時系列アクティブ探索による類似音楽検索システムの改良

大西　正志† 　辻　紗千† 　獅々堀正幹†† 　北　研二††

† 徳島大学工学部知能情報工学科　〒 770–8506 徳島県徳島市南常三島 2 丁目 1 番地
†† 徳島大学大学院ソシオテクノサイエンス研究部　〒 770–8506 徳島県徳島市南常三島 2 丁目 1 番地
E-mail: †ohnishi-masashi@iss.tokushima-u.ac.jp, ††{tsujis,bori,kita}@is.tokushima-u.ac.jp

あらまし　近年におけるデータベースは巨大化と共に音楽や動画像など様々な種類のデータを扱うようになってきており, マルチメディアデータベースの検索システムに関する研究も活発である. 本研究では時系列アクティブ探索を用いた類似音楽検索システムの構築とその最適化を行う. 最適化は「探索時に用いるヒストグラムのデータ構造改善」および「Chroma ベクトルの利用およびそれにおける適応的な二値化処理」を行う. 前者によって検索速度の向上及びメモリ使用量の低下を図り, 後者によってノイズに対する頑健化などを行い類似度検索の性能向上を図る.

キーワード　時系列アクティブ探索, マルチメディア検索, 音楽検索

Improvement of Retrival System of Similar Music by Time-Series Active Search

Masashi ONISHI†, Sachi TUJI†, Masami SHISHIBORI††, and Kenji KITA††

† Department of Information Science and Intelligent Systems, Faculty of Engineering, University of Tokushima
2-1 Minami-Jyosanzima, Tokushima, Tokushima, 770–8506 Japan
†† Institute of Technology and Science, Tokushima University
2-1 Minami-Jyosanzima, Tokushima, Tokushima, 770–8506 Japan
E-mail: †ohnishi-masashi@iss.tokushima-u.ac.jp, ††{tsujis,bori,kita}@is.tokushima-u.ac.jp

Key words　Time-Series Active Search, Multimedia Retrieval, Music Retrieval

1. 背景と目的

近年, 動画像や音楽などのマルチメディアコンテンツをインターネット上各種投稿サイト等で取り扱うようになったことに伴い, データベースにおいてもマルチメディアコンテンツを扱う場面が増えている. しかし, この種のデータベースでは検索手段が必ずしも充実しておらず, 基本的に「文字」をクエリとしたタイトル・タグ等のキーワードを検索する以外に検索方法があまり提供されていない. マルチメディアコンテンツを利用する上で, あるコンテンツと同じ, あるいは類似したものを検索する技術 (マルチメディア検索) は重要である. これは例えば「初めて聞いた曲のタイトルが知る (逆引き)」「好きなこの曲と似たような楽曲を探す」ということを可能にし, また前述のような投稿サイトを管理する人々はタイトルを偽装した無断転載, 無断利用への対処手段としてそのまま用いる事が可能になるなど, 恩恵を得られる範囲は極めて大きい.

本論文では, マルチメディア検索の一手法である時系列アクティブ探索法 [1] を用いた音楽検索システムの構築, およびそれを通してこの探索法の改良を行う. 時系列アクティブ探索法は, 動画や音楽といった時系列を持つマルチメディアを断片から探索する手法として従来から提案されているものである. この探索方法のポイントは「ヒストグラムによる探索」および「特徴量ベクトルの量子化」という点であり, これらによって探索精度を落とすこと無く高速な探索を行うことを可能としている. 改良は「ヒストグラムのデータ構造」および「ベクトル量子化手法」の 2 点において行う. 前者は, この手法によって扱うヒストグラムが, 頻度値 (縦軸) に対して階級数 (横軸) が大きい「疎」な状態になっている事に着目したものであり, これを連想配列によって扱う事で空間計算量および時間計算量の削減を目指す. また後者は, ノイズに強い音楽検索を行うための特徴量ベクトルや, その適切な量子化方法を探るために行う. また, 実用的な音楽検索システムの構築を行う上で必要なものを十分に考慮し, あらゆる意味での妥当な処理方法およびデータの扱い方を行うよう心がける.

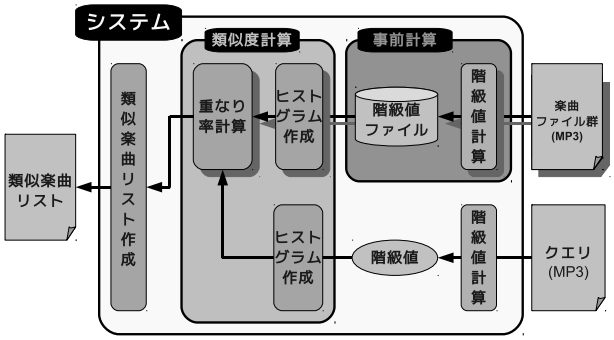

図 1　時系列アクティブ探索法の手順

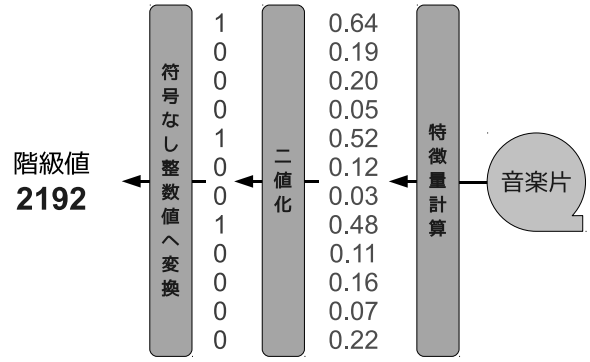

図 2　特徴量ベクトル量子化の一例 (閾値 0.30 による各次元の 2 値化)

2.　時系列アクティブ探索法

時系列アクティブ探索法は, 時系列を持つマルチメディアデータの検索手法の一つとして提案されたものである.

