Audio Retrieval Based on Chinese Keyword Search in Relational Databases

Boyan Zhu, Guang Liu
College of Art
Hebei University
Baoding, Hebei, 071002 China
E-mail: zhu_boyan@yahoo.com.cn (B. Zhu)
E-mail: lg672@tom.com (G. Liu)

Liang Zhu
Key Laboratory of Machine Learning and Computational Intelligence, School of Mathematics and Computer Science
Hebei University
Baoding, Hebei, 071002 China
E-mail: zhu@hbu.edu.cn (L. Zhu)

***Abstract*—In this paper, we propose a new method based on Chinese keyword search to select the WAV or MP3 files in audio post-production. First, we listen to each file and label it with Chinese characters, and then classify and store the files in a relational database system. Then, we use the techniques of Chinese keyword search to match query characters and the tuple characters quickly, and to compute similarities between the query and candidate tuples. For the characteristics of Chinese keyword search, we present a ranking strategy and an algorithm to refine the candidate tuples resulting from the first round matching, and finally get top-*N* results of audio files. The experimental results show that our method is efficient and effective.**

Keywords-Relational database; Audio retrieval; Post-production; Chinese Keyword Search; Ranking strategy

I. INTRODUCTION

Audio retrieval has gained more attention of the research community, involving speech, music, and general environmental sounds [1, 2]. The existing work includes mainly four types: the methods based on DCMI (Dublin Core Metadata Initiative), techniques of traditional information retrieval (IR), content-based retrieval, and the retrieval methods for special audio information. In general, speech can be transformed into text by using the automatic speech recognition (ASR), and then we employ IR techniques for speech indexing and retrieval. Music retrieval will be based on pitch and a set of features, including structured music and sample-based music. For general environmental sounds, audio retrieval will deal with a variety of sound data such as bird songs, thunders, and applause. For the applications of general-purpose sounds, there are many challenging problems in audio retrieval, which requires specialized audio features.

In audio post-production of a film or a television program, the sound files are managed generally by a file system (say, explorer.exe in Windows XP). To obtain a desired audio in thousands of files, a user has to open some files and listen carefully to them one by one and again and again, it is not easy for the user to find audio files manually in the file system. In this paper we give a framework of managing sounds files by using a relational database management system (RDBMS) and the techniques of Chinese keyword search for audio retrieval.

Inspired by the success of free-form keyword search on information retrieval (IR) and Web search engines, i.e., it is popular to users who need not know query languages and the structure of underlying data. Researches of English keyword search with IR-style free-form in relational databases have been extensively studied since 2002 [3-5]. For the differences between Chinese and English, [6] proposed a method to process the Chinese keyword search. We will utilize the methods in [6, 7] for our audio retrieval in audio post-production, and then deal with top-*N* keyword queries. For example, if the description of a tuple t_0 has seven Chinese characters meaning “sound of wind-bell, slowly” in underlying database, and a query Q has also seven Chinese characters with the same meaning as t_0 matching five Chinese characters with t_0 but not matching completely, then we can obtain the top-*N* results sorted by a ranking strategy, and the tuple t_0 will be in the results since there are five matching Chinese characters between query Q and t_0, and the matching characters lead to a high similarity. Of course, the tuple t_1 with six Chinese characters meaning ‘sound of gentle wind, slowly’ may be in the top-*N* results with four matching Chinese characters for Q, and will rank behind the tuple t_0.

II. CONCEPTS AND TERMINOLOGIES

DCMES (the Dublin Core Metadata Element Set) [8]: it is a vocabulary of fifteen properties for use in resource description. The fifteen elements are “Title, Creator, Subject, Description, Publisher, Contributor, Date, Type, Format, Identifier, Source, Language, Relation, Coverage, Rights”.

