Using Dual Ternary Indexing for Music Retrieval Systems

Chuan-Wang Chang and Hewijin Christine Jiau
Department of Electrical Engineering
National Cheng Kung University, Tainan City 701, Taiwan
chuan@ee.ncku.edu.tw and jiauhjc@mail.ncku.edu.tw

ABSTRACT

Space requirement for storing indexes and performance for query processing are two critical issues in music information retrieval (MIR) system. To overcome difficulties in variable length of queries and enhance efficiency of music retrieval, we propose an effective and efficient numeric indexing structure. It differs greatly from pre-existing researches in textual indexing techniques. We show how the development of this framework has been motivated and demonstrate how the technique may be naturally applied to solve this two fundamental MIR issues. Experiments are performed to compare our method with previous solutions. The results show that our method is more scalable and economical than previous methods. The method we proposed can achieve dramatically and significantly improvement in saving time and storing space for retrieving and indexing.

1. INTRODUCTION

The rapid growth of computer science has resulted in a rapid increase of the size of digital multimedia data collections and greatly increases the availability of multimedia contents to user. Digital music is one of the most popular data types distributed by the Internet. Large web-based music collections are continuing to grow in size exponentially. Clearly, the ability to effectively and efficiently manage the great amount of music data is desirable. Music information retrieval (MIR) system is expected to be such a system that has powerful ability to handle music data and to satisfy the requirements of users.

Ghias, *et al.* [1] transform a music object into a string which consists of three kinds of symbols ("U", "D" and "S" stand for a note which is higher than, lower than, or equal to the previous note, respectively). The string can be regarded as a coarse melodic contour of the music. Then, the problem of music retrieval is transformed into string matching.

No matter how the pre-existing methods [1-4] work, they view the music object as a string, and the processes of indexing and retrieval are by means of text-related techniques.

Suffix tree [5, 6] is a tree-based index structure which is widespread used in MIR systems. If $T = t_1 t_2 t_3 ... t_i ... t_n$ is a string, then $T_i = t_i t_{i+1} ... t_n$ is the *suffix* of T that starts at position i. It means a suffix is a substring that includes the final character of the string. Figure 1 shows a suffix tree for the string T='bcababc'. We can find that 'c', 'bc', 'abc', 'babc', 'ababc', 'cababc', and 'bcababc' are suffixes of T. The leaf nodes indicate the starting position of the corresponding suffix.

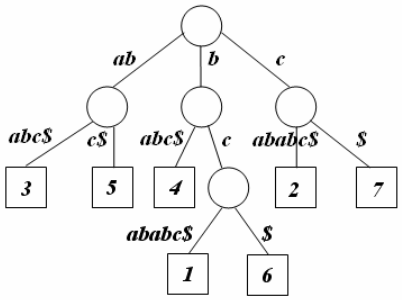

Figure 1. The suffix tree representation of T='bcababc'.

N-gram is another technique widely used in MIR systems [7, 8, 9]. The sequence of characters (symbols) is divided into overlapping constant-length subsequences (substrings). A string formed from n adjacent characters within text is called an *n-gram*. For example, the string 'abababc' comprises the following 4-grams: 'abab', 'baba', 'abab' and 'baba'. Downie [7] adopted this technique to evaluate his MIR system. The songs were converted to a sequence of intervals and then using a gliding window to fragment the sequence into overlapping length-n substrings. These n-grams were then encoded as text words. Hence, each song can be viewed as a text document, and a pre-existing text search engine can be used for text retrieval.

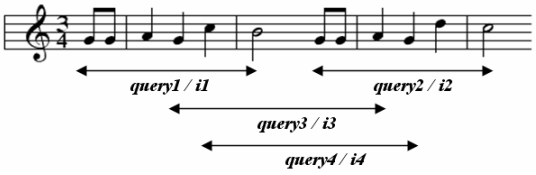

Figure 2. An excerpt of 'Happy Birthday to You'.

