DBSJ Japanese Journal, Vol.13-J, No.1
October 2014

大規模音楽音響信号データベースに対する超高速部分一致近似検索手法

Fast Approximate Matching Approach to Very Large Music Audio Signal Databases

高田 怜 ♥　　喜田 拓也 ♦

Satoru TAKADA　　**Takuya KIDA**

本稿では，大規模な音楽音響信号データベースに対し，音響信号をクエリとして，任意の楽曲の一部分と近似的に一致する箇所を高速に検索する手法について論じる．提案手法では，楽曲の音響信号から抽出された音楽指紋に対して接尾辞配列を適用し，クエリ信号からサンプリングされた多数の小片を厳密一致で検索する．その結果を統合することで高速かつ精度の良い近似検索を実現する．この手法を実現し，大規模なデータベースに対する楽曲検索の実験を行った．速度評価を行うために，実際の音楽音響信号に加え，およそ **10** 万曲分の人工楽曲データベースを作成し，一般的な **PC** 上でそのデータベースに対する検索性能を評価した．その結果，**84%** 程度の精度で，**1** 曲あたり **0.04** 秒未満で検索できることを示した．

In this paper we discuss the problem of finding a piece of music at high speed from a very large music audio signal database. We assume that a query is given as an audio signal of short passages or the whole piece, and also that it can be recorded by other artists. Therefore, the problem is reduced to finding parts that approximately match with the query. Our proposed method uses a set of suffix arrays for sequences of audio fingerprints extracted from the database. We search positions as candidates that exactly match with short bits of the query, and then we merge them to answer the approximate matching problem. We implemented our method and performed an experiment. In the experiment, we added a hundred thousand synthetic music data to an actual database in order to measure the performance of our method. The experimental results showed that our method found a corresponding piece within 0.04 second per query with about 84% accuracy.

♥ 非会員　北海道大学大学院情報科学研究科修士課程
takada@ist.hokudai.ac.jp
♦ 正会員　北海道大学大学院情報科学研究科
kida@ist.hokudai.ac.jp

1. はじめに

近年，iTunes [1] や YouTube [9]，ニコニコ動画 [16] などをはじめとして，個人がデジタル上で扱うことのできる楽曲数が非常に膨大になっている．これらのようなインターネット上のサービスでは，楽曲のアーティスト名や作曲者名などのメタ情報が付加されていることが多い．しかし，個人の PC に保存されているような未整理なデータについてはその限りではない．そのような状況において楽曲の情報を探し出すためには，音源の音響信号データをクエリとして用いて検索を行う手法が必要となる．

そこで本稿では，楽曲の一部分の音楽音響信号をクエリとし，巨大な楽曲データベースに対して高速な検索を実現する手法を提案する．高速な検索を実現するための鍵は，楽曲の音楽音響信号データを音楽指紋と呼ばれる系列データへと変換することである．音楽指紋は，楽曲の元の信号データから抽出される離散的なデータで，元の音源データと比較してデータ量が非常に小さくなる特徴がある．この音楽指紋を用いて，楽曲の高速な検索を実現する．

楽曲の検索手法については，楽曲データ同士のマッチングや類似検索に関する手法が研究されている [2] [14] [8] [17] [15]．また，信号処理分野でよく知られているフーリエ変換 [3] やメル周波数ケプストラム係数 [10] などのほか，特定のワードをクエリとした検索 [18] [21] や，検索のために音楽音響信号データを MIDI に変換するもの [7]，ボーカルの声の質の類似さに基づく検索 [20]，ユーザの好みをジャンルで学習した上での検索 [5]，楽曲にタグ付けを行う手法 [6] などがある．このように，楽曲データ同士のマッチング以外においても楽曲検索のための様々な研究がなされている．

本稿では，楽曲を音楽指紋に変換した上で，音楽指紋同士のマッチングを行うことで楽曲同士のマッチングを実現する．ここでの楽曲検索は，歌唱者が異なった場合でも検索が可能な，類似楽曲検索を想定している．そのため，Haitsma と Kalker が提案した音楽指紋 [4] を元に，異なる歌唱者による同一楽曲検索に適するよう拡張した文献 [19] の音楽指紋を土台として議論を行う．