Depending on the requirements of our application and referring to DCMES, we design our Foley library such that sound-relation has 16 attributes as follows: qqe

SoundTable(*Serial-number*, *Class*, *File-name*, *Sampling-ratio*, *Track*, *Mike-type*, *REC-model*, *Sound-class*, *REC-place*, *REC-creator*, *REC-date*, *Source*, *Size*, *Keywords*, *Description*, *Recommendation*).

Classifications and descriptions for sound files are the key steps in developing our application, which are heavy work manually. The attribute *Description* in ***SoundTable*** will play an important role in our application since reading text costs much less than hearing a sound file. For instance, a *Description* has thirty-three Chinese characters meaning ‘in a quiet street, at 1 minute 25 second, a person coughing far away, at 2 minute 10 second, sound of leaves trembling in a fit of breeze’.

Tuple word: Consider a relation R with m text attributes $\{A_1, A_2, \ldots, A_m\}$ and n tuples (i.e., $|R| = n$). For a tuple $t\in R$ and attribute $A\in\{A_1, A_2, \ldots, A_m\}$, $t[A]$ consists of one or more Chinese character(s), which is denoted by $\{z_1z_2z_3 \ldots z_k\}$. Each single Chinese character z_i ($1\leq i \leq k$) is called a Chinese tuple word (or a tuple word, for short). Also, we do not distinguish "Chinese character" and "Chinese word" in the following discussion. A **Chinese phrase** will contains two or more Chinese characters/Chinese words.

Index Table: An Index Table is composed of tuple words and their related information extracted from the database, its schema is ***TupleTable***(*wid, word, size, DBValue*), where *wid* is the primary key. For a tuple $t\in R$ and an attribute $A \in\{A_1, A_2, \ldots, A_m\}$, a tuple word $z \in t[A]$ will be stored in the text attribute *word*. *DBValue* is a text attribute with form "*tid,cid,dl,tf,df;…; tid,cid,dl,tf,df;*", where *tid* is the identifier of *t*, *cid* is the identifier of attribute (or column) *A*, $dl = |t[A]|$ is the length of $t[A]$ which contains the tuple word z , tf is the number of occurrences of z in the cell with *tid* and *cid*, df is the number of cells that have the same *cid* and contain z, equivalently, $df = |\{t_k;\ z\in t_k[A]\}|$. The attribute *size* is the total number of cells that contain z, that is, the number of semicolons (";") in *DBvalue.*

Query: A query Q is a set of Chinese query words with or without some semicolons, $Q = \{q_1q_2\ldots q_i[;\ldots;q_1q_2\ldots q_k]\}$, where each q_h ($1\leq h \leq \max(i,\ldots, k)$) is a Chinese word.

Simple query: A simple query Q^s is the query Q without semicolon. Q^s has the form of $\{q_1q_2\ldots q_i\}$.

Complex query: A complex query Q^c is composed of two or more Q^ss. $Q^c = \{q_1q_2\ldots q_i;\ldots;q_1q_2\ldots q_k\} = \{Q^s_1;\ \ldots;\ Q^s_p\}$.

Length: The length of a query Q^c is the total number of query words contained in the query, $|Q^c|=|Q^s_1|+|Q^s_2|+\ldots+|Q^s_p|$. The length of $t[A_i]$ is the number of tuple words in $t[A_i]$.

III. Index and Ranking strategy

The index techniques and ranking strategies play important roles in processing of top-*N* queries (or ranking queries) [6, 7], and are described in this section.

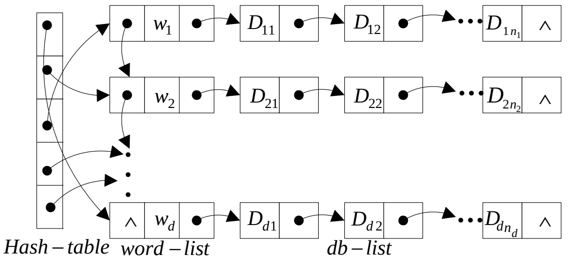

Figure 1. Structure of cwIndex.