However, it is quit obvious that the memory usage for storing indexes by using suffix tree or n-gram method is very large. The indexes are full of redundancies. Since most constituents of suffix tree and n-grams are unnatural and dissimilar to human habit. People tend to submit a perceptually meaningful query segment (phrase). For example, Figure 2 shows an excerpt of a familiar song--

'Happy Birthday to You'. The *query1* and *query2* are two possible meaningful queries. Because they seem to be two clear phrases, while *query3* and *query4* are unlikely to be hummed. Unfortunately, using *n*-gram method (6-gram in this case), *i3* that is composed of 'sol-do-si-sol-sol-la' and *i4* that is composed of 'do-si-sol-sol-la-sol' are included in indexes. This phenomenon indicates that the indexes contain many unnecessary data. The space for storing indexes will become smaller when redundancies pruning is conducted.

To reduce memory space requirement and enhance query processing, in this paper, we propose an efficient indexing approach based on the advantages of numeric indexing technique. Instead of textual indexing techniques, the performance of memory usage and retrieval for MIR by using numeric indexing techniques is better.

The rest of this paper is organized as follows. In Section 2, we introduce terminologies and notations used in this paper. Section 3 describes the basic concepts of our proposed indexing techniques. In Section 4, we demonstrate the accomplished experimental results. Finally, in Section 5, we summarize this paper.

2. TERMINOLOGY AND NOTATION

To illustrate the proposed indexing strategies, introduce terminologies and notations used in this paper is necessary. In our method, music are segmented to phrases [10] and transformed into ternary melodic contours. Figure 3 sketches several terminologies. The definitions of specific terms used in this paper are as follows.

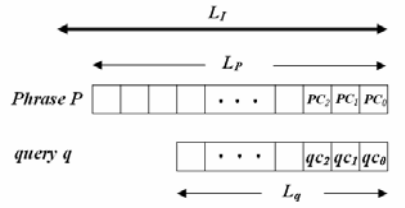

Figure 3. A simple diagram shows several terminologies.

- *P*: melodic contour of a music phrase that intends to be indexed
- PC_i: the *i*th digit of phrase contour *P*
- *q*: a query submitted by user
- qc_i: the *i*th digit of query contour *q*
- L_I: the maximum index length of MIR system
- L_P: the length of phrase *P*
- L_q: the length of query *q*

A music phrase is denoted as a 5-tuple *(I_{PRE}, I_{SUF}, length, Song_id, Phrase_num)*, where

- I_{PRE} indicates the first index of music phrase *P*.
- I_{SUF} indicates the second index of music phrase *P*
- *length* denotes the length of music phrase *P*.
- *Song_id* represents the identification number of the music.
- *Phrase_num* is the serial number of segmented phrase that belongs to the music with the number of *Song_id*.

For example, suppose a phrase *P* with length of 4 is the first phrase that extracted from the song *w2s1*, the first index is 3 and the second index is 4, then the phrase *P* is denoted as *(3, 4, w2s1, p1)*.

3. THE PROPOSED INDEXING TECHNIQUES

3.1 DUAL TERNARY INDEXING APPROACH

In order to decrease the size of indexes and accelerate music retrieval, we introduce the advantage of numeric indexing technique and propose an effective and efficient indexing structure. The essence of the proposed numeric indexing scheme is adoption of perceptually meaningful phrase to establish index unlike suffix tree and *n*-gram methods. Besides, the proposed method is regardless of phrase length, that is, it needs not to divide the phrases into overlapping constant-length subsequences in advance.

Figure 4 shows the basic idea of the proposed numeric indexing structure. The extracted representative fragments (music phrases) are indexed by *f(c)*, where *f( )* is a mapping function while *c* is the melodic contour of music phrase. That is, a phrase with key *c* maps to slot *f(c)*. Here, phrases have the same *f(c)* are stored in the same slot. The submitted query is used to retrieve phrases that have the same *f(c)*.

