一个基于压缩后缀数组的乐纹索引算法

刘学政 史有群 罗 辛 陶 然
(东华大学计算机科学与技术学院 上海 201620)

摘 要 在基于乐纹的音乐检索系统中,提取的乐纹的多少决定了检索结果的匹配度,这就造成了数据库大小与检索匹配度不能兼顾的矛盾。提出使用压缩后缀数组来压缩乐纹索引的方法,解决全文索引时索引空间过大的问题。主要利用有序乐纹数据中较高位特征出现重复的概率大的特点,使用游程编码对乐纹序列进行无损压缩。实验结果表明,该方法在包含 2000 首歌曲的数据库中仅需要使用原来 80% 的乐纹数据空间,在包含 12000 首歌曲的数据库中只需要使用原来 30% 的乐纹数据空间。与传统的后缀数组索引方法相比,该方法需要的索引存储空间仅为原来的 60%。

关键词 乐纹,压缩后缀数组,索引压缩,游程编码,垂直编码
中图法分类号 TP399 文献标识码 A

Index Algorithm for Audio Fingerprints Based on Compressed Suffix Array

LIU Xue-zheng SHI You-qun LUO Xin TAO Ran
(School of Computer Science and Technology, Donghua University, Shanghai 201620, China)

Abstract In music retrieval methods based on audio fingerprints, a large database is required in order to compare with the fingerprints leading to low retrieval efficiency. In this paper, we proposed a method for index compression using a compressed suffix array, in order to overcome the disadvantage of large memory cost with full-text indexing. The proposed method compresses data sequences lossless by run length encoding, mainly taking advantage of the fact that the repetitivecharacters occur frequently in higher bits of the sorted audio fingerprint data. The experimental results show that the proposed method, compared with the conventional method, only needs 30% of the space of an audio fingerprints database for a music database consisting of 12000 songs, and around 80% of the index space for a database of 2000 songs. Moreover, theentire space cost is reduced to around 60%, compared with the method based on the suffix array.

Keywords Audio fingerprint, Compressed suffix array, Index compression, Run length encoding, Vertical code

1 概述

随着互联网的迅猛发展、音频压缩技术的进步以及大容量存储器的出现,人们从互联网上获取海量音频信息变得越来越容易。面对大量的音乐信息,利用手工方式在这些音乐数据中查找特定歌曲已经变得费时费力。传统的通过文字匹配来查找歌曲的方式已经远不能满足用户对此类资源的检索要求,除了通过传统的利用关键字检索歌名信息获得所需的歌曲外,人们更希望能够通过音乐本身的特征进行检索,以便快速、准确、高效地查找音乐信息。近年来,基于内容的音乐检索 CBMR(Content-based Music Retrieval)发展成为网络环境下处理多媒体海量数据的一项重要课题,与图像检索、视频检索并列成为基于内容的多媒体信息检索研究的热点[1]。

基于内容的音频信息检索的研究工作开始于 20 世纪 90 年代,最初以哼唱检索系统的研究与开发为主[2]。近年,将音乐的特征值用二进制数来表示的乐纹(Audio Fingerprint)高速检索技术成为音频信息检索引擎的关键技术[3,4]。大部分音乐检索系统使用的是由 Haitsma 和 Kalker 提出的乐纹[5]来进行检索。一段未知音乐片段的信息可以通过乐纹以及音乐信息数据库推断得到。使用乐纹而不是音频数据本身进行检索,是因为乐纹具有有效信息量集中、抗噪性强以及乐纹数据库更小且更利于高效搜索等优点[6]。另外,乐纹不仅能用于检索歌曲,还能用于歌曲的版权保护,如检测歌曲的抄袭以及网上侵权歌曲的分布等。

对于从歌曲特定部分开始的音乐检索而言,每首歌曲只要提取几个或十几个乐纹即可达到目的,但是一个好的音乐检索系统应该能够从一首歌曲的任一部分开始检索并快速得到结果。为了做到这一点,就要将每首歌曲分成尽可能小的部分建立索引。在这种情况下通常使用全文索引,如哈希表[7]、树形结构[8]以及后缀数组技术[9,10]等,它们都可以检索数据库中所有歌曲的任意部分的子串。但全文索引需要的空间过于庞大,在实际应用中很难直接应用于音乐搜索引擎。