A. Creation of cwIndex

The relation ***TupleTable*** in Section II is employed to store the information of our cwIndex (stands for *C*hinese *w*ord index) used in our application. The structure of cwIndex is shown in Figure 1, which consists of one hash table, one word-list, and *d* db-lists where *d* is the size of word-list. The node *w* in the list *word-list*(*w*, *pdbvalue*) corresponds to ***TupleTable****.word*. The *pdbvalue* is a pointer that points to a list *db-list*(*D*, *pdblist*), where *D* in *db-list* corresponds to the substring which is split by semicolon ";" in ***TupleTable****.DBvalue*. The *Hash-table* is used to lookup *word-list.w* quickly. The process of creating our cwIndex needs: (1) Normalization of tuples in *R*. (2) For each $z\in t[A]$, extract its related information in tuple *t* to create list *db-list.* (3) Create list *word-list.*

To evaluate a query, we can use two storing strategies. Strategy-1, the entire cwIndex is in main memory. Strategy-2 will store db-lists in fixed disk and only load the hash table and word-list into main memory. Strategy-2 will be used in our experiments.

B. Ranking Strategy

For a query Q, to rank its answers, we define the similarities between the query and tuples based on the ranking model in IR described in [9]. For the query Q and a tuple t, our ranking strategy will be given according to the similarities below:

$$sim(Q^c,t) = \sum_{Q^s\in Q^c} sim(Q^s,t) \qquad (1)$$

$$sim(Q^s,t) = \max_{A_i\in\{A_1,\ldots,A_m\}} (sim(Q^s,t[A_i]))*y \qquad (2)$$

$$sim(Q^s,t[A_i]) = \sum_{q\in Q^s, z\in t[A_i]} weight(q,Q^s)*weight(z,t[A_i]) \qquad (3)$$

$$weight(z,t[A_i]) = \frac{1+\ln(1+\ln(tf))}{(1-s)+s*\frac{dl}{avdl}}*\ln\frac{n+1}{df} \qquad (4)$$

$$y = \max(c)/l_a \qquad (5)$$

Equation (1) shows that the similarity between Q^c and t is the sum of similarities between t and Q^s for all $Q^s\in Q^c$. The similarity between Q^s and t is the product of parameter y (which we will discuss in the following Section IV.B) and the maximum value of similarities between Q^s and $t[A_i]\in t$ (i=1,2,…,m), as shown in (2). Equation (3) calculates the similarity between Q^s and $t[A_i]$ by the inner product function, where $weight(q, Q^s)$ is the appearance frequency of query word q in query Q^s. Component $weight(z, t[A_i])$ computes a weight for each tuple word z in the text attribute $t[A_i]$. Equation (4) being one of the most widely used weighting methods in IR [9], we employ it to compute the weight of a tuple word z in $t[A_i]$, where s is a constant and usually set to 0.2, $avdl$ is the average length of $t[A_i]$($t\in R$), and $n = |R|$ is the total number of tuples in R.

IV. Evaluation of Chinese keyword Query

For a given query $Q = Q^s$ or $Q = Q^c = \{Q^s_1;\ Q^s_2;\ldots;Q^s_p\}$, without loss of generality, let $Q = Q^c$. Firstly, obtain its candidate tuples using cwIndex. Secondly, based on our ranking strategy, a ranking algorithm is defined such that the more relevant answers for the query are ranked higher. Thirdly, a refined method with phrase-based ranking is introduced to find the high-ranking desired answers from the candidate tuples. Finally, output ranked top-*N* results with similarities by friendly user interface. These methods guarantee the efficiency and effectiveness of *Q*.