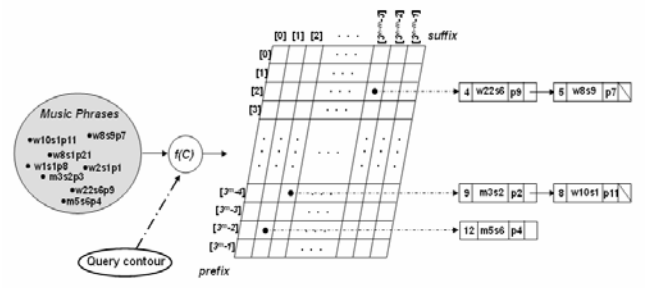

Figure 4. Basic idea of the proposed indexing structure.

Because of the different segmentation for phrases between user's cognition and MIR system's judgment, user may submit a self-determined phrase that differs from original phrase segmented by MIR system.

To handle various kinds of query cases, we adopt dual index structure. The first index I_{PRE} and the second index I_{SUF} are derived from the phrase *P* directly.

In general, we can constrain the total length $L_I \le L_{I_{PRE}} + L_{I_{SUF}}$ of index to prevent the explosively increase of space requirement.

The indexing functions are defined as

$$\boldsymbol{I_{PRE}} = \boldsymbol{f}(\boldsymbol{P_{PRE}}) = \sum_{i=0}^{i=L_{I_{PRE}}-1} \boldsymbol{PC}_{L_P-1-i} \times 3^{L_{I_{PRE}}-1-i} \quad (1)$$

$$\boldsymbol{I_{SUF}} = \boldsymbol{f}(\boldsymbol{P_{SUF}}) = \sum_{i=0}^{i=L_{I_{SUF}}-1} \boldsymbol{PC}_i \times 3^{L_{I_{SUF}}-1-i} \quad (2)$$

To show feasibility and advantages of the proposed methods, all kinds of cases are considered and comprehensive discussions are made in the following.

Case 1. If $q=P$, $L_q \le L_{I_{PRE}}$ and $L_q \le L_{I_{SUF}}$

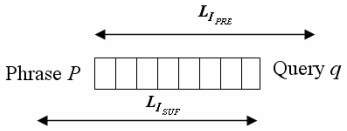

Figure 5. A diagram to explain Case 1.

Figure 5 shows the concept of this case. According to Eq.(1) and Eq.(2), phrase *P* is indexed as

$$\boldsymbol{I_{PRE}} = \boldsymbol{f}(\boldsymbol{P_{PRE}}) = \sum_{i=0}^{i=L_P-1} \boldsymbol{PC}_{L_P-1-i} \times 3^{L_{I_{PRE}}-1-i} \quad (3)$$

$$\boldsymbol{I_{SUF}} = \boldsymbol{f}(\boldsymbol{P_{SUF}}) = \sum_{i=0}^{i=L_P-1} \boldsymbol{PC}_i \times 3^{L_{I_{SUF}}-1-i} \quad (4)$$

Since *q*=*P*, we have

$f(q_{PRE}) = f(P_{PRE})$ and $f(q_{SUF}) = f(P_{SUF})$.

Thus, phrase *P* can be exactly retrieved.

Case 2. If *q*=*P* and $\boldsymbol{L_q} \le (\boldsymbol{L}_{\boldsymbol{L_{PRE}}} + \boldsymbol{L}_{\boldsymbol{L_{SUF}}} - \boldsymbol{L}_{(\boldsymbol{L_{PRE}} \cap \boldsymbol{L_{SUF}})})$

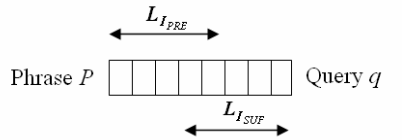

Figure 6. A diagram to explain Case 2.

As shown in Figure 6, in this case, phrase *P* is indexed by Eq.(1) and Eq.(2).