基于这些问题,本文提出了一种结合游程编码和垂直编码来压缩后缀数组的方法,其通过压缩乐纹数据库来解决乐纹数据库占用空间过大的问题。通过对比实验表明,对包含 12000 首歌曲的数据库只需要传统方法 30% 的空间,并且随着数据量的增加,压缩的空间也会明显增加;此外,与传统的后缀数组索引方法相比,本文方法需要的索引存储空间是传统方法的 60%。由此,验证了本文方法在乐纹数据压缩方面的有效性及其对检索速度的提高。

刘学政(1989—),男,硕士生,主要研究方向为人工智能与云计算,E-mail:shuidao0001@126.com。

2 乐纹检索系统

乐纹是指基于内容的，可以代表一段音乐重要声学特征的紧致数字签名。它利用音频信号的声音和知觉特征将声音信号转换成一个相对紧凑的表示。对于主要用于身份验证和数字签名的信息摘要（如 MD5），初始输入的细微差别将会产生两个完全不同的哈希值，这就意味着从原始音频信号和它的一个有损镜像中得到的两个哈希值是完全不同的，而这正是在检索“有损”音乐时查询性能急剧下降的原因。但是，在乐纹中，相似的输入能够得到相似的哈希值。

基于乐纹的音频检索系统主要由两部分组成：乐纹库构建与相似性查询。

建库过程包括特征计算、乐纹提取以及乐纹库的构建。由于查询位置不确定，需要考虑各种不同的起始位置，因此针对数据库中每一首乐曲的所有帧都要逐一提取乐纹块。为了提高查询速度，乐纹的存储采用哈希表结构。对从音乐库的每首歌曲中提取的乐纹使用压缩后缀数组处理后按照哈希表的数据结构保存到乐纹库里。

查询时首先求出样本片段的匹配候选者集，即音乐库里每首歌曲的乐纹与查询样本片段的乐纹的交集，交集不为空集的所有歌曲都可以成为“匹配候选者”，然后利用度量算法计算出每个候选者与样本片段之间的“距离”。经过排序后，选择距离最小的前 N 首歌曲作为检索结果[11]。

整个乐纹检索过程如图 1 所示。

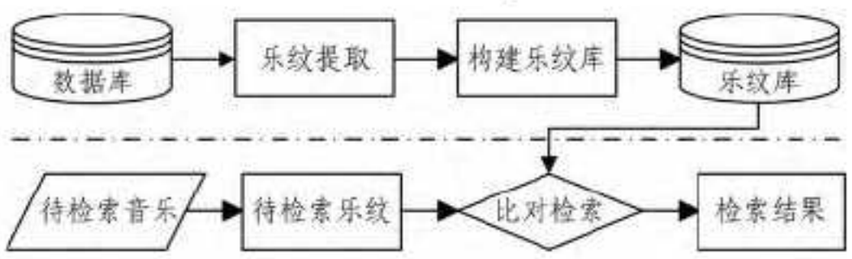

图 1 基于乐纹的音频检索系统

3 压缩后缀数组算法

在计算机中，后缀是指从一个字符串的相应位置出发直到字符串结尾的子串，如：对于字符串 $S=cdba$，那么它共有 4 个后缀，$S_0=cdba$，$S_1=dba$，$S_2=ba$，$S_3=a$。后缀排序就是对字符串的所有后缀按照字典序进行排序，上例的后缀排序结果为：S_3，S_2，S_0，S_1。

在构造索引文件时，要考虑检索时如何更快捷地找出所需的索引数据，这就涉及到后缀排序问题，在实际应用的时候，原始的音频乐纹数据是非常巨大的[12]。本文给出一种压缩算法来减小索引，减少检索时间。