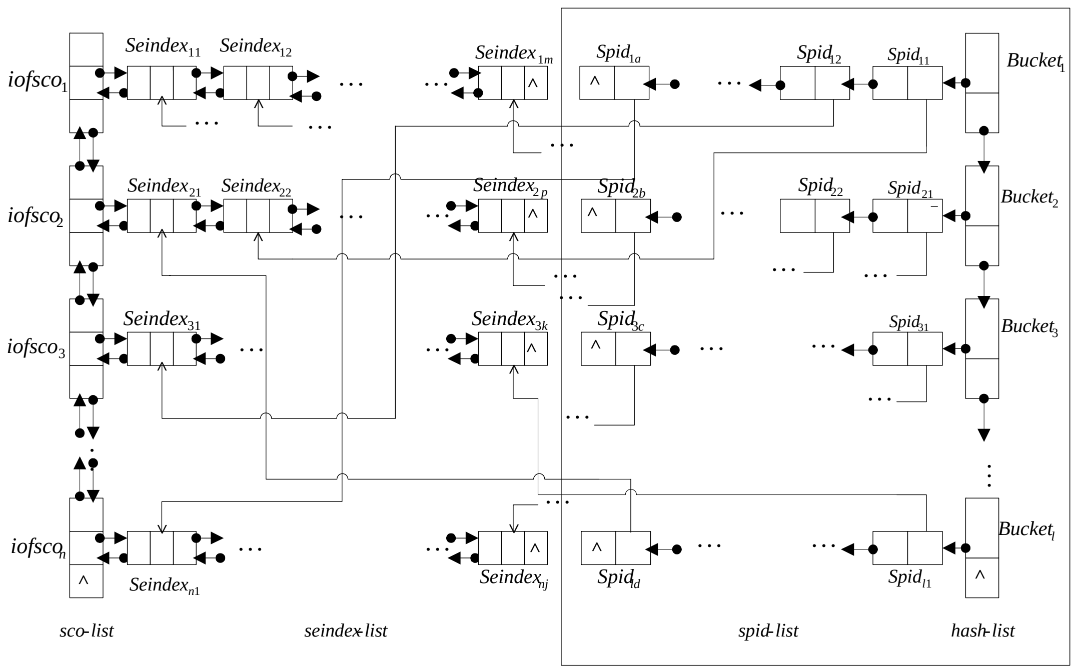

Figure 2. Lists of seraching and randing of candicate tupels.

A. Seraching and Randing of Candicate Tupels

To obtain and rank candidate tuples of *Q* is based on the hypothesis: the answers of *Q* are the tuples that contains the query words as many as possible. Thus, we design the procedure that consists of linked lists as shown in Figure 2. The nodes of lists are generated dynamically in the processing of the query, meanwhile the lists are built and candidate tuples are ranked. Figure 2 includes two parts: R-part is the right part, i.e., the part in the pane with real line; L-part is the left part outside the pane.

(1) R-part contains two layers of lists, *hash-list*→{*spid-list*}, which are used to find the candidate tuples. For hash list *hash-list*, the structure of its bucket is *Bucket*(*B*, *pspid*, *pnext*), *B* is the value of hash function at a tuple identifier (*tid*), and *B* := int(*tid*) mod 100 in our experiments. In order to speed up matching, we sort the buckets of *hash-list* decreasingly according to their values {*Bucket.B*}. The pointer *pspid* points to the first node of *spid-list*, and the pointer *pnext* points to the next *Bucket* of *hash-list*. For list *spid-list*, its node is indicated by *Spid* with structure {*tid*, *pseindex, pspid*} where *tid* is tuple identifier, pointer *pspid* points to the next node of list *spid-list*, and pointer *pseindex* points to the corresponding node of list *seindex-list* in L-Part. There is a one-to-one correspondence of nodes of *spid-list* to nodes of *seindex-list* in Figure 2.

