E-011

楽曲検索インデックスのコンパクト化
Index Compression for Audio Fingerprinting Systems

齋藤成美 † Narumi Saito　**松本和幸** † Kazuyuki Matsumoto　**鈴木基之** ‡ Motoyuki Suzuki　**北研二** † Kenji Kita

1. はじめに

楽曲の曲名やアーティスト名，アルバム名などの楽曲情報を得るときには，楽曲検索システムが利用されている．マイクから楽曲の一部を読み込み，楽曲情報を得る技術にも使われている．

多くの楽曲検索システムでは，オーディオ指紋 (audio fingerprint) が使われている．オーディオ指紋は，楽曲の特定に用いられる特徴である．オーディオ指紋を楽曲情報データベースと共に用いることにより，楽曲の一部から楽曲情報を得ることができる．また，オーディオ指紋は楽曲を識別できることから，楽曲検索のみではなく，ネットワーク上に流通している著作権を侵害した楽曲の検出も可能である．

効率的なオーディオ指紋検索には，接尾辞配列を利用した手法 [1] がある．これは，楽曲データベースの増大に比例して空間コストが増加する欠点を持つ．本稿では，データベースのインデックスを圧縮することにより，空間コストの削減を行う．

2. オーディオ指紋による楽曲検索

2.1. オーディオ指紋の抽出

オーディオ指紋の抽出には，楽曲のデータ信号の特徴が利用されている．Haitsma-Kalker アルゴリズム [2] では，周波数帯のエネルギー差から得られる符号を特徴量としている．以下に，Haitsma-Kalker アルゴリズムによるオーディオ指紋の抽出を示す．

まずは，楽曲を一定時間のフレーム (フレーム長 0.37 秒) に分割し，各フレームを 33 個の周波数帯に分ける．このとき，フレームは 31/32 をオーバーラップさせるが，周波数帯は重ならないように分割する．次に，各フレームにおける周波数帯のエネルギー差の 32 個の符号を 32 ビットのサブ指紋として抽出する．フレーム n の第 m ビット $F(n,m)$ は，次の式となる．

$$F(n,m) = \begin{cases} 1 & \text{if } ED(n,m) > 0 \\ 0 & \text{if } ED(n,m) \leq 0. \end{cases}$$

ここで，$ED(n,m) = E(n,m) - E(n,m+1) - (E(n-1,m) - E(n-1,m+1))$ であり，$E(n,m)$ は フレーム n の周波数帯域 m におけるエネルギーである．

2.2. オーディオ指紋の照合

サブ指紋は 1 フレームに対する 32 ビットの特徴量である．1 つだけでは楽曲の識別に十分な情報とならないため，時系列のブロックを用いてオーディオ指紋の照合を行う．

オーディオ指紋の照合には，ビットエラー率 (bit error rate) を用いる．2 つの楽曲片 A, B から抽出されたサブ指紋について，フレーム n の第 m ビットをそれぞれ $F_A(n,m)$, $F_B(n,m)$ とする．このとき，長さ N のサブ指紋ブロックに対するビットエラー率 $BER(A,B)$ は，以下の式となる．ただし，$\wedge$ は XOR 演算子を表す．

$$BER(A,B) = \frac{\sum_{n=1}^{N}\sum_{m=1}^{32}[F_A(n,m) \wedge F_B(n,m)]}{32N}.$$

2.3. オーディオ指紋の検索

オーディオ指紋による楽曲検索では，まず，検索対象となる楽曲データベースと検索質問の楽曲片からそれぞれのサブ指紋ブロックを抽出する．データベースの各楽曲と検索質問の楽曲片について，開始フレームの異なる複数のサブ指紋ブロックが得られる．検索質問のサブ指紋ブロックとの距離が最小となるようなデータベースのサブ指紋ブロックを探し出す．距離が最小であるサブ指紋ブロックの情報が検索結果となる．

オーディオ指紋を利用した効率的な検索手法には，高速ハミング空間検索 [1] が提案されている．高速ハミング空間検索では，サブ指紋データベースでの検索を高速に行うため，接尾辞配列 (suffix array) に似た，サブ指紋系列へのインデックス SA を保持している．データベース中の全曲から得たサブ指紋を $FP = FP[0], FP[1], \cdots, FP[n]$ とする．$SA = SA[0], SA[1], \cdots, SA[n]$ における $SA[i]$ は，長さ 3 のサブ指紋系列をソートした位置を示している．SA は，楽曲検索用のインデックスとなる．インデックスを利用することでソート順のデータを得られることから，検索には 2 分探索を用いることができる．

検索インデックスの大きさは，楽曲データベースのサブ指紋系列の長さに比例する．つまり，楽曲データベースの増大に伴い，インデックスも増加してしまう．

3. 検索インデックスの圧縮

インデックスは接尾辞配列と同様の構造である．このことから，インデックスの圧縮には，圧縮接尾辞配列 (compressed suffix array) を利用する [3] .