Since *q*=*P*, we have

$f(q_{PRE}) = f(P_{PRE})$ and $f(q_{SUF}) = f(P_{SUF})$.

Thus, phrase P can also be exactly retrieved.

Case 3. if $\boldsymbol{L_P} > \boldsymbol{L_I} = (\boldsymbol{L}_{\boldsymbol{I_{PRE}}} + \boldsymbol{L}_{\boldsymbol{I_{SUF}}})$ and *q*=*P*

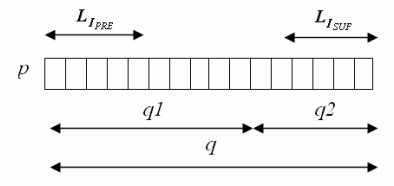

Figure 7. A diagram to explain Case 3.

Consider Figure 7, it illustrates the case when the length of extracted phrase is longer than the length of system-allowed index. Due to the limitation of index length, phrase *P* is indexed as

$$\boldsymbol{I_{PRE}} = \boldsymbol{f}(\boldsymbol{P_{PRE}}) = \sum_{i=0}^{i=L_{I_{PRE}}-1} \boldsymbol{PC}_{L_P-1-i} \times 3^{L_{I_{PRE}}-1-i} \quad (5)$$

$$\boldsymbol{I_{SUF}} = \boldsymbol{f}(\boldsymbol{P_{SUF}}) = \sum_{i=0}^{i=L_{I_{SUF}}-1} \boldsymbol{PC}_i \times 3^{L_{I_{SUF}}-1-i} \quad (6)$$

If *q*=*P*, we have

$$\boldsymbol{f}(\boldsymbol{q_{PRE}}) = \sum_{i=0}^{i=L_{I_{PRE}}-1} \boldsymbol{qc}_{L\boldsymbol{q}-1-i} \times 3^{L_{I_{PRE}}-1-i} = \boldsymbol{f}(\boldsymbol{P_{PRE}})$$

$$\boldsymbol{f}(\boldsymbol{q_{SUF}}) = \sum_{i=0}^{i=L_{I_{SUF}}-1} \boldsymbol{qc}_i \times 3^{L_{I_{SUF}}-1-i} = \boldsymbol{f}(\boldsymbol{P_{SUF}}).$$

Hence, phrase *P* can be exactly retrieved.

Case 4. if $\boldsymbol{L_P} > \boldsymbol{L_I} = (\boldsymbol{L}_{\boldsymbol{I_{PRE}}} + \boldsymbol{L}_{\boldsymbol{I_{SUF}}})$ and $\boldsymbol{L}_{\boldsymbol{q}1} < \boldsymbol{L_P}$

This case is similar to *Case 3*, but the query is *q1*. By Eq.(5), since

$$\boldsymbol{f}(\boldsymbol{q}1_{\boldsymbol{PRE}}) = \sum_{i=0}^{i=L_{I_{PRE}}-1} \boldsymbol{qc}_{L_{q1}-1-i} \times 3^{L_{I_{PRE}}-1-i} = \boldsymbol{f}(\boldsymbol{P_{PRE}}),$$

the MIR system will carry out a process of *suffix query expansion* (will be described in the next subsection) and return many recalls that have the same prefix as *q1*, of course including phrase *P*, in spite of $\boldsymbol{f}(\boldsymbol{q}1_{\boldsymbol{SUF}}) \neq \boldsymbol{f}(\boldsymbol{P_{SUF}})$.