(2) L-Part includes three layers of lists, that is, *sco-list*→{*seindex-list*→{*detail-list*}} (notice that the third layer list *detail-list* is not drawn in Figure 2). L-Part is applied to rank the candidate tuples. the structure of node *iofsco* is {*sco, pseindex, pnext, pprior*}, and The nodes {*iofsco*} of list *sco-list* are sorted decreasingly by the values of *iofsco.sco*. Since a tuple word *z* may appear in multiple tuples, the value of *sco* is the number of query words that belong to a tuple in underlying database. The nodes with the same value of *sco* are inserted into list *seindex-list* pointed by pointer *pseindex*. For double-linked list *seindex-list*, its nodes are sorted decreasingly by the values of *Seindex.ssco*, the node of *seindex-list* is denoted by *Seindex* with the structure of {*tid, sco, ssco, pdetail, pnext, pprior* }, where *ssco* is used to store the similarity between a query and a tuple computed by a procedure, and pointer *pdetail* points to list *detail-list* not drawn in Figure 2. The node of *detail-list* is indicated by *Detail* with structure {*zid, cid, dl, tf, df, size, pnext*}, where *zid* is the pointer of the array that stores the word *z*, the array will store the codes of GB2312-80 for Chinese words with tuple-identifier *Seindex.tid* and column-identifier *Detail.cid*, *cid* is the column-identifier (or called attribute-identifier) that the Chinese word belongs to the value of tuple *tid* at the attribute *cid*, *size* is the number of tuples in the underlying database.

B. Refined Method with Phrase-based Ranking

Equation (5) in the Section III.B, $y = max(c)/l_a$, adjusts the similarity between a query Q^s and a tuple *t* in (2), where *c* is the longest length of the matching substring between Q^s and $t[A_i]$ ($A_i \in \{ A_1, \ldots, A_m \}$), and l_a is the length of $t[A_a]$ that corresponds with the length *c*. For instance, $Q^s = \{\ldots q_i q_{i+1} \ldots q_j \ldots\}$, $t[A_i] = \{\ldots z_m z_{m+1} \ldots z_n \ldots\}$ ($i<j$, $m<n$), if $q_i = z_m$, $q_{i+1} = z_{m+1}$, …, $q_j = z_n$ and the matching substring "$q_i q_{i+1} \ldots q_j$ " (= "$z_m z_{m+1} \ldots z_n$" in $t[A_i]$) is the longest one, then $c = | q_i q_{i+1} \ldots q_j | = j-i+1 (= | z_m z_{m+1} \ldots z_n |)$.

V. EXPERIMENTS

Our experiments are carried out using Microsoft's SQL Server 2000 and VC++6.0 on a PC with Windows XP, Intel(R) Core2 Duo 2.0 GHz CPU, and 2.0GB memory. In addition, ODBC and ODBC API functions are used in our implementations. The real dataset contains 500 WAV or MP3 sound files that come from a foley library as described in Section II.

The parameters that we change in the experiments are the number of query words and the number *N* of results requested in top-*N* queries. The workload contains 100 queries, and is used to measure the efficiency and effectiveness of our method. The number of query words is between 2 and 16, and *N* will be 1, 3, 10, 20, or 50 for top-*N* queries.

In the following figures, the suffixes "1", "3", . . . , and "50" of legends indicate the top-1, top-3, . . . , and top-50 queries, respectively.

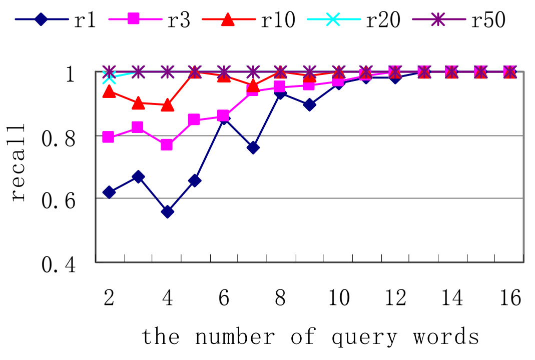

Figure 3. Recalls of queries.