音楽指紋は，一般的に高次元のビット列である．高次元のビット列どうしを単純に比較計算しようとすると，計算にかかる時間が次元数に対し指数的に増加するため，高速にマッチング検索を行うことが困難であることが知られている．本稿では，文字列検索で用いられる索引構造である接尾辞配列 [11] を音楽指紋に適用する手法を提案する．また，10 万曲分の擬似データベースに対して歌唱者の異なる楽曲検索実験を行い，高速に楽曲を検索できることを示す．

2. 音楽指紋

音楽指紋 (audio fingerprint) とは，楽曲の音響信号データから得られる文字列 (ビット列) のことで，その楽曲の特徴を数キロビット程度の比較的小さいデータサイズで表現したものである．本稿では，音楽指紋を楽曲の同一性を判定するために用いる．音楽指紋から元の音響信号へ復元することはできないが，音楽指紋を用いることで元の楽曲データをそのままデータベースとした時と比べ，保持するデータ量を大幅に少なくすることができる．

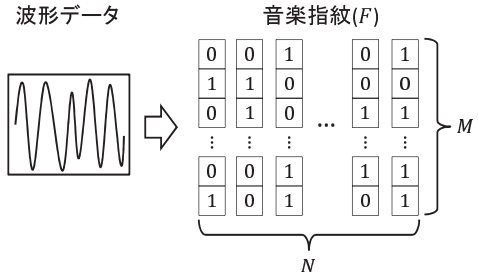

図 1　音楽音響信号から音楽指紋への変換

2.1　音楽指紋の生成

本稿では，著者らが提案した文献 [19] の音楽指紋を用いる．この音楽指紋は，2002 年に Haitsma と Kalker [4] が提案した音楽指紋を元にしている．彼らの音楽指紋と同様に，著者らの音楽指紋も，実際の楽曲の差異が音楽指紋でのビット誤りに反映されるような設計となっている．二つの音楽指紋の違いは，音楽音響信号から抽出する際に選択する周波数帯，およびフレームの重複の有無の違いである．以下では，どちらの音楽指紋にも共通する基本的な部分について説明する．

まず，音響信号を先頭から順に，ある短い長さの区間で区切りながら周波数解析を行う．この時の区間をフレームと呼ぶことにする．各フレームでの周波数の区切り方は対数スケールで行い，ひとつの領域がピアノの鍵盤の音のひとつに対応するような周波数の区切り方をとる．あるひとつのフレームに着目すると，周波数解析によって得られるデータは，区切られた周波数ごとの音の強さである．フレームの数を $N+1$，区切られる周波数の数を $M+1$ とおくと，$M+1$ 次元のベクトルが $N+1$ 個並んだデータが得られる．このデータを行列 E とおく．E の要素 $E(i,j)$ には，i 番目のフレームの，j 番目の高さの音の強さが格納されている．i,j はそれぞれ $[0,N],[0,M]$ の範囲をとる．

次に，E の隣接する成分同士の差分を求め，大きさが (M,N) となるような行列 E' を計算する．具体的には，

$$E'(i,j) = E(i,j) - E(i,j+1) - (E(i-1,j) - E(i-1,j+1)) \quad (1)$$

という計算を行う．この E' の値を用いて，次のような行列 F を生成する．

$$F(i,j) = \begin{cases} 1 & E'(i,j) > 0 \text{ のとき}, \\ 0 & E'(i,j) \leq 0 \text{ のとき}. \end{cases} \quad (2)$$

この行列 F を音楽指紋とよぶ．F は各要素が 0 または 1 であり，ビット行列となっていることがわかる (図 1)．

先に述べたとおり，本稿の実験で使用する音楽指紋は，使用する周波数帯やフレームのとり方などにおいて Haitsma と Kalker [4] が提案したパラメータとは異なるものとなっている．具体的なパラメータの設定については，4.1 節で述べる．

2.2　音楽指紋を用いた楽曲のマッチング

上記の手順によって生成された音楽指紋を用いて楽曲検索を行う方法について概説する．この手順により生成される音楽指紋は，同じ音源からは全く同じものが生成される．しかしながら，例えばクエリの音源にノイズがのっていたとすると，同一の楽曲でも生成される音楽指紋は若干異なるものとなる．そのような音楽指紋に対して検索を可能とするためには，厳密に一致する音楽指紋を検索するのではなく，多少の異なりを許容した類似検索を行う必要がある．本手法においては，楽曲同士の類似度を，音楽指紋同士のハミング距離として定義する．したがって，楽曲を検索するという問題は，クエリの音楽指紋と最もハミング距離が小さくなるようなデータベース中の音楽指紋をもつ楽曲を見つけ出す問題に帰着する．