Case 5. if $\boldsymbol{L_P} > \boldsymbol{L_I} = (\boldsymbol{L}_{\boldsymbol{I_{PRE}}} + \boldsymbol{L}_{\boldsymbol{I_{SUF}}})$ and $\boldsymbol{L}_{\boldsymbol{q}2} < \boldsymbol{L_P}$

This case is similar to *Case 4*, if we assume the submitted query is *q2*, then from Eq.(5) we get:

$$\boldsymbol{f}(\boldsymbol{q}2_{\mathrm{PRE}}) = \sum_{i=0}^{i=L_{I_{PRE}}-1} \boldsymbol{qc}_{L_P-1-i} \times 3^{L_{I_{PRE}}-1-i} \neq \boldsymbol{f}(\boldsymbol{P_{PRE}})$$

and from Eq.(6) we obtain:

$$\boldsymbol{f}(\boldsymbol{q}2_{\boldsymbol{SUF}}) = \sum_{i=0}^{i=L_{I_{SUF}}-1} \boldsymbol{qc}_i \times 3^{L_{I_{SUF}}-1-i} = \boldsymbol{f}(\boldsymbol{P_{SUF}}).$$

In this case, after the *prefix query expansion* process (will be described in the next subsection), system will return many recalls that have the same suffix as *q2*, of course including phrase *P*, in spite of $f(q1_{PRE}) \neq f(P_{PRE})$.

3.2 QUERY EXPANSION FOR SIMILARITY MATCHING

In addition to improve recall, the process of *query expansion* is conducted in our prototype system. The *edit distance* between two contours (strings) is defined as the number or cost of editing operations that must be performed to make the contours identical. In general, there are three kinds of editing operations, namely *insertion* (inserting a note), *deletion* (deleting a note), and *substitution* (replacing a note).

The popular methods for measuring similarity between two strings are the dynamic programming algorithm [11] and the longest common subsequence algorithm [12]. Because of the array-based operation, they are space-consuming methods. Besides, searching the entire database will cause a seriously time-consuming problem.

Our proposed numeric indexing approach can extend the ability of approximate matching easily by conduct a query expansion process.

In our system, the goal of query expansion is to generate possible queries that similar to the original query *q* with certain edit distance *d*. Without losing any generality, we restrict the edit distance to 1 (*d* = 1) in our consideration, although it is easy to handle larger edit distance. The reason for this restriction is to reduce the explosively growth number of expanded queries. With the edit distance growth, the number of expanded query will become an inconsiderable status.

(A)*Query expansion for insertion error*

If *d* = 1, there are $3\times(L_q+1)-L_q$ queries will be expanded. The expansion function for *insertion* error is

define as

$$f(qe)=\begin{cases} f(q)\times 3+x & ,\ t=1 \\ \sum_{i=t-1}^{L_{qc}-1} qc_i\times 3^{i+1}+x\times 3^{t-1}+\sum_{i=0}^{t-2} qc_i\times 3^{i} & ,\ 2\le t\le \sim L_q \\ f(q)+x\times 3^{Lq} & ,\ t=L_q+1 \end{cases} \quad (7)$$

where $1\le t\le L_q+1$ and $x\in\{0,\ 1,\ 2\}$.

For example, the original query q is assumed as "10120" with length of Lq = 5. By Eq.(7), queries '10120*x*', '1012*x*0', '101*x*20', '10*x*120', '1*x*0120', and '*x*10120' will be expanded. Here, {96, 258, 276, 285, 288, 289, 290, 291, 294, 312, 339, 420, 582} is the set of expanded queries.

(B)*Query expansion for deletion error*

If d = 1, there are Lq queries will be expanded. The expansion function for *deletion* error is define as

$$f(qe)=\begin{cases} \sum_{i=t}^{i=Lq-1} qc_i\times 3^{i-1} & ,\ t=1 \\ \sum_{i=t}^{i=Lq-1} qc_i\times 3^{i-1}+\sum_{j=0}^{j=t-2} qc_j\times 3^{j} & ,\ 2\le t\le \sim L_q-1 \\ \sum_{j=0}^{j=t-2} qc_j\times 3^{j} & ,\ t=Lq \end{cases} \quad (8)$$

where $1\le t\le L_q$ and $x\in\{0,\ 1,\ 2\}$.