3.1 索引压缩

由于索引和后缀数组有着相同的结构，我们使用压缩后缀数组来压缩索引。首先对从数据库中提取的子乐纹进行有序压缩，保存压缩的数据。

例如，给定一个子乐纹 FP，令它的分隔间隔 $M_1=8$，FP 是有序的，则它的一个分割块 $FP0'$ 可以表示为：

$$FP0'=\begin{bmatrix}00&00&00&01\\00&00&00&01\\00&00&01&01\\00&00&01&11\\00&01&01&11\\00&01&01&11\\00&01&11&11\\00&01&11&11\end{bmatrix} \tag{1}$$

取式(1)的最低位（矩阵的第一列）的值，可以得到：

$$FP0'(0,0),FP0'(1,0),\cdots,FP0'(7,0)=$$
$$00,00,00,00,00,00,00,00 \tag{2}$$

由于重复字符“00”共出现了 8 次，因此对式(2)进行游程编码，可以得到式(3)：

$$00,00,08 \tag{3}$$

这里“00”表示原始字符的值，将它重复两次主要是为了防止在处理和存储过程中出现错误或丢失导致读取处理时后面的值全部出错，“08”代表字符“00”重复了 8 次。

同样地，下一位（第二列）的数据如(4)所示，进行游程编码后得到的结果如式(5)：

$$FP0'(0,1),FP0'(1,1),\cdots,FP0'(7,1)=$$
$$00,00,00,00,01,01,01,01 \tag{4}$$
$$00,00,04,01,01,04 \tag{5}$$

剩余的两位如式(6)和式(8)所示，它们分别进行游程编码后的结果如式(7)和式(9)所示：

$$FP0'(0,2),FP0'(1,2),\cdots,FP0'(7,2)=$$
$$00,00,01,01,01,01,11,11 \tag{6}$$
$$00,00,02,01,01,04,11,11,02 \tag{7}$$
$$FP0'(0,4),FP0'(1,4),\cdots,FP0'(7,4)=$$
$$01,01,01,11,11,11,11,11 \tag{8}$$
$$01,01,03,11,11,05 \tag{9}$$

这样，$FP0'$ 经过游程编码后变成 $FP0''$，如式(10)所示：

$$FP0''=00,00,08,00,00,04,01,01,04,00,00,02,01,01,$$
$$04,11,11,02,01,01,03,11,11,05 \tag{10}$$

数据库越大，游程编码的长度（一系列的重复字符）越长，压缩效率也随之提高。子乐纹 FP 经过上述过程被压缩为 $FP'=FP0',\cdots,FPn/M_1'$，最终被保留为子乐纹数据库。

在我们的系统中，经过上述处理之后，就可以使用 $FP'=FP'[0],\cdots,FP'[n](FP'[i]=FP'[SA[i]])$ 来代替 FP，并且将 FP' 以 M_1 的大小划分成块。即，从乐纹块 $FPk'=FP'[M_1\times k],FP'[M_1\times k+1],\cdots,FP'[M_1\times(k+1)-1]$ 中可以推导出 $FP'=FP0',\cdots,FPn/M_1'$。这是因为在搜索的时候压缩数据需要能够被瞬间恢复。在这里使用 $M_1=8$ 是因为这是实验的最佳值，由于 FP' 经过排序之后在上侧的元素值很可能是相同的。

在上面的处理过程中，我们针对的是经过排序之后的数据，但是经过排序之后数据库中乐纹的顺序会被打乱，而在检索的时候需要对数据进行恢复，因此我们使用 $\psi[i]$ 来保存原有的顺序信息，计算方法如(11)所示：

$$\psi[i]=\begin{cases}SA^{-1}[SA[i]+1], & \text{if } SA[i]\neq n\\ 0, & \text{if } SA[i]=n\end{cases} \tag{11}$$

引入 $\psi[i]$ 后，与数据库相同的序列 $FP[SA[i]],FP[SA[i]+1],\cdots,FP[SA[i]+j]$ 可以用 $FP'[i],FP'[\psi[i]],\cdots,FP'[\psi^j[i]]$ 来表示，其中 $[\psi^j[i]]$ 表示 $i=\psi[i]$ 中第 j 个重复的字符。因为 $\psi[i]$ 是局部单调递增的，所以它能够被压缩。$\psi[i]$ 局部单调递增的原因是，如果存在重复值，那么排序的顺序是由它下一个以及后面的数值决定的。

