1, 2, … *m*-1, respectively. If we pick out a music segment with *n* sequential notes from a melody feature string, denoted $x_1, x_2, \ldots, x_n$, the integer value of each note can be represented by $P(x_i)$, $1 \le i \le n$. Therefore, this segment of *n* sequential notes can be transformed into a numeric value by the conversion function – $v(n)$, as shown below.

$$v(n) = \sum_{x=1}^{n} P(x) \times m^{x-1} \quad \ldots (1)$$

Afterward, the converted numeric value can be inserted into the index, such as R-tree [8], for music data retrieval.

III. Existing Multi-Feature Indexing For Music Data

In the researches of indexing for music database retrieval, most of existing works were concentrated in constructing single-feature index structures for query searching: for instance, in 1999, the Key Melody Extraction and N-note Indexing by Tseng [19], Melodic Matching Techniques by Uitdenbogerd, et al. [20], and Approximate String Matching Algorithm by Liu, et al. [15]; in 2000, Query by Music Segments by Chen, et al. [3]; and in 2002, Numeric Indexing by Lo, et al. [16]. There are only a couple of researches emphasized on how to create a multi-feature index for music data retrieval. The most of recent works are Multi-Feature Index Structures [12] and Multi-feature Numeric Indexing [17]. We briefly discuss these two approaches in the following sections.

A. *Grid-Twin Suffix Trees*

There were four multi-feature index structures for music data retrieval proposed by Lee and Chen [12], in which it consists of Combined Suffix Trees, Independent Suffix Trees, Twin Suffix Trees, and Grid-Twin Suffix Trees. They claimed that the structure of Grid-Twin Suffix Trees provides most scalability among them. Since the structure of Grid-Twin Suffix Trees is an improved version from Twin Suffix Trees, let's introduce the Twin Suffix Trees first.

There could be two music features in the Twin Suffix Trees and each feature has its own index structure of independent suffix tree. There are links between them pointing from each node in one independent suffix tree to the corresponding feature nodes in another independent suffix tree. For example, they use letters, i.e., 'a'. 'b', 'c', … , and '1', '2', '3', … , to represent the melody symbols and rhythm symbols, respectively, then they can be combined into a two-feature music string, such as "$a_1b_2a_2b_1a_2b_2c_2$". A portion of the Twin Suffix Tree for "$a_1b_2a_2b_1a_2b_2c_2$" is shown in Fig. 2. There are links between two trees, from nodes 'a' and 'b' in the 2nd level of melody suffix tree to corresponding nodes '1' and '2' in the 2nd level of rhythm suffix tree, respectively, to represent the melody string "ab" corresponding to rhythm string "12".

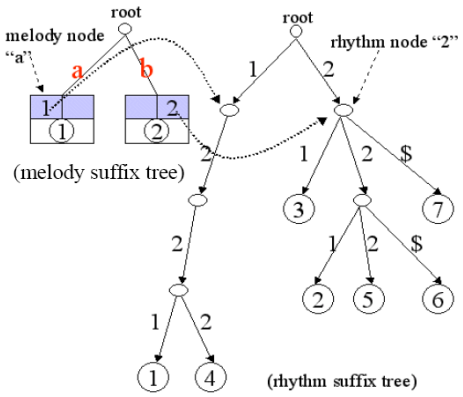

Fig. 2. Construction of the Twin Suffix Tree [12]

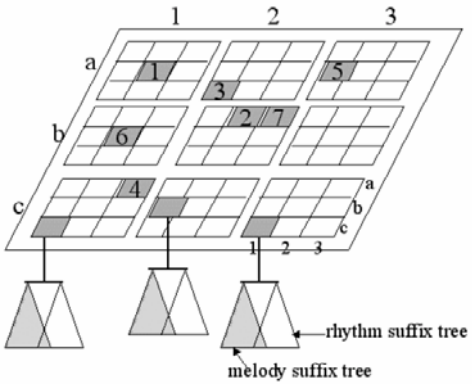

Fig. 3. An example of the Grid-Twin Suffix Trees[12]