(C)*Query expansion for Substitution error*

If d = 1, there are $2Lq$ queries will be expanded. The expansion function for *substitution* error is define as

$$f(qe)=\begin{cases} \sum_{i=t}^{i=Lq-1} qc_i\times 3^{i-1} & ,\ t=1 \\ \sum_{i=t}^{i=Lq-1} qc_i\times 3^{i-1}+\sum_{j=0}^{j=t-2} qc_j\times 3^{j} & ,\ 2\le t\le \sim L_q-1 \\ \sum_{j=0}^{j=t-2} qc_j\times 3^{j} & ,\ t=Lq \end{cases} \quad (9)$$

where $1\le t\le L_q$ and $x\in\{0,\ 1,\ 2\}$.

(D)*Prefix/Suffix query expansion*

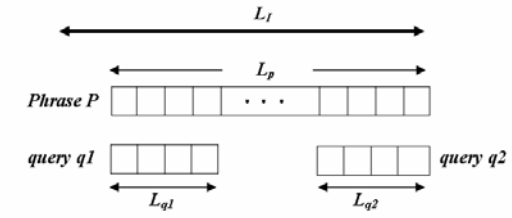

Figure 8. A diagram to explain query expansion.

Consider Figure 8, when submitting the input query *q1* that is the left part (or prefix) of phrase P and $L_{q1} < L_P$. By Eq. (3), we should not expect to be able to retrieve the phrase *P*, because

$$f(q1)=\sum_{i=0}^{i=L_{q1}-1-i} qc_{L_{q1}-1-i}\times 3^{L_{q1}-1-i} \neq \sum_{i=0}^{i=L_I-1} PC_{L_P-1-i}\times 3^{L_P-1-i}=f(P)$$

This circumstance occurs frequently when the determination of MIR system for phrase segment is different from user's perception. That is, the phrase P determined by MIR system with length of L_P maybe segmented into many shorter phrases by users.

If the input query *q2* is the right part (or suffix) of phrase P and $L_{q2} < L_P$, phrase P may be retrieved by carrying out the process of *prefix query expansion*. Query expansion is a technique commonly used in database system to improve recall. In this case, the goal of query expansion is to generate all possible queries that have the same suffix as input query. Thus, if $L_q < L_I$, the amount of possible queries is $3^{(L_I-L_q)}$. This process resembles the similarity matching for text retrieval.

Let us review Figure 8, suppose a phrase P = '010211', $L_I = 8$, and query *q2* = '0211', we must expand 3^4 queries to retrieve all possible results that have the same suffix as query *q2*. Figure 9 shows the results of prefix query expansion.

If query *q1*='0102' is submitted, since $L_{q1} \le L_I$, the process of *suffix query expansion* is conducted to retrieve possible music. Completing all possible expansions, there would result in $3^{(L_I-L_{q1})}$ queries. In this case, 3^4 extra queries are expanded, as shown in Figure 10. Then, phrase P is retrieved when the process of query expansion is conducted.

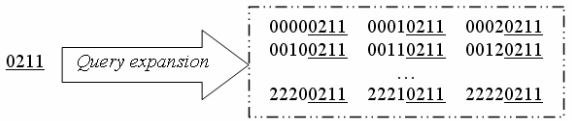

Figure 9. Example of *prefix query expansion* when query *q2* is the right part of phrase *P*.

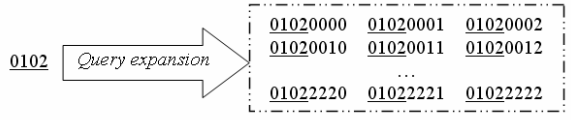

Figure 10. Example of *suffix query expansion* when query *q1* is the left part of phrase *P*.

In reality, the process of query expansion can be easily accomplished in our proposed indexing structure. The indexes are organized as a two dimensional array, the indexes in the same *row* have the same prefix, while the indexes in the same *column* have the same suffix. Hence, when a query *q* is submitted, the key of the *q* is calculated for exactly matching. Then, entries in the same *row* or *column* are retrieved for similarity matching. Figure11 and Figure 12 show the concepts of *suffix query expansion* and *prefix query expansion*, respectively. The shade indicates the possible retrieved phrases.