また，クエリとなる音源が楽曲のある一部分のみである場合を考慮すると，音楽指紋同士の比較は，それぞれのデータを先頭部分から比較するだけでは不十分である．最も単純に音楽指紋を類似検索する方法としては，クエリとデータベースの比較位置を 1 フレームずつずらしながら最もハミング距離の小さくなる場所を見つけ，その部分を内包する楽曲を答えとする線形探索が考えられる．しかしながら，そのような線形探索ではデータベースの大きさに比例して検索時間が増大する．高速な検索を行うには，索引構造を用いるなどの工夫が必要となる．

3.　接尾辞配列

本節では，文字列の全文検索のための索引構造のひとつである接尾辞配列について解説する．

3.1　文字列

文字列 (String) とは，アルファベット Σ 上の記号の列である．文字列 T を $T = x_0x_1 \ldots x_{n-1}$ と表す $(x_i \in \Sigma, i = 0,1,...,n-1)$．この時，文字列 T の長さは n である．また，文字列 T の i 番目の文字から j 番目の文字を取り出すことでできる部分文字列を，$T[i,j]$ と表す．さらに，終端の文字を含まない部分文字列を $T[i,j) = T[i,j-1]$ と表す．つまり，$T = x_0x_1 \ldots x_{n-1}$ のとき，$T[1,5] = x_1x_2x_3x_4x_5$ であり，$T[1,5) = x_1x_2x_3x_4$ である．

3.2　接尾辞配列

接尾辞配列 (Suffix Array) とは，1990 年に Manber と Myers によって提案された，文字列を高速に検索するための索引データ構造である [11].

まず，接尾辞配列の生成に使われる接尾辞 (Suffix) について述べる．長さ n の文字列 T について，T の接尾辞 $S_i (i = 0,1, \ldots, n-1)$ は以下のように定義される．

$$S_i = T[i,n)$$

つまり，ある接尾辞 S_i は，文字列 T の i 番目の文字から最後までを取り出した部分文字列である．接尾辞は，長さ n の文字列に対して n 個存在する．また，接尾辞に対し，ある文字列の先頭から数文字を取り出したものを接頭辞 (Prefix) と呼ぶ．文字列 T の任意の部分文字列は，ある接尾辞 S_i の接頭辞になっていることに注意しよう．

文字列 T に対する接尾辞配列は，T の全ての接尾辞 S_i を辞書順にソートした時の接尾辞開始位置 i を並べた配列である．

3.3　接尾辞配列上での検索

接尾辞配列の特徴として重要なのが，文字列 T 中に存在する任意の部分文字列の先頭の位置が検索できるということである．文字列 T の全ての接尾辞の開始位置を，その接尾辞の辞書順にソートしてあるため，同じ部分文字列の開始位置は接尾辞配列上で連続して格納されている．したがって，文字列 T 上にクエリの文字列が含まれているかどうか，あるいは，文字列 T のど

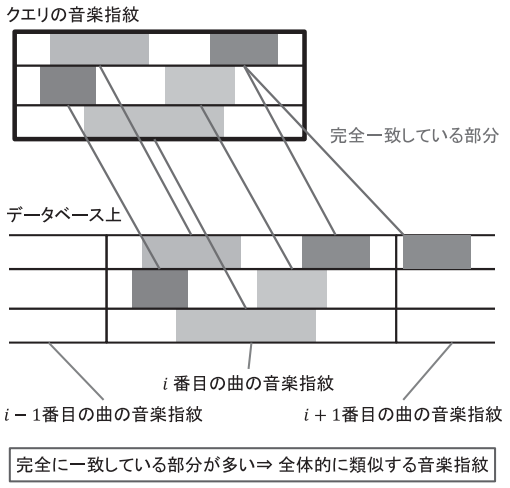

図 2　小片の厳密一致検索による類似楽曲検索のアイデア

の部分にクエリの文字列が含まれているのかは，接尾辞配列上を二分探索することで求めることができる．これにより，任意の部分文字列に対して，文字列 T 上を単純に線形探索する時と比べて高速に検索することができる．