Furthermore, the Fig. 3 shows an overview of the structure for Grid-Twin Suffix Tree. They first use a hash function to map each suffix of the feature string into a specific bucket of a 2-dimensional grid. The hash function uses the first *n* symbols of the suffix to map it into a specific bucket. Considering melody and rhythm only, the hash function is as following,

$$P(x,y) = \sum_{i=1}^{n} \left(\frac{(Num_m)^n}{(Num_m)^i} M_i, \frac{(Num_r)^n}{(Num_r)^i} R_i \right) \quad \ldots (2)$$

where *x* and *y* are the row and column coordinates, respectively, and $P(x, y)$ denotes the position of the bucket. The Num_m, Num_r, M_i, and R_i are the sizes of the melody and rhythm, the values of the *i*th symbols of melody and rhythm, respectively. The length of the suffix is denoted by *n*. Suppose Num_m and Num_r are both assumed to be 3 in Figure 3 and the values that represent melody symbols 'a', 'b', and 'c', and rhythm symbols '1', '2', and '3' are same as 0, 1, and 2,

respectively. To insert the first 2 symbols of the music feature string "$a_1b_2a_2c_1a_3$", said "a_1b_2", it can be computed as $P(x, y)$ = ($3^2/3^1$*0,$3^2/3^1$*0) + ($3^2/3^2$*1, $3^2/3^2$*1) = (0, 0) + (1, 1) = (1, 1). Then it can be mapped into the (1, 1) bucket in Figure 3. After hashing all suffixes, the remaining symbols of feature string following the suffixes are used to construct the Twin Suffix Trees and accompanied under the buckets as shown in Fig. 3.

B. Multi-Feature Numeric Index

The Multi-Feature Numeric Index for music data retrieval was proposed by Lo and Chen [17]. Like in Section 3.1, each music feature segment can be converted into a numeric value by equation (1) and these values for a music feature segment can be looked as a coordinate for multi-dimensional space. Such that the coordinate can be inserted into a multi-dimensional index tree, such as R-tree [8], for music retrieval. For example, let's assume that there are two music features, melody with ten distinct melody notes of 'a' to 'j' and rhythm with ten distinct symbols of '0' to '9', such that the symbols of both features can be individually mapped into integer values of 0 to 9, respectively. If there is a 2-feature music segment denoted by "$a_3b_3c_3a_3$", such that melody string and rhythm string can be extracted as "abca" and "3333", respectively. Consequently, these two feature strings can be transformed into numeric values 210 and 2222 by equation (1). Thereafter, the 2-dimensional coordinate, (210, 2222), of the music segment can be inserted into a 2-dimensional index R-tree for music retrieval. It also can be extended for converting 3 or more features into high dimensional index tree.

C. Discussions

Although, the authors claimed that Grid-Twin Suffix Trees provides more scalability than the other three index structures in [12]. However, this approach didn't point out the index structure for Twin Suffix Trees, if there are three or more music features. Therefore, the scalability of Grid-Twin Suffix Trees for number of music features is unclear.

In addition, since numeric index is created by transforming fixed length, n in equation (1), of music segment into numeric value, the main drawback of Multi-Feature Numeric Index is that the length of a query (Query By Example, QBE) is inflexible. It had better equal to the length of music segment which the index created, otherwise, searching time for the query will be a multiple times increasing.

IV. Grid Suffix Trees with Bit Arrays for Multi-Feature Music Indexing

In this section, we address the problem of Twin Suffix Trees for three or more music features and propose a new multi-feature music index structure called Grid Suffix Trees with Bit Arrays. Our approach is not only more scalable but also less memory space needed for index.

To construct Grid Suffix Trees with Bit Arrays, the grid structure as in Grid-Twin Suffix Trees is extended to higher dimensions and the suffix trees with bit arrays in non-leaf nodes is redesigned to instead of the Twin Suffix Trees under the grid structure. Suppose there are d features in music data and we organize the creating of our approach in the following two steps:

Step 1. **Computing entry in grid structure:** We first create a d-dimensional grid structure. For each music feature string, the first n symbols of the music segment will be transformed into a coordinate as the entry in grid structure. We extend the equation (2) and design new equation (3) for d-feature coordinate $P(x_1, .., x_d)$ as follows,

$$P(x_1,..,x_d) = \sum_{i=1}^{n} (F_1(i) \cdot N_1^{n-i},..,F_d(i) \cdot N_d^{n-i}) \quad \ldots (3)$$

Where $F_1(i)$, …, $F_d(i)$ and N_1, …, N_d represent the values and sizes of alphabet symbols, respectively, for d music features. The coordinate $P(x_1, .., x_d)$ can direct the entry in the grid structure. We note that the transformation of a prefix within any music segments, such as "a_1" or "a_1b_2", will have only one corresponding coordinate. An example of 2-feature grid structure is shown in Fig 4.