垂直编码，一种用较小的大小表示较小值的编码方式，在这里用于表示压缩 $\psi[i]$ 时的 ψ 的差异值。$\psi[i]$ 的差异值 $d[i]$ 的计算如式(12)，如果 $d[i]<0$，它总是随着 n 的增长而单调递增。

$$d[i]=\begin{cases}\psi[i]-\psi[i-1], & \text{if } i\neq 0\\ 0, & \text{if } i=0\end{cases} \tag{12}$$

把 $\psi[i]$ 以 M_2 长的间隔划分成块，即使用块 $\psi_0,\cdots,\psi_{n/M_2}$ 代替 $\psi_k=\psi[M_2\times k],\cdots,\psi[M_2\times(k+1)-1]$。第一块的数据 $\psi[M_2\times k]$ 被存储在 ψ_k' 作为样本。这样做是因为在进行搜索的时候，与 FP' 一样，数据需要能够被迅速还原。

同样地，把 $d[i]$ 也以 M_2 长的间隔划分成块，即使用块 $d_0,\cdots,d_{n/M2}$ 代替 $d_k=d[M_2\times k],\cdots,d[M_2\times(k+1)-1]$。首先，从 $d[i]$ 的最大值中获得位掩码 $MSB[k]$，用它表示块中的数据。此外，将 $d[M_2\times k+p]$ 的二进制表示的第 q 位存储在 $V_k[q]$ 的第 p 位。也就是说，V_k 的大小（每个 V_k 的 q 的最大值）等于 $MSB[k]$。这样，作为 $8M_2$ 的倍数，$V_k[q]$ 可以以字节为单位进行处理。

例如，假定 $M_2=8$，十进制 $d_0=1,0,1,2,3,2,1,0$，因为 d_0 的最大值为 3，所以 d_0 中的数据都可以用两位二进制来表示。把 d_0 转换为两位二进制数，则 $d_0=01,00,01,10,11,10,01,00$，然后由 $V_0=V_0[0],V_0[1]$ 可以推出，$V_0[0]=0101\ 0101$，$V_0[1]=0011\ 1000$。

在这里，索引和 $\psi[i]$ 的区别就是它在被压缩的同时能够通过上述步骤保存 $V=V_1,\cdots,V_{n/M_2}$ 和 $MSB=MSB[1],\cdots,MSB[n/M_2]$ 的信息。

3.2 索引恢复

在进行检索的时候需要对压缩后的索引 ψ 进行恢复。恢复通过 ψ'（ψ 的一个样本）来进行。如上所述，ψ 记录了排序后的子乐纹 SFP 的顺序信息，它可以通过 ψ' 和差异值 d 表示如下：

$$\psi[i]=\psi'[i']+j'+\sum_{k}^{j'}d[M_2\times i'+k] \tag{13}$$

式中，$i'=i/M_2$，$j'=i \bmod M_2$。当 $\psi[i]>n$ 时，$\psi[i]=\psi[i] \bmod n$。作为 $d[i']$ 从 1 到 j' 的和，$\sum_{k=1}^{j'}d[M_2\times i'+k]$ 使用 $MASK=\{(1<<j'|((1<<j')-1)-1\}$ 来表示，这样它就变成 $\sum_{k=0}^{MSB[i']}\{popcount(V'[k]\&MASK)<<k\}$，其中的 $MASK$ 表示 $MASK$ 从 1 到 j'，$popcount(x)$ 表示在数据 x 中位值为 1 的位的数量，“|”表示或运算，“&”表示与，“<<”表示左移。经过上述过程，我们可以由 ψ,V 和 MSB 最终还原得到 $\psi[i]$。

4 基于压缩后缀数组的检索系统

基于后缀数组的音乐检索系统使用二分查找的方法，它的查找对象是从数据库的所有歌曲中提取的子乐纹 FP 和存储着长度为 3 的有序子乐纹序列的后缀数组 SA。本文的方法是基于压缩后缀数组的，它直接在 SFP 上进行二分查找，SFP 是指已经提前排序好的乐纹。