4. 提案手法

4.1 手法の概要

先の 2.2 節で述べたように，音楽指紋を用いた楽曲の検索は，先頭の開始位置が一致しないビット列同士の類似検索によって行うことができる．提案する手法では，ビット列同士のハミング距離を正確に調査することをあきらめ，クエリの多数の小片が完全一致する楽曲を類似楽曲と判断する手法を取る (図 2)．

提案手法で使用する音楽指紋は，筆者らの先行研究 [19] で提案したものを用いる．すなわち，2.1 節の手法による音楽指紋の生成において，フレーム長を 0.1 秒とし，また各フレームは重複させないものとする．さらに，周波数の高さの区切り数 M を $M = 24$ とし，使用する周波数帯は 41.2Hz〜164.8Hz とする．つまり，フレーム毎に，楽曲の音楽音響信号から 41.2Hz〜164.8Hz の部分を取り出し，その対数領域を 25 分割する．そして，式 (1) と (2) によって，長さ 24 のビット列を生成する．以上の処理によって，例えば長さ 200 秒の楽曲からは，縦 24 ビット，横 2000 フレーム分の音楽指紋が生成される．このような音楽指紋は，歌唱者が異なるクエリに対する類似楽曲の検索に適したものであることがわかっている [19]．

4.2 音楽指紋の接尾辞配列

音楽指紋に対する接尾辞配列を用いた索引データ構造について述べる．

前述の音楽指紋は，各フレームごとに 24 ビットのデータになっている．これを，図 3 のように連続する 8 ビットごとに 3 つのセットに分割する．以降，このセットひとつを 1 バイトでコード化された文字とみなす．図 3 の右側のひとつの四角形がひとつの文字に対応する．すなわち，各フレームは 3 つの文字で表現され，それぞれは周波数の高さで区切られている．この分割を全てのフレームについて行うと，データベース上の各楽曲は縦が 3 文字，横がフレーム数の行列データとなる．ここで，

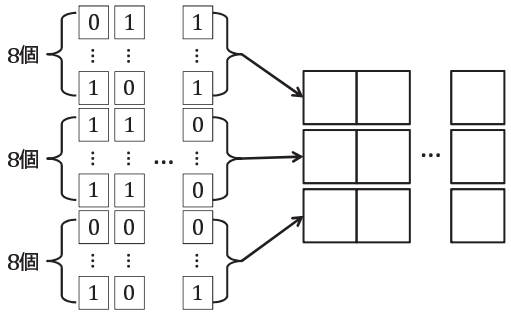

図 3　フレームの文字列への分解の仕方

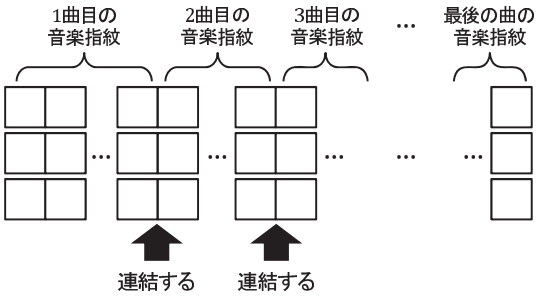

図 4　データベース全体から 3 本の文字列を構成するイメージ

行ごとに文字を横方向に連結すると，楽曲は 3 本の文字列として表現されていると見ることができる．データベース全体では，同じ高さの文字列を順に連結することで 3 本の長大な文字列が得られる (図 4)．

こうして得られた 3 本の長い文字列に対して，それぞれ接尾辞配列を生成する．クエリの音楽指紋において，先頭から j 番目のフレームの上から i 番目の高さの文字を $x_{i,j}$ と表す．クエリの音楽指紋のフレーム数を N とすると，i, j はそれぞれ $i = 0, 1, 2$，$0 \leq j \leq N-1$ の範囲をとる．この i と対応するように，データベースの 3 つの接尾辞配列を SA_0, SA_1, SA_2 と表すことができる．

4.3 接尾辞配列上での検索方法

接尾辞配列を用いた音楽指紋の検索は，以下のような手順で行われる．

まず，クエリの音楽指紋からひとつの文字をランダムに取り出す．とり出された文字を $x_{i,j}$ とし，文字列 X を $X = x_{i,j}$ とおく．次に，SA_i 上において，X を接頭辞として含む接尾辞を二分探索で見つけ出す．このとき，該当する接尾辞は複数あることが考えられるので，その接尾辞の数 R_1 を求める．接尾辞配列の生成方法から，ここで探索される接尾辞は SA_i 上で全て連続していることに注意する．