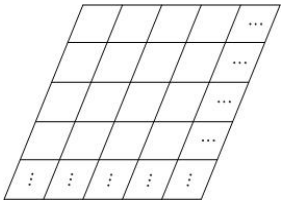

Fig. 4. 2-feature grid structure

Step 2. **Constructing Suffix Trees with Bit Arrays:** The remaining symbols behind the first n symbols of suffix are then used to construct the suffix trees accompanied under f the d-dimension grid structure at $P(x_1, .., x_d)$. Instead of the links between corresponding feature nodes in Twin Suffix Tree, we create the bit arrays to indicate the relationships between suffix trees. Each non-leaf node of suffix tree for the first feature consists of d-1 bit arrays. The number of entries (bits) for each array is the number of symbols (m) for each of corresponding feature such that the bit arrays can record the relationships (or virtual links) of this first feature and each of other features. Similarly, each non-leaf node in the suffix tree of second feature consists of d-2 bit arrays to record the relations with other d-2 features. Furthermore, the non-leaf nodes in the suffix trees of third, fourth, and etc. features should consist of the bit arrays in the same manner except the suffix tree of last feature without a bit array. The relationship between two suffix trees denotes the occurrence of the symbol combinations of two features. For example, "a_1b_2" representing melody 'a' and 'b' combined with rhythm '1' and '2', respectively, may occur in one or some music. If there is a bit array in each node of melody suffix tree recorded its relationship with rhythm suffix tree, the corresponding bits in the bit array of node 'a' related to node '1' and node 'b' related to node '2' will be marked as 1. Therefore, the structures of suffix trees with bit arrays for 2 features and 3 features can be presented in Fig 5 and Fig 6. In Fig 5, there is a bit array in each of non-leaf node of melody suffix tree indicating the relationship with the rhythm suffix tree. The first bit in the bit array of 'a' marked '1' denotes that 'a_1' occurs in some music. Likewise, in Fig 6, there are two bit arrays in melody suffix tree indicating the relationships with rhythm suffix tree and chord suffix tree.

There also has a bit array in each non-leaf node of rhythm suffix tree indicating its relationship with chord suffix tree. The structure of complete Grid Suffix Trees with Bit Arrays for 2-feature music index is shown in Fig 7.