基于压缩后缀数组的搜索方法是要在 $FP'[i],\cdots,FP'[\psi^j[i]]$ 中查找与从待查音乐中提取的子乐纹块之间误码率最小的子乐纹块。这个过程可以分为如下 3 步：

对于像式(14)中表示的从查询 Q 中导出的的长度为 3 的子乐纹列，使用从数据库中导出的长度为 3 的子乐纹序列（如式(15)所示）对它进行二分查找。使用误码率来评估它们的相似度。

$$Q_{i,i+2}=Q[i],Q[i+1],Q[i+2] \tag{14}$$

$$SFP_{j,j+2}=SFP[j],SFP[\psi[j],SFP[\psi^2[j]] \tag{15}$$

对于上面探讨的 $SFP_{j,j+2}$，计算子块乐纹的相似程度。因为乐纹块的长度是 128，这里就是要计算 $Q_{i,i+127}$ 和 $SFP_{j,j+127}$ 之间的误码率。经过计算，如果误码率小于阈值，那么包含 $SFP_{j,j+127}$ 这一块的音乐将被选为候选。

将候选数据按照误码率排序，然后输出较低误码率的音乐作为结果。

5 实验和结果

设计了对比实验来分别评估基于后缀数组和基于压缩后缀数组的两种方法在数据压缩和检索速度方面的有效性。

基于后缀数组的检索系统，采用的是文献[7]中给出的基于后缀数组的海明空间搜索。海明空间搜索有一个类似于后缀数组的子乐纹序列的索引，以保证能够在子乐纹数据库中高速搜索。该方法通过复用子乐纹序列(SSF)查询而不是直接增大数据库的方式，极大地节省了搜索空间。

SSF 搜索方式如图 2 所示。

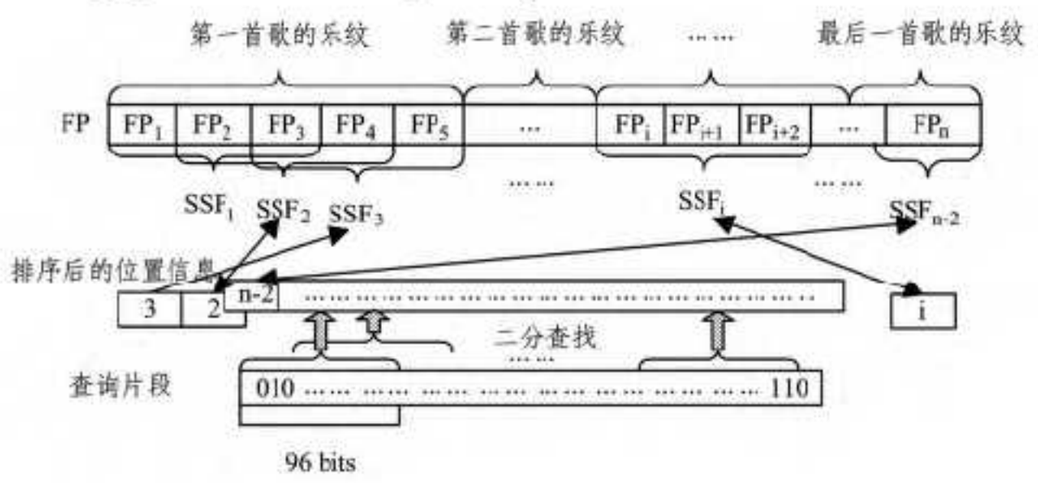

图 2 SSF 检索方法

在对数组 S 进行二分查找的时候，可以通过检查 S 的搜索块的邻居位找到大部分相似的 SSF。

与基于后缀数组的方法一样，也使用 Haitsma-Kalker 算法来提取乐纹，然而它们有几点不同：1）每个帧的长度为 1.024s；2）32 毫秒帧移动；3）改进的 Hamming 窗口；4）子乐纹块的长度是 128。