提案手法では，データベース中のサブ指紋系列をソート順で圧縮し，保持する．すなわち，FP の代わりに $FP'[i] = FP[SA[i]]$ である $FP' = FP'[0], \cdots, FP'[n]$ を用いる．このとき，FP' を一定間隔 M_1 で分割しておく．つまり，FP' のブロック $FP'_k = FP'[M_1 \times k], FP'[M_1 \times k+1], \cdots, FP'[M_1 \times (k+1)-1]$ について $FP' = FP'_0, \cdots, FP'_{n/M_1}$ を考える．これは，検索時に必要となる復元を瞬時に行うためである．

さらに，分割したブロック FP'_k ごとに，1 つのサブ指紋 (32 ビット) を 8 ビット単位で考え，8 ビットデータ列に対してランレングス符号化 (Run-length Encoding, RLE) を行う．RLE は，同じ符号の繰り返しを符号と長さの組に置き換えていく符号化方法である．このとき，8 ビットデータ列は同じ位のデータを並べたもの

†徳島大学, The University of Tokushima
‡大阪工業大学, Osaka Institute of Technology

を考える．FP' はソートされたデータであるため，上位バイトは同じ値になりやすいためである．

たとえば，$M_1 = 8$ で，16 進数で表記した FP'_0 が以下のようだったとすると，RLE により FP''_0 のように表せる．データベースが大きくなるほどランレングス (同じ値が連続する長さ) は長くなり，圧縮効率は増加する．$FP''_0, \cdots, FP''_{n/M_1}$ をデータベースのサブ指紋として保持する．

$$
\begin{array}{rl}
FP'_0 = & 00\ \ 00\ \ 00\ \ 00, \\
& 00\ \ 00\ \ 00\ \ 00, \\
& 00\ \ 00\ \ 00\ \ 01, \\
& 00\ \ 00\ \ 00\ \ 01, \\
& 00\ \ 00\ \ 01\ \ 01, \\
& 00\ \ 00\ \ 01\ \ 01, \\
& 00\ \ 00\ \ 11\ \ 01, \\
& 00\ \ 00\ \ 11\ \ 11.
\end{array}
\qquad
\begin{array}{rl}
FP''_0 = & 00,\ 00,\ 08,\ 00, \\
& 00,\ 08,\ 00,\ 00, \\
& 04,\ 01,\ 01,\ 02, \\
& 11,\ 11,\ 02,\ 00, \\
& 00,\ 02,\ 01,\ 01, \\
& 05,\ 11.
\end{array}
$$

FP' では，元のデータベースの順番が失われてしまうため，元データの順序を示す $\Psi[i]$ を用いる．

$$\Psi[i] = \begin{cases} SA^{-1}[SA[i]+1] & \text{if } SA[i] \neq n \\ 0 & \text{if } SA[i] = n. \end{cases}$$

$\Psi[i]$ により，データベースと同じ系列を示す $FP[SA[i]],\ FP[SA[i]+1],\ \cdots,\ FP[SA[i]+j]$ は $FP'[i],\ FP'[\Psi[i]],\ \cdots,\ FP'[\Psi^j[i]]$ と表すことができる．$\Psi^j[i]$ は，$i = \Psi[i]$ を j 回繰り返すことを示している．

$\Psi[i]$ は部分的に単調増加となるため，圧縮が可能である．部分的に単調増加となるのは，先頭データが同じ値のとき，ソートの順番は 2 番目以降の値により決まるためである．

$\Psi[i]$ の圧縮には，Vertical Code を利用する．Vertical Code では，$\Psi[i]$ の差分である $d[i]$ を使用する．

$$d[i] = \begin{cases} \Psi[i] - \Psi[i-1] - 1 & \text{if } i \neq 0 \\ 0 & \text{if } i = 0. \end{cases}$$