次に R_1 と，あるしきい値 Δ を比較する．この時，$R_1 \geq \Delta$ であるときは，続いて $x_{i,j+1}$ を取り出し X の末尾に加え，文字列 X を $X = x_{i,j}x_{i,j+1}$ とする．そして SA_i 上において，X を接頭辞とする接尾辞の数 R_2 を新たに求める．この時，一致させる文字列が長くなっているので，$R_2 \leq R_1$ である．

$x_{i,j+2}, x_{i,j+3}, \cdots$ をひとつずつ X に加えながら以上の操作を繰り返し，$R_l, (l = 1, 2, \cdots)$ が Δ 以下になったときに探索を終了

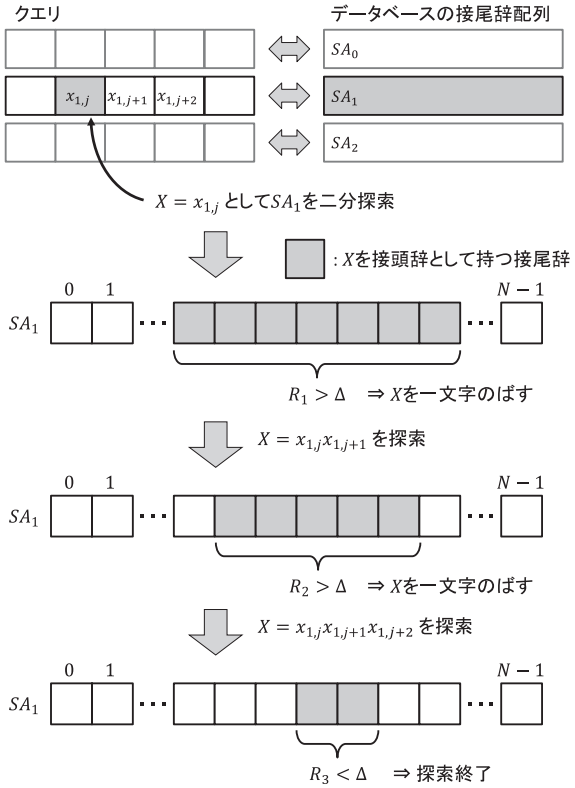

図 5　接尾辞配列を用いて楽曲の小片を検索する様子

する (図 5)．この時点で残っている接尾辞の開始位置を内包している楽曲に，1 点を加点する．よって，ひとつの楽曲に複数の開始位置が含まれている場合，含まれている個数分の点数が加点される．

上述の，ある $x_{i,j}$ を取り出しデータベースの楽曲に加点するという操作を，ある一定数 S 回繰り返す．そして，最終的に最も加点数の多かった上位数曲を，クエリの楽曲と同一の楽曲であるとして出力する．

5. 実験

本節では，前述の提案手法を用いて楽曲の検索実験を行う．

5.1 実験に使用するデータ

データベースとして，7849 曲の楽曲データを用いた．この中には，一般的な J-POP 楽曲，ニコニコ動画や YouTube などの動画投稿サイトで公開されている VOCALOID 楽曲 (以下，VOCALOID 曲) などが含まれている．クエリとしては，J-POP 曲から 78 曲，VOCALOID 曲から 100 曲，オフボーカル (伴奏のみ) の曲 22 曲の計 200 曲を用意した．データの収集は，主にニコニコ動画や CD 音源などを用いて行った．クエリの J-POP 曲と VOCALOID 曲はニコニコ動画から，オフボーカルの楽曲は CD から取得した．

クエリのうち，オフボーカル曲については，いわゆるカラオケバージョン（Instrumental バージョン) をデータベース中の楽曲と同一の CD から入手したものであるため，クエリとして用いるデータの中では最も条件の良いものである．一方で，J-POP 曲，VOCALOID 曲に対するクエリは，同一の楽曲がデータベース中に存在するが，原曲とは歌唱者が異なるものを選択した．ただし，どちらの場合においても，原曲からキーを変えずに歌唱されているものを選んだ．特に J-POP 曲については，著作権の問題から，音質が悪いものやバンドのアレンジが異なっているなど，原曲とはかなり録音状態が異なるものになっている．したがって，オフボーカル曲のクエリは，同一の音楽指紋を持つ楽曲の検索に相当し，他方，J-POP 曲と VOCALOID 曲のクエリは，歌唱の録音状況や歌唱者が異なる類似楽曲の検索に相当する．