1）实验条件

①音乐数据。数据库包含从 CD 或者因特网上获得的 12000 首 MP3 格式音乐。这些音乐分为许多流派，比如流行、古典、乡村音乐等等。在我们实验中，从数据库中选择了 3 套音乐，分别包含 2000，4000 和 8000 首歌曲，然后创建各组的索引，以便获得大小和实践的变化比例。

②压缩设置。有序数据 ψ 的顺序和差异值 d 的分段间隔都被定为 M_2，大小为 32，这是因为子乐纹的长度是 32。同样地，将有序子乐纹 SF 的分段间隔 M_1 也设置为 32，SA 的样本的采样间隔 $M3$ 也设成同样大小，以获得歌曲编号。

2）数据大小

将本文提出的基于压缩后缀数组的索引压缩方法和传统基于后缀数组的方法产生的数据进行了比较，分别对比了子乐纹的大小和索引数据的大小。

①子乐纹大小。文中提出的方法首先对乐纹 FP 进行排序，然后对排序后的 FP 使用 RLE 进行压缩编码。两种方法得到的子乐纹大小如表 1 所列。

表 1 子乐纹大小

歌曲(首)	传统方法(MB)	本文方法(MB)	压缩率(%)
2000	58.4	24.0	41.09
4000	125.5	46.7	37.21
8000	258.3	87.5	33.88
12000	391.2	122.6	31.34

表 1 中的压缩率是由式(16)决定的。随着歌曲的数量变多，压缩率减小，压缩效率提高。这是因为子乐纹增多导致游

（下转第 488 页）

[12] 钟志文. 基于平行坐标的关联规则挖掘技术可视化研究与实现[J]. 常州工学院学报,2012,25(2):29-33
[13] 郭晓波,赵书良,赵娇娇,等. 基于概念格的多值属性关联规则可视化[J]. 计算机应用,2013(8):2198-2203
[14] 吴天真. 基于修补项的关联规则可视化挖掘方法的研究[D]. 武汉:华中师范大学,2009
[15] 易先卉. 关联规则可视化技术的研究及实现[D]. 长沙:湖南大学,2008
[16] 石大文. 基于 LDA 模型的 BBS 话题演化[J]. 工业控制计算机,2012,25(5):82-84
[17] Chen Y, Cheng X Q, Yang S. Finding high quality threads in Web forums[J]. Journal of Software,2011,22(8):1785-1804
[18] 孟海东,林志举,徐贯东. 可视化数据挖掘工具的设计与实现[J]. 计算机与现代化,2011(6):132-135
[19] 张浩,郭灿. 数据可视化技术应用趋势与分类研究[J]. 软件导刊,2012,11(5):169-172
[20] 王华金,蔡虔. 数据挖掘可视化技术综述[J]. 科技广场,2009(1):235-237

(上接第 464 页)

程(一系列的重复字符)变长。

$$\text{压缩率}=\frac{\text{使用本文方法得到的数据大小}}{\text{使用传统方法得到的数据大小}}\times 100\% \quad (16)$$

②索引大小。为了获得数据 SFP 的顺序,本文方法通过垂直编码对 ψ 进行存储。表 2 列出 SA 和 ψ 的大小。SA 的数据 ψ 的大小会随着数据库中歌曲数量的增长而增长,也就是说,随着数据库的增大,压缩效率会变低。如果因为提高音乐的数量而增加了子乐纹的种类,那么相邻的数据也不一定是相同的,即便它存在于排序好的数据中。换句话说,单调增加的部分减少了,这导致了压缩效率的恶化。

表 2 索引大小

歌曲(首)	传统方法(MB)	本文方法(MB)	压缩率(%)
2000	57.3	46.8	81.68
4000	123.3	103.6	84.02
8000	254.1	217.5	85.60
12000	387.0	336.1	86.85

③总数据大小。总数据的大小如表 3 所列。总体来说,对于数据库中的每首歌,压缩率在 60% 左右。随着歌曲数量的增加,压缩率也略有提升,这归功于子乐纹压缩率的高度。