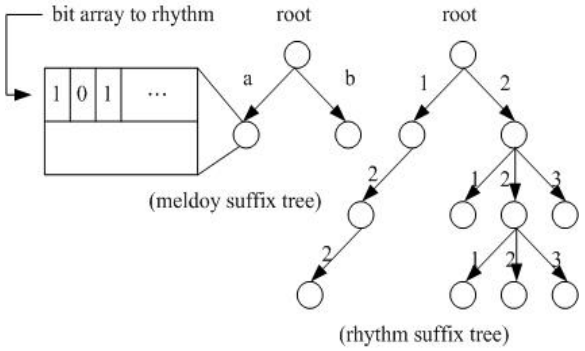

Fig. 5. Bit array in non-leaf node of 2-feature suffix trees

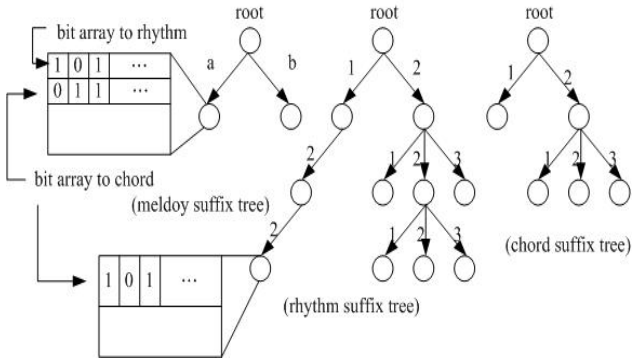

Fig. 6. Bit arrays in non-leaf nodes of 3-feature suffix trees

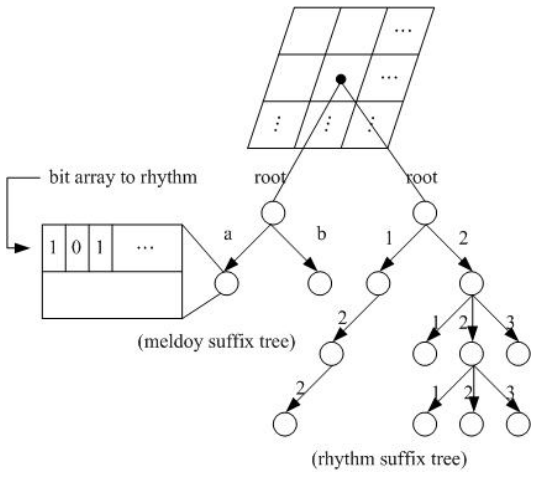

Fig. 7. Complete Grid Suffix Trees with Bit Arrays

To retrieve the Grid Suffix Trees with Bit Arrays, we first use equation (3) to convert the first *n* symbols of query string into corresponding coordinate. This coordinate is the entry in the grid structure and we continue to examine the remaining symbols of query in the suffix trees under the entry. Unlike in Grid-Twin Suffix Tree, there is no link being examined between any two suffix trees. There only one suffix tree which includes all of the features in the query needs to be examined. If the search for a query string is exactly match the symbols and bit arrays in that suffix tree, the target music can be found.

V. Performance Study

In order to evaluate the performance of our new approach, a series of experiments are performed in this section. Since, in Section III, we have discussed that the disadvantage of Multi-Feature Numeric Index [17] is inflexible for query lengths, it will not be used for comparison in our study. The Grid-Twin Suffix Trees [12] (GTST) and our Grid Suffix Tree with Bit Array (GST-BA) are without any specific limitation for queries. Therefore, we would like to know how memory is desired by comparing GTST with GST-BA. We consider following four factors in our experiments and the parameters used are also listed in Table 1.

(a) The effect of length of feature strings (*l*).
(b) The effect of number of symbols for each features (N_i).
(c) The effect of music database size.
(d) The effect of number of music features (*d*).

Table 1.Parameters for experiments

Parameter	Domain
No. of symbols for creating grid structure (*n*)	2
Average length of feature strings(*l*)	8~16
No. of symbols for each feature(N_i)	10~30
Database size(music segments)	50K~250K
No. of music feature(*d*)	2~5

A. The Effect of Length of Feature Strings

The length of feature string (*l*) denotes the string length of single music feature which will be used to create suffix tree for index. For example, there is a music segment "$a_1b_2a_2b_1a_2b_2c_2$" which consists of "abababc" and "1221222" two feature strings with length 7. The length of feature strings in a music database will affect the memory needed for constructing the music index. Since the structure of Twin Suffix Trees for 3 or more music feature was not pointed out for GTST in [12], we will only examine 2 music features in this study. In this experiment, we investigated melody and rhythm as the two features of music. Suppose that there are 20 notes most frequently used, such that, the number of symbols for melody can be set to 20. On the other hand, the rhythm consists of 1/8, 1/4, 1/2, 3/8, 3/4, $1^1/_2$, $1^3/_4$, 2 and etc., we assume the most frequently used of rhythm is 15 of them the number of symbols for rhythm can be 15. The average lengths for music feature strings we examined are classified into 5 categories, 8, 10, 12, 14, and 16. The first 2 symbols of each feature string are transformed into coordinates as the entry to grid structure, then, the remaining symbols of feature string are used to create the corresponding suffix tree. Since, in research of music database retrieval, it usually stores the music themes or repeating segments in databases instead of storing entire melodies for saving storage [9][14][18], the database size is set on 100,000 music feature segments. The experiment result for the effect of length of feature strings is shown in Fig. 8.