例えば, 長大な音響信号の中から特定の音響信号が存在する地点を探し出すことを考える. ごく単純には, 波形やスペクトラムをずらしつつ照合するという方法が考えられるが, これを隈なく行えば極めて長い処理時間を要する. この短縮の為, ある程度飛ばしながら探索することも考えられるが, 単純に行えば正解地点の見逃しによる精度の低下が避けられない.

時系列アクティブ探索法ではこれを解決する為のアイデアとして, 各地点での比較を特徴量のヒストグラムによって行っている. 類似度はヒストグラム同士の重なり率であり, さらにここから合理的な計算によって探索間隔を間引くことができ, 精度を落とさずに高速な検索が可能となる事が大きなポイントである.

具体的な処理の流れは図 1 および, 以下に示す通りである.

- 事前計算

（ 1） データベース全てのマルチメディアコンテンツに対し, 固定間隔で時系列順に特徴量ベクトルを取る.

（ 2） 特徴量をベクトル量子化し, そのままの順序で保存する.

- 検索処理

（ 1） クエリとして与えられたデータに, 事前計算時と同様の特徴量ベクトル抽出, およびベクトル量子化を行う.

（ 2） クエリにおいて, 量子化された特徴量からヒストグラムを作成する.

（ 3） データベース中ある一つのコンテンツにおける最初の地点を参照する.

（ 4） 参照地点からクエリと同じ長さの区間に関して, ヒストグラムを作成する. 参照地点+クエリの長さがコンテンツの末尾を越えたら次のコンテンツへ進み, データベース中の全てのコンテンツに対して処理を行った後は (7) へ進む.

（ 5） クエリと参照地点のヒストグラムにおける重なり率を計算し, その値がコンテンツ中で最大となった場合にはコンテンツ毎に参照地点とセットで保存する.

（ 6） 参照地点を時系列方向へ少し進めて (4) へ戻る. この時, 進める時間は先ほど求めた重なり率から算出する.

（ 7） 重なり率の大きい順にコンテンツを並び替え, それを検索結果として何らかの方法で出力する.

手順から分かるように,1 コンテンツ中で計算される最大のヒストグラム重なり率が, そのコンテンツとクエリの類似度として扱われる事となる. また, 特徴量の種類や次元数についていかなるものを選択してもよく, マルチメディアの種類についても時系列が存在すること以外の制限は無い. 本研究ではこのシステムを元にして音楽データベースを作成した.

2.1　特徴量のベクトル量子化

時系列アクティブ探索では, 高次元の特徴量ベクトルを 1 次元の数値へ写像する「ベクトル量子化」と呼ばれる処理を行っている. これは, 特徴量のパターンに対して整数番号を割り振ることと同等であり, 一種のクラスタリングであるとも言える.

ヒストグラムにする際に利用する値は量子化後の値であり, また全体を通してその値しか利用することはないため, 実装上も量子化された値のみを特徴量として扱い, 事前計算後にも保持することとなる. また, この量子化に関して特に方法は指定されておらず, 何らかの方法で多次元特徴量ベクトルを 1 次元整数値へ写像すればよい. 例えば, 各次元の値を閾値で B 値化したあと, そのパターンに対して番号を割付ける (特徴量ベクトルをそのまま『B 進数による非負整数値表現』と見なす) というものがあり, 参考文献 [1] ではこの方法を用いていた. 図 2 は $B = 2$ での一例である.

2.2　実際に使用する特徴量について

前述の通り, 特徴量は比較的自由に選択可能である. 本論文では音楽検索を行うに当たり「クロマ・ベクトル (Chroma Vector)」および「MFCC」を用いることを考えた.

クロマ・ベクトルとは,「全てのオクターブにおける『ド・ド#・レ・レ#・ミ・ファ・ファ#・ソ・ソ#・ラ・ラ#・シ』各 12 音が表す周波数でのパワーの合計」を表した 12 次元ベクトルである. 今回は, 楽曲の和音およびその進行を元に検索される事を期待して利用する.

MFCC は, 主に音声認識において用いられる特徴量である. これを用いて音色や音楽ジャンルに間する同定を行おうとする研究もあり, 本論文でも音色の特徴によって検索が可能であると考え, 使用した.MFCC にはいくつかの利用方法があり, それによって最終的な次元数が大きく異なる. 今回は最も基本的な 12 次元の MFCC 特徴量のみを用いて実験を進めた.