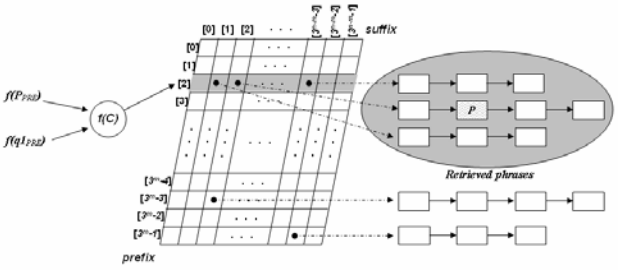

Figure 11. The concept of suffix query expansion.

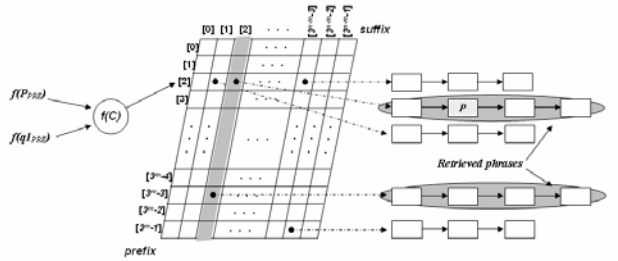

Figure 12. The concept of prefix query expansion.

4. EXPERIMENTAL RESULTS

To evaluate the effectiveness and efficiency of the proposed indexing method, we performed an extensive performance study of three methods: *n-gram*, *suffix tree*, and the proposed numeric indexing, with various kinds of index length.

We collected more than 1,000 MIDI files from the public domain to form the music database. To construct the music database, the melody data have to be extracted first. After melody extraction, the music database consists of over 374,000 notes. We transformed each song into note string. To evaluate the proposed numeric indexing approach, we segmented the melodic contours of song into perceptually meaningful phrases [10]. By pruning the repeating phrases, there are about 18,000 phrase stored in music database.

For *n*-gram method, we transform note strings into melodic contours. Consider a melodic contour with length of *n*. We partition the melodic contours to $n-m+1$ successive segments with specific length *m* and construct *interval blocks* for indexing [9]. Then, information of these melodic segments, such as the title and the occurrence position of the song, are stored in interval blocks. In exactly matching, we retrieve songs that contain identical piece with the query. In similarity matching, we retrieve songs that have the same melodic contour of query.

Table 1: Performance Comparison of Different Methods

Indexing Method \ Performance		Memory Consumption of Index Construction (KB)	Elapsed Time for Exactly Matching (Sec.)	Elapsed Time for similarity Matching (Sec.)
n-gram	*4-gram*	1,505	6.6226	9.0486
	5-gram	1,683	4.9723	6.7692
	6-gram	1,894	2.9015	3.6532
	7-gram	2,083	1.7098	2.1828
	8-gram	2,229	0.9772	1.2859
	9-gram	2,338	0.4537	0.7350
	10-gram	2,419	0.2324	0.4566
Suffix Tree		78,160	1.8642	3.4325
Our Method (I_{PRE}_I_{SUF})	*4_3*	242	< 0.001	0.063
	4_4	260	< 0.001	0.031
	4_5	286	< 0.001	0.016
	4_6	333	< 0.001	0.015
	4_7	452	< 0.001	0.001
	5_4	289	< 0.001	0.015
	5_5	336	< 0.001	0.016
	5_7	802	< 0.001	0.016
	6_4	342	< 0.001	0.016
	6_6	802	< 0.001	0.001
	7_4	447	< 0.001	0.001
	7_5	730	< 0.001	0.016

For suffix tree method, the goal of exactly matching is to traverse from the root of suffix tree to a specific leaf node according to the query string. Under the limitation of the depth for searching, the similarity matching is to return all leaves under a specific internal node.