In the Fig. 8, the memories needed for GTST and GST-BA are increasing as average length of feature strings growing. However, GTST consumes memory more urgently than GST-BA does. We can see that the memories required for GTST and GST-BA are very close when the average length of feature strings is 8. Nevertheless, GST-BA can have 37% of memory saving compared to GTST at average length being 14. The memory saving for GST-BA is keeping extended as average length of feature strings increasing. This experiment demonstrates that GST-BA is more scalable than GTST in the length of feature string in music databases.

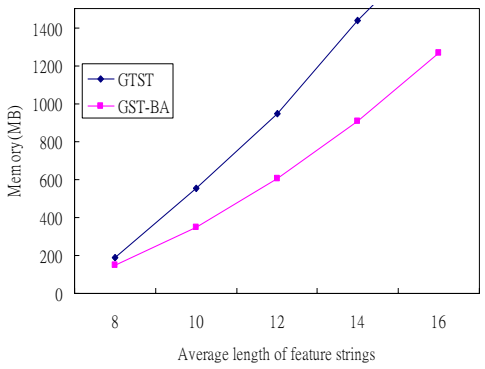

Fig. 8. Memory needed for average length of feature strings

B. *The Effect of Number of Music Features*

For GST-BA, the number of dimensions for grid structure and the number of bit arrays for non-leaf nodes are depended on the number of music features (*d*). We would like to discuss the effect of number of music features in this section. Furthermore, the structure of Twin Suffix Trees for 3 or more music feature was not clear for GTST in [12], we only examine the performance of GST-BA alone in this study. In the experiment, we investigated the number of music features form 2 to 5, 20 symbols for each feature, length 12 for each feature string, and the database size still set on 100,000 music feature segments. The experimental result is given in Fig. 9. The memory consuming for GST-BA versus the number of music features represents a linear growing up behavior. It clarifies the scalability of GST-BA and should be good for constructing the index for large music databases.

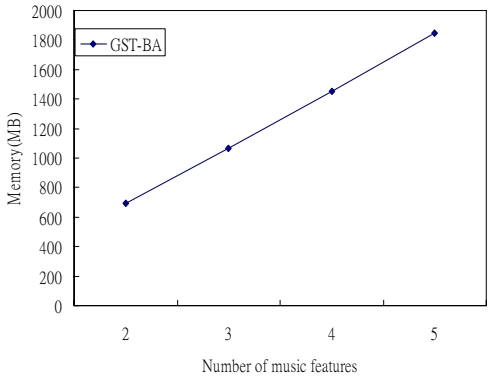

Fig. 9. Memory needed for no. of music features

VI. Conclusions

There are many features in music data, such as melodies, rhythms, chords and the tone differences of adjacent notes. We can extract these features to develop the multi-feature index to help the query searching quickly and to improve the accuracy of query results. The researches on multi-feature indexing for music data is relative rarer to the researches of music data retrieval. In this paper, we propose an index structure, named Grid Suffix Trees with Bit Arrays, for music data retrieval. Our approach creates bit arrays in non-leaf nodes of suffix trees instead of link pointer in Twin Suffix Trees to denote the relationships among suffix trees. We examined our proposed index structure by comparing with Grid-Twin Suffix Tree in varied parameters of experiments. As expected, the experimental results show that the memory needed for our Grid Suffix Trees with Bit Arrays is far less than it is needed for Grid-Twin Suffix Trees. It also demonstrated that our approach is more scalable to support large music databases.