表 3 总数据大小

歌曲(首)	传统方法(MB)	本文方法(MB)	压缩率(%)
2000	115.7	70.8	61.19
4000	248.8	150.3	60.41
8000	512.4	305.0	59.52
12000	778.2	458.7	58.94

3)搜索时间

在检索时,用于查询的歌曲和数据库的是一样的,并且和原始歌曲有相同的长度。在本节中,使用每 10 秒的查询结果。也就是说,对于一个有 S_q 秒的查询乐曲,如果搜索总共花费了 S_s 秒,那么每 10 秒的查询时间可以表示为 $S_s/S_q\times 10$。

表 4 列出了所有音乐数据的集合中每首歌曲的平均搜索时间。减速因子(SLF)指的是采用本文方法所消耗的时间与传统方法相比的倍数,即 SLF 越小,本文方法越快。SLF 的计算如式(17)所示。

$$SLF=\frac{\text{使用本文方法的搜索时间}}{\text{使用传统方法的搜索时间}} \quad (17)$$

表 4 平均搜索时间

歌曲(首)	传统方法(ms)	本文方法(ms)	SLF
2000	1.1	12.9	11.7
4000	2.3	18.5	8.0
8000	3.8	23.1	6.1
12000	5.0	25.5	5.1

表 4 还表明本文方法搜索会花费更多时间,这主要是数据结构的关系。然而,随着数据库中歌曲的增多,减速因子呈下降趋势。

结束语 本文提出一种基于游程编码和垂直编码来压缩乐纹后缀数组的索引压缩方法。实验结果表明,该方法能够有效地节省大量的乐纹数据库空间。尽管该方法在检索时需要花费更多的查询时间,但查询时间的倍数会随着数据库中歌曲数量的增加而呈下降趋势。因此,我们下一步工作将包括在海量数据中如何提高检索速度以及数据库音乐检索的具体应用研究等。

参 考 文 献

[1] Casey M A, Veltkamp R, Goto M, et al. Content-based music information retrieval: current directions and future challenges[J]. Proceedings of the IEEE, 2008, 96(4): 668-696
[2] Xiao Q, Xin Luo, Saito N, et al. Index Compression for Audio Fingerprinting Systems Based on Compressed Suffix Array [J]. International Journal of Information and Education Technology, 2013, 3(4): 455-460
[3] Bellettini C, Mazzin G. A Framework for Robust Audio Fingerprinting [J]. Journal of Communications, 2010, 5(5): 409-424
[4] Xiao Qing-mei, Saito N, Luo Xin, et al. Index Compression for Audio Fingerprinting Systems Based on Compressed Suffix Array [J]. International Journal of Information and Education Technology, 2013, 3(4): 455-460
[5] Haitsma J, Kalker T. Ahighly robust audio fingerprinting system [C]// Proc. 3rd International Conference on Music Information Retrieval. 2002
[6] 李伟,李晓强,陈芳,等. 数字音频乐纹技术综述[J]. 小型微型计算机系统,2008,29(11):2124-2130
[7] Wang A L. An Industrial-Strength Audio Search Algorithm [C]// Proc. the 4th International Conference on Music Information Retrieval(ISMIR 2003). 2003
[8] Miller M, Rodriguez M, Cox I. Audio fingerprinting: Nearest neighbor search in high dimensional binary spaces [J]. Journal of VLSI Signal Processing, 2005, 41(3): 285-291
[9] Manber U, Myers G. Suffix arrays: a new method for on-line string searches [C] // 1st ACM-SIAM Symposium on Discrete Algorithms. 1990
[10] Jensen R, Shen Q. Semantics-preserving Dimensionality Reduction: Rough and Fuzzy-rough-based Approaches [J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(12): 1457-1471
[11] 徐英进,蔡锐,蔡莲红. 一种基于“乐纹”的海量音乐检索系统[C]//第二届和谐人机环境联合学术会议(HHME2006)——第 15 届中国多媒体学术会议(NCMT'06)论文集. 北京:清华大学出版社,2006
[12] 姚全珠,张楠,杨增辉,等. 基于压缩后缀数组技术的搜索引擎[J]. 计算机工程,2008,34(10):83-85