さらに，クエリとして用意した楽曲は，楽曲全体のものの他に，楽曲のサビ部分を抽出したものの 2 種類を用意した．これは，イントロや間奏など，歌唱の無い部分が検索精度に影響を与えてしまう可能性があるということを考慮したためである．データベースの楽曲の長さは平均で 3 分 49 秒，クエリのデータは楽曲全体のもので平均 4 分 19 秒，サビ部分を抽出したもので平均 27 秒程度であった．

音楽指紋の生成の際，音響データの解析には Muller らが公開している chroma toolbox [13] を用いた．また，接尾辞配列の生成には Yuta Mori が公開しているプログラム [12] を用いた．実験に使用した計算機は，Intel Core i5 (2.5GHz)，メモリサイズ 8GB，OS は Ubuntu12.04 (64bit 版) である．

検索実験は，まず検索が間違いなく行われているかどうかの調査を行い，続いて検索精度を調査する．その次に，巨大なデータベースに対しての検索にかかる時間がどの程度であるかを調査する．前章の最後で，提案手法の最終的な出力は，得点によるランキング上位数曲であると述べた．今回の実験では，上位 10 曲を出力するものとし，その中にクエリと同一の楽曲が含まれているならば検索が成功したと判断する．検索時間は，クエリの音楽指紋で接尾辞配列を探索し始めてから答えが出力されるまでとする．

5.2 システムの調査

まず，検索システムがビット列を間違いなく検索できているかを確認する．データベースとして用意した楽曲から，200 曲を複製し，それをクエリとして検索実験を行う．つまり，ここではクエリと全く同じビット列を検索することになる．

結果としては，全てのクエリにおいて，同一の楽曲をデータベースから発見することに成功した．検索システムが，類似ビット列を間違いなく発見できていることを確認した．

5.3 類似楽曲の検索精度

続いて，歌唱者の異なる楽曲を用いた検索実験について述べる．ここで，文字を選び加点するという操作を繰り返す回数 S を 2000，接尾辞配列の幅 Δ を 15 とする．これらのパラメータは，予備実験によって検索時間と精度のバランスを考慮して決定した．また，楽曲全体，サビのみの双方で，検索にかかる時間を測定した．

各クエリについて 20 回の検索を行った．その時の正答率と計算時間の平均を表 1 に載せる．

まず，検索精度について考察する．オフボーカル曲は，クエリの音源が CD から得られているため，クエリが楽曲全体，サビのみのどちらでも検索精度は非常に良い結果となった．VOCALOID 曲と J-POP 曲については，サビのみとした時の正答率が，楽曲全体で検索した時と比べて下がる結果となった．

表 1　実際の楽曲 7849 曲に対する検索の正答率と一曲あたりの検索時間.

	全体	サビのみ
オフボーカル	100.0%	100.0%
VOCALOID 曲	95.3%	86.0%
J-POP 曲	66.0%	57.7%
合計	84.4%	76.5%
一曲あたりの検索時間	0.018 秒	0.006 秒

表 2　人工データを加えた 107849 曲に対する検索の正答率と一曲あたりの検索時間.

	全体	サビのみ
オフボーカル	100.0%	100.0%
VOCALOID 曲	94.5%	88.0%
J-POP 曲	66.3%	56.4%
合計	84.1%	77.0%
一曲あたりの検索時間	0.035 秒	0.010 秒

これは，ボーカルの違いが音楽指紋の違いに影響し，それが検索精度に影響を与えるという予想を裏付けているといえる．また，特に J-POP 曲での正答率は，全体，サビのみのどちらにおいても低い結果になった．これは，クエリとして用意した音源の録音状況が悪いことが大きく影響していると考えられる．同様に VOCALOID 曲は，J-POP 曲ほどではないが，データベース中の原曲とは録音状況が異なっているため，正答率低下にある程度の影響がみられた．以上のように，検索精度は，クエリの音源データの質そのものが結果に大きく影響しており，検索手法の性能というよりは音楽指紋の設計の仕方により大きく依存していると考えられる．