References

[1] S. Blackburn and D. DeRoure, "A Tool for Content-based Navigation of Music," *In Proc. Of ACM Multimedia*, Pages 361-368, 1998.
[2] James C.C. Chen and Arbee L.P. Chen, "Query by Rhythm An Approach for Song Retrieval in Music Databases," *In Proc. Of Int'l Workshop on Research Issues in Data Engineering*, Pages 139-146, 1998.
[3] Arbee L.P. Chen, M. Chang, J. Chen, J.L. Hsu, C.H. Hsu, and Spot Y.S. Hua, "Query by Music Segments: An Efficient Approach for Song Retrieval," *In Proc. of IEEE Int'l Conf. on Multimedia and Expro*, 2000.
[4] G. Davenport, T.A. Smith, and N. Pincever, "Cinematic Primitives for Multimedia," *IEEE Computer Graphics & Applications*, Pages 67-74, July 1991.
[5] Y.F. Day, S. Pagtas, M. Iino, A. Khokhar, and A. Ghafoor, "Object-Oriented Conceptual Modeling of Video Data" *In Proc. Of IEEE Data Engineering*, Pages 401-408,1995.
[6] E.A. El-Kwae and M.R. Kabuka, "Efficient Content-Based Indexing of Large Image Databases," *ACM Trans. On Information Systems*, Vol. 18, No. 2, Pages 171-210, April 2000.
[7] Shen-Tat Goh and Kian-Lee Tan, "MOSAIC: A Fast Multi-Feature Image Retrieval System," *Data & Knowledge Engineering 33*, Pages 219-239, 2000.
[8] A. Guttman, "R-Trees A Dynamic Index Structure For Spatial Search," *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, Pages 47-57, 1984.
[9] J.L. Hsu, C.C. Liu, and Arbee L.P. Chen, "Efficient Repeating Patterrn Finding in Music Databases," *In Proc. of ACM Int'l Conf. on Information and Knowledge Management*, 1998.
[10] H.V. Jagadish, N. Koudas, and D. Srivastava, "On Effective Multi-Dimensional Indexing for Strings," *In Proc. of ACM SIGMOD*, Pages 403-414, May 2000.
[11] C. L. Krumhansl, "Cognitive Foundations of Musical Pitch," *Oxford University Press*, New York, 1990.
[12] W. Lee and A.L.P. Chen, "Efficient Multi-Feature Index Structures for Music Data Retrieval," *In Proc. Of SPIE Conf. on Storage and Retrieval for Image and Video Database*, 2000.
[13] Chia-Han Lin and Arbee L. P. Chen, "Indexing and Matching Multiple-Attribute Strings for Efficient Multimedia Query Processing," *IEEE Transactions On Multimedia*, Vol. 8, No. 2, April 2006.
[14] C.C. Liu, J.L. Hsu, and Arbee L.P. Chen, "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases," *In Proc. of IEEE Data Engineering*, Pages 14-21, 1999.
[15] C.C. Liu, J.L. Hsu, and Arbee L.P. Chen, "An Approximate String Matching Algorithm for Content-Based Music Data Retrieval," *In Proc. of IEEE Int'l Conf. on Multimedia Computing and Systems*, Pages 451-456, 1999.
[16] Yu-lung Lo and Shiou-jiuan Chen, "The Numeric Indexing For Music Data," *Proceedings of the IEEE 22nd Int'l Conference on Distributed Computing Systems (ICDCS'2002) Workshops – the 4th Int'l Workshop on Multimedia Network Systems and Applications (MNSA'2002)*, Vienna, Austria, July, Pages 258-263, 2002.
[17] Yu-lung Lo and Shiou-jiuan Chen, "Multi-feature Indexing For Music Data," *IEEE 23nd International Conference on Distributed Computing Systems (ICDCS'2003) Workshops – the 5th International Workshop on Multimedia Network Systems and Applications (MNSA'2003)*, Providence, Rhode Island, USA, Pages 654-659, May 19-22, 2003.
[18] Yu-lung Lo, Wen-Ling Lee, and Lin-huang Chang, "True Suffix Tree Approach for Discovering Non-trivial Repeating Patterns in a Music Object," accepted by *Journal of Multimedia Tools and Applications*, Springer, to appear.
[19] E. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," *Journal of Association for Computing Machinery*, Pages 262-272, 1976.
[20] Y.H. Tseng, "Content-Based Retrieval for Music Collections," *In Proc. of ACM SIGIR'99*, Pages 176-182, 1999
[21] A.L. Uitdenbogerd and J. Zobel, "Melodic Matching Techniques for Large Music Databases," *In Proc. of ACM Multimedia*, Pages 57-66, 1999.
[22] E. Ukkonen, "On-Line Construction of Suffix Tree," *Algorithmica*, Vol. 14, Pages 249-260, 1995.
[23] P. Weiner, "Linear Pattern Matching Algorithms," *in Proc. Of IEEE Ann. Symp. On Switching and Automata Theory*, Pages 1-11, 1973.