For the proposed numeric indexing approach, the task of exactly matching is to retrieve all phrases that have both the same prefix and suffix as query. In addition, the task of similarity matching is to retrieve all phrases that have either the same prefix or suffix as query.

All experiments were conducted on a 2.8 GHz Intel Pentium 4 PC with 1G main memory, running Microsoft Windows XP professional. Table 1 demonstrates the memory usage for indexing and elapsed time for retrieving. In the aspect of memory usage, the suffix tree method is the worst (76 MB) because it enumerates all suffixes. The disk space cost by the *n*-gram method in various lengths are about 1~2 MB. The proposed numeric indexing

method can dramatically reduce the space requirement for storing indexes. In the aspect of retrieving performance, we randomly select 20 music excerpts as queries for further validation. Compare to these methods, using the proposed numeric indexing method can greatly decrease the elapsed time for exactly matching and similarity matching.

The depth of indexing is the most important factor which dominates the performance of that using suffix tree method. In the proposed numeric indexing approach, no matter what the index length is, the elapsed time for exactly matching is quite small (<0.001 second).

5. CONCLUSIONS

In this paper, we mainly focus on two issues: the size of indexes and the time for retrieving music. Because the indexes are full of redundancies by using pre-existing indexing methods, we adopt music phrase as the basic unit for further processing. In addition to overcome the difficulties caused by the difference between phrase extracted by system and phrase segmented by user, we introduced numeric indexing approach. Besides, the proposed indexing approach not limit to the length of phrase for indexing or querying. The experimental results demonstrate that our method performs more efficient and effective compared with the pre-existing methods.

6. REFERENCES

[1]A. Ghias, H. Logan, D. Chamberlin, and B.C. Smith, "Query by Humming：Musical Information Retrieval in an Audio Databases," *Proc. of the 3rd ACM International Conference on Multimedia*, pp. 231-236, 1995.

[2]A L.P. Chen, M. Chang, J. Chen, J.L. Hsu, C.H. Hsu, and S.Y.S. Hua, "Query by Music Segments: An Efficient Approach for Song Retrieval," *Proc. of IEEE International Conference on Multimedia and Expo*, pp. 873-876, 2000.

[3]J.L. Hsu, C.C. Liu, and A.L.P. Chen, "Discovering Nontrivial Repeating Patterns in Music Data," *IEEE Transactions on Multimedia*, pp. 311-325, 2001.

[4]W. Lee, and A.L.P. Chen, "Efficient Multi-Feature Index Structures for Music Data Retrieval," *Proc. of SPIE Conference on Storage and Retrieval for Image and Video Databases*, pp. 177-188, 2000.

[5]E. M. McCreight, "A Space-Economical Suffix Tree Construc-tion Algorithm," *Journal of Algorithms*, vol. 23(2), pp. 262-272, 1976.

[6]E. Ukkonen, "On-line Construction of Suffix Trees," *Algorithmica*, Vol. 14(3) pp. 249-260, 1995.

[7]S. Downie and M. Nelson, "Evaluation of a simple and effective music information retrieval method," *Proc. of 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 73-80, 2000.

[8]C. L. Yip and B. Kao, "A Study on N-gram Indexing of Music Features," *Proc. of IEEE International Conference on Multimedia and Expo*, pp. 869-872, 2000.

[9]C.W. Chang, C.Y. Chang and H.C. Christine Jiau, "A Practical Music Retrieval System Based on Sliding Melodic Contour," *Proc. of International Computer Symposium*, pp. 1162-1167, 2004.

[10]C.W. Chang and H.C. Christine Jiau, "Representative Music Fragments Extraction by Using Segmentation Techniques," *Proc. of International Computer Symposium*, pp. 1156-1161, 2004.

[11]S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two sequences," *Journal of Molecular Biology*, pp. 443-453, 1970.

[12]H. Thomas, et al, Introduction to Algorithms, The MIT press, 2001.