次に，検索時間について考察する．7849 曲のデータベースに対し，楽曲全体での検索にかかる時間は 0.02 秒未満となった．また，サビのみでは 0.01 秒未満である．サビのみでの検索が全体での検索よりも時間が小さい理由は，サビのみのクエリは全体の文字数がサンプル数 $S = 2000$ よりも少ないものが多く，開始位置を重複して選択することを排除しているために小片の検索が 2000 回未満で打ち切られるからである．

参考までに，同じく接尾辞配列を用いた楽曲検索手法を提案している Xiao ら [17] の実験では，Intel Core i7 (1.73GHz), メモリサイズ 4GB の計算機が用いられ，楽曲の検索に要した時間は 8740 曲のデータベースに対し 1 曲あたりおよそ 0.4〜0.6 秒程度とのことであった．同じ計算機，実験データを用いた比較が行えなかったのは，[17] の著者から提供を受けた検索システムが，楽曲の解析から検索までを一度に全て行う仕様となっており，検索部分のみの時間計測を行えなかったためである．Xiao らの検索手法について簡潔に概説すると，次のような手順で楽曲の検索が行われる．まず，ある固定長の数フレーム分 (文献 [17] の実験においては，1 フレーム 32 ビットで 3 フレーム分) に相当する音楽指紋 (SSF と呼ばれる) を接尾辞配列で探索する．その結果得られるマッチ位置において，より長いフレーム分の音楽指紋 (文献中ではサブ指紋ブロックと呼ばれる) を取り出し，クエリとのビット誤り率を計算する．クエリ中の全ての SSF について上記の処理を行い，総合的にビット誤り率の低い順に上位の数曲を出力する．Xiao らの手法と比べると，本稿の提案手法は，任意長のフレームを探索することで候補位置を絞り込む点や，ビット誤り率を計算することなくランキングを行う点などが異なる．

5.4　検索速度

続いて，巨大なデータベースに対しての検索速度を調査する．

実際の音楽データを多数集めることが困難であったため，乱数によって音楽指紋を擬似的に生成することで巨大なデータベースを作成した．ここでは，擬似音楽指紋として 10 万曲分の音楽指紋を生成し，前小節までで使用したデータベースに追加する形で新たなデータベースとした．したがって，実際に使用するデータベースは合計で 107849 曲分のものとなる．データベースの 1 曲あたりの長さは 3 分 49 秒であったため，ここで生成する擬似音楽指紋は 1 曲あたりのフレーム数を 2300 とした．したがって，文字数は 6900 である．

各クエリについて 20 回の検索を行った．その時の正答率と計算時間の平均を表 2 に載せる．

楽曲全体での検索で 1 曲あたり 0.035 秒，サビのみでの検索で 1 曲あたり 0.01 秒という結果となった．7849 曲のデータベースに対しての検索時間は 0.018 秒であり，およそ 14 倍の大きさのデータベースに対して検索時間は 2 倍程度で抑えられるという結果となった．これは，接尾辞配列上を二分探索するために，検索がデータベースサイズの対数に比例した時間で抑えられていることが理由として挙げられる．また，精度についても，7849 曲のデータベースでの検索と大きな差はないという結果となった．データベースを大きくした上で精度が上昇したものがあったのは，ランキングの阻害となる偽陽性の照合結果がならされ，正解データが目立ちやすくなったためだと考えられる．

6.　結論

本稿では，楽曲の高速な検索手法について，歌唱者の異なる音源データを用いて議論した．その結果，我々の提案する手法は，10 万曲分を超える擬似的な楽曲データベース上において，1 曲あたり 0.04 秒未満で検索を可能とすることが確認できた．

今後の課題としては，本稿の実験で用いた J-POP 曲のような，ノイズの多い音源を用いても正確に検索を行えるように検索手法を改良することが挙げられる．

[謝辞]

本研究を進めるにあたって，文献 [17] のシステムを快く提供してくださった徳島大学の北 研二先生に深く感謝いたします．

[文献]

[1] Apple.com. iTunes store. http://www.apple.com/jp/itunes/.

[2] Luke Barrington, Antoni Chan, Douglas Turnbull, and Gert Lanckriet. Audio information retrieval using semantic similarity. In *Proceedings of IEEE International Conference on Acoustics, Speech and*

Signal Processing, ICASSP 2007, pp. II–725–II–728, 2007.