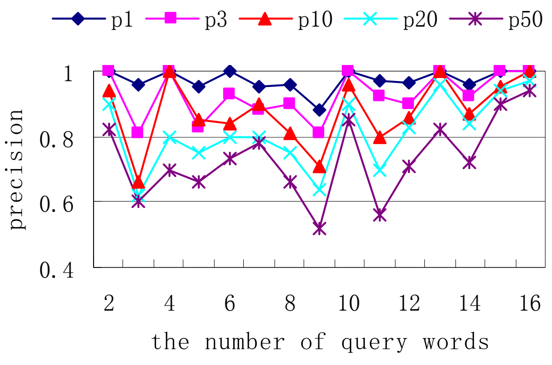

Figure 4. Precisions of queries.

For all queries, the average elapsed times including *Index-time* and *Result-time* are not larger than hundreds of milliseconds. Our method is efficient due to the efficiency of RDBMS.

Figure 3 and Figure 4 show recalls and precisions of top-*N* results of keyword queries respectively. In Figure 3, with the increase of *N*, recall will become larger. The reason is that the total number of results desired in the database is constant, while the number of matching tuples in the top-*N* results will increase as *N* becomes larger. The precision in Figure 4 is calculated by the actual number of results returned rather than the value of *N* for each query. With the increase of *N*, precisions will decrease. Usually, the more concrete queries are, the more accurate answers will be obtained, and the number of results will be less than *N* for larger *N*'s (say *N* = 50) in our experiments.

VI. CONCLUSIONS

For a framework of managing sound files by using a relational database management system, we proposed a new method based on Chinese keyword search to select the WAV or MP3 files in audio post-production. We listen to each file and label it with Chinese words, and then classify and store the files in a relational database system. The basic idea of techniques of Chinese keyword search is to create an index and build a ranking strategy. Thus, we can use Chinese keyword queries to find audio files. For a query, we employ the index to match query words and tuple words quickly, and use the ranking strategy to compute similarities between the query and candidate tuples, and finally get top-*N* results of audio files ranked by similarities. The experimental results show that our method is efficient and effective.

ACKNOWLEDGMENT

This work is supported in part by NSFC (61170039) and the NSF of Hebei Province (F2012201006).

REFERENCES

[1] G. Lu, "Indexing and Retrieval of Audio: A Survey," Multimedia Tools and Applications, Vol.15, No.3, 2001, pp. 269–290.

[2] D. Mitrovic, M. Zeppelzauer, and C. Breiteneder, "Features for Content-Based Audio Retrieval," Advances in Computers, vol.78, 2010, pp. 71-150, doi:10.1016/S0065-2458(10)78003-7

[3] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Database," Proceedings of the 18th International Conference on Data Engineering, San Jose, 26 February -1 March 2002, pp. 5-16.

[4] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases," 26th ACM SIGMOD/PODS International Conference on Management of Data/Principles of Database Systems, Chicago, 27-29 June 2006, pp. 563-574.

[5] J. Yu, L. Qin and L. Chang, "Keyword Search in Relational Databases: A Survey," IEEE Data Eng. Bull. Special Issue on Keyword Search, Vol. 33 No.1, 2010, pp. 67–78.

[6] L. Zhu, Y. Zhu, and Q. Ma, "Chinese Keyword Search over Relational Databases," 2010 Second World Congress on Software Engineering (WCSE 10), Wuhan, 19-20 December 2010, pp. 217-220, doi: 10.1109/WCSE.2010.81.

[7] L. Zhu, Q. Ma, C. Liu, G. Mao and W. Yang, "Semantic-distance based evaluation of ranking queries over relational databases," Journal of Intelligent Information Systems, Vol. 35, No. 3, 2010, pp. 415-445, doi:10.1007/s10844-009-0116-5.

[8] Dublin Core Metadata Element Set, Version 1.1, http://dublincore.org/documents/dces/

[9] A. Singhal, "Modern Information Retrieval: A Brief Overview," IEEE Data Eng, Vol. 24, No. 4, 2001, pp. 35- 43.