$d[i] < 0$ の場合は，n を加えて常に単調増加としておく．

$\Psi[i]$ を一定間隔 M_2 でブロックに分割する．つまり，$\Psi_k = \Psi[M_2 \times k], \cdots, \Psi[M_2 \times (k+1) - 1]$ についてブロック $\Psi_0, \cdots, \Psi_{n/M_2}$ を考える．ブロックの先頭データ $\Psi[M_2 \times k]$ はサンプリングとして $\Psi'[k]$ に保存する．これは，FP' 同様，検索時に必要となる復元を瞬時に行うためである．

$d[i]$ も一定間隔 M_2 でブロックに分割して考える．つまり，$d_k = d[M_2 \times k], \cdots, d[M_2 \times (k+1) - 1]$ についてブロック $d_0, \cdots, d_{n/M_2}$ を考える．まずは，d_k の最大値から，ブロック内データを表すのに必要なビット数 $MSB[k]$ を得る．また，$d[M_2 \times k + p]$ をバイナリ表現したときの第 q ビットの値を，$V_k[q]$ の第 p ビットとして保存する．すなわち，V_k の大きさ (各 V_k に対する q の最大値) は $MSB[k]$ となる．M_2 を 8 の倍数にすることにより，$V_k[q]$ はバイト単位で処理することができる．

たとえば，$M_2 = 8$ で，10 進数で表記した d_0 が $d_0 = 1,\ 0,\ 1,\ 2,\ 3,\ 2,\ 1,\ 0$ だったとする．ブロック内の最大値が 3 であることから，すべての値を 2 ビットで表すことができる．d_0 を 2 ビットの 2 進数で表すと，$d_0 = 01,\ 00,\ 01,\ 10,\ 11,\ 10,\ 01,\ 00$ であるから，$V_0 = V_0[0],\ V_0[1]$ について，$V_0[0] = 0101\ 0101,\ V_0[1] = 0011\ 1000$ となる．

$\Psi[i]$ の代わりに，前述の処理により算出された $V = V_1, \cdots, V_{n/M_2}$ と $MSB = MSB[1], \cdots, MSB[n/M_2]$ を保持することで，インデックスの圧縮が可能となる．

4. 検索インデックスの復元

$\Psi[i] = \Psi'[i'] + j' + \sum_{k=1}^{j'} d[M_2 \times i' + k]$ と表すことができる．(ただし，$i' = i/M_2,\ j' = i \bmod M_2$．mod は剰余を表す．また，$\Psi[i] > n$ の場合，$\Psi[i] = \Psi[i] \bmod n$ となる．) ブロック $d_{i'}$ の j' までの和を示す $\sum_{k=1}^{j'} d[M_2 \times i' + k]$ は $\mathrm{MASK} = \{(1 << j') | ((1 << j') - 1)\} - 1$ を用いて，$\sum_{k=0}^{MSB[i']} \{\mathrm{popcount}(V'_i[k] \& \mathrm{MASK}) << k\}$ となる．(MASK は 1 から j' までを示すマスクである．popcount(x) は データ x における 1 のビットの個数，| は OR 演算子，& は AND 演算, << は左シフト演算を表す．) 以上の処理により，Ψ', V, MSB から $\Psi[i]$ を得ることができる．

検索には，前述のように，$FP'[i], \cdots, FP'[\Psi[i]^j]$ に対して検索質問のサブ指紋ブロックとのビットエラー率が最小となるものを探せばよい．

5. 実験と結果

1001 曲の MP3 データから抽出したサブ指紋は 29.9MB である．従来手法では，インデックスの 29.4MB を加えて，59.3MB の空間コストがかかっている．提案手法では，サブ指紋のデータベースを RLE ($M_1 = 32$) で圧縮することにより 13.3MB とし，インデックスを Vertical Code ($M_2 = 32$) で圧縮することにより 23.1MB とした．合計で 37.3MB となり，従来手法に比べて 62.4% の圧縮となっている．

6. まとめ

接尾辞配列を利用したオーディオ指紋検索について，サブ指紋のデータベースを Run-length Encoding により，インデックスを Vertical Code により圧縮した．従来手法に比べ，空間コストを削減することができた．

参考文献

[1] 北研二, 肖清梅, “オーディオ指紋検索に適した高速なハミング空間検索”, 情報処理学会研究報告, 2011-MUS-90(4), 1-6, 2011

[2] Jaap Haitsma, Ton Kalker, “A Highly Robust Audio Fingerprinting System”, 3rd International Conference on Music Information Retrieval, ISMIR 2002.

[3] 岡野原大輔, “現実的な圧縮付全文索引”, 情報処理学会 夏のプログラミング・シンポジウム 2005, 2005.