[3] Dimitrios Fragoulis, George Rousopoulos, Thanasis Panagopoulos, Constantin Alexiou, and Constantin Papaodysseus. On the automated recognition of seriously distorted musical recordings. *IEEE Transactions on Signal Processing*, Vol. 49, No. 4, pp. 898–908, 2001.

[4] Jaap Haitsma and Ton Kalker. A highly robust audio fingerprinting system. In *Proceedings of the 3rd International Society on Music Information Retrieval, ISMIR 2002*, pp. 107–115, 2002.

[5] Keiichiro Hoashi, Kazunori Matsumoto, and Naomi Inoue. Personalization of user profiles for content-based music retrieval based on relevance feedback. In *Proceedings of the Eleventh ACM International Conference on Multimedia*, pp. 110–119, 2003.

[6] Matthew D. Hoffman, David M. Blei, and Perry R. Cook. Easy as cba: A simple probabilistic model for tagging music. In *Proceedings of the 10th International Society on Music Information Retrieval, ISMIR 2009*, pp. 369–374, 2009.

[7] Ning Hu, Roger B. Dannenberg, and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, WASPAA 2003*, pp. 185–188, 2003.

[8] Frank Kurth and Meinard Müller. Efficient index-based audio matching. *IEEE Transactions on Audio, Speech and Language Processing*, Vol. 16, No. 2, pp. 382–395, 2008.

[9] YouTube LLC. Youtube. http://www.youtube.com/.

[10] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval, Electronic Edition*, 2000.

[11] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327, 1990.

[12] Yuta Mori. White page. http://homepage3.nifty.com/wpage/.

[13] Meinard Müller and Sebastian Ewert. Chroma toolbox: Matlab implementations for extracting variants of chroma-based audio features. In *Proceedings of the 12th International Society on Music Information Retrieval, ISMIR 2011*, pp. 215–220, 2011.

[14] Meinard Müller, Frank Kurth, and Michael Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Society on Music Information Retrieval, ISMIR 2006*, pp. 288–295, 2005.

[15] Hidehisa Nagano, Kunio Kashino, and Hiroshi Murase. Fast music retrieval using polyphonic binary feature vectors. In *Proceedings of 2002 IEEE International Conference on Multimedia and Expo, ICME 2002*, 2002.

[16] niwango. ニコニコ動画. http://www.niconico.jp/.

[17] Qingmei Xiao, Motoyuki Suzuki, and Kenji Kita. Fast hamming space search for audio fingerprinting systems. In *Proceedings of the 12th International Society on Music Information Retrieval, ISMIR 2011*, pp. 133–138, 2011.

[18] 辻康博, 星守, 大森匡. 曲の局所パターン特徴量を用いた類似曲検索・感性語による検索. 電子情報通信学会技術研究報告. SP, 音声, Vol. 96, No. 565, pp. 17–24, 1997.

[19] 高田怜, 喜田拓也. 歌唱者の異なる同一楽曲の検索に適した音楽指紋. 情報処理学会研究報告. 音楽情報科学研究会報告, Vol. 2013, No. 7, pp. 1–6, 2013.

[20] 藤原弘将, 後藤真孝. Vocalfinder: 声質の類似度に基づく楽曲検索システム. 情報処理学会研究報告. 音楽情報科学研究会報告, Vol. 2007, No. 81, pp. 27–32, 2007.

[21] 福村純也, 川越恭二. 特徴空間とメロディ空間を用いた楽曲の類似検索方法. 情報処理学会研究報告. データベース・システム研究会報告, Vol. 2003, No. 5, pp. 17–24, 2003.

高田 怜 **Satoru TAKADA**

執筆当時 北海道大学大学院情報科学研究科コンピュータサイエンス専攻修士課程在学．2014 同課程修了．現在 大日本印刷株式会社勤務．

喜田 拓也 **Takuya KIDA**

北海道大学大学院情報科学研究科准教授．2001 九州大学大学院システム情報科学研究科博士後期課程修了，博士（情報科学）．2001 九州大学附属図書館，研究開発室専任講師．2004 年より現職．テキストアルゴリズムおよび情報検索技術に関する研究に従事．情報処理学会正会員．電子情報通信学会正会員．日本データベース学会正会員．