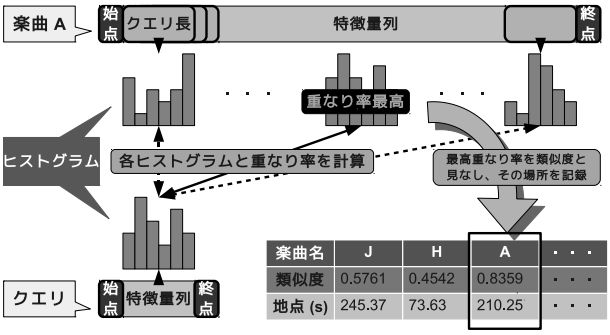

図 3　ヒストグラムの比較による探索

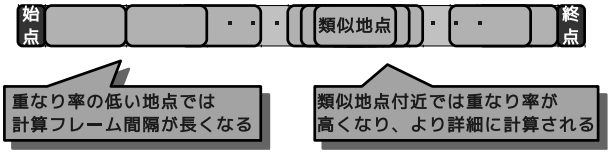

図 4　探索スキップの方法論

2.3　ヒストグラムによる探索

時系列アクティブ探索では, クエリと各データの比較を全て特徴量から作成されるヒストグラムによって行う. 手順は図 3 及び前述の通りである. ヒストグラム重なり率をデータベース内コンテンツ各々に対して計算し, それぞれのなかで最大の重なり率とその地点を, そのコンテンツに対する類似度および類似地点としている. この探索の際, 参照地点を時系列順に進めるにあたり, それをどの程度の間隔とするのかで速度と精度の兼ね合いとなってくるが, 時系列アクティブ探索においては精度とのバランスが取れたスキップ幅を求めることが可能である. 今, 重なり率 $\theta(0 \leqq \theta \leqq 1)$ 以上の地点を求めたいと仮定し, 参照地点における頻度値の合計 (クエリの時系列サンプル数と同数) を D, 重なり率を S とすると, その地点からのスキップ幅 w は以下の様になる.

$$w = \left\{ \begin{array}{cc} floor(D(\theta - S)) + 1 & (S < \theta) \\ 1 & (otherwise) \end{array} \right\} \qquad (1)$$

なお $floor(x)$ は床関数を表し, 実数 x 以下の最大の整数を返す. この式は, 図 4 のような仕組みを実現するものである.

この式によると, 重なり率が θ 以上である場合には全探索と同様の処理を行うが, そうでない場合にはスキップ幅を大きく取る. この根拠としては, スキップ幅 1 で探索を進めても重なり率が上昇する値が限られるという計算上の性質が挙げられる. この性質のため, ある地点での重なり率が θ を下回る場合には, そのまま探索を続けてもしばらくの間は重なり率が θ 以上とならない事がある. この探索する意味の無い範囲を計算し, 精度を落とすことのない最大の w を算出しているのが式 (1) である. 時系列アクティブ探索法では, このようにして精度を保った探索速度の向上を実現している.

3.　時系列アクティブ探索法の改良

3.1　連想配列によるヒストグラムの取扱い

時系列アクティブ探索法におけるヒストグラムでの「階級 (横軸)」の数は,d 次元特徴量の各次元を B 値化する場合には B^d となり, 簡単に大きくなりやすい. この様なヒストグラムを通常の配列で処理した場合, 特徴量の次元数増加と共に, 使用するメモリ空間量や, 配列全体の走査が必要な重なり率の計算時間も大幅に増加する. このため, 使用可能な特徴量がある程度限られていた.

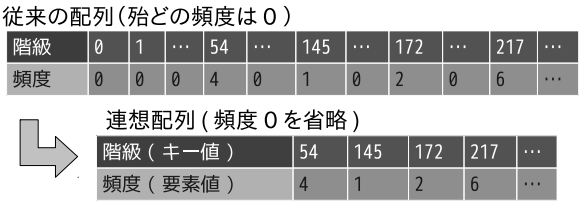

図 5　ヒストグラムのデータ構造

一方,「頻度の総値が階級数に比較して小さい」状態でもある. これは, クエリの長さがデータベース内のコンテンツに比べると短いことが多いに起因する. 例えば今回, 特徴量を得る間隔を 80[msec] としたため,10 秒間で 125 個の階級値が得られる. また特徴量は 12 次元なので 12bit データのヒストグラムとなり, 階級の種類は $2^{12} = 4096$ 個となる. 音楽クエリを 20 秒とすれば階級数は 250 個得られ, それらの値によってヒストグラムが生成されるが, 頻度 0 となる階級が 9 割以上発生する. 頻度 0 となる場合が一つも発生しなくなるような状況は, クエリの長さが最低でも 328 秒以上なければ発生しない.

この「疎」な状態のヒストグラムを扱うデータ構造を考える上では,「必要なデータのみ」を「個々で即座に参照可能な形」によって保存する事が最もよいと考えられる. これは即ち図 5 のような構造をしていて, なおかつ「値の追加・削除を行いやすく」「キー値等で特定の値を高速に参照可能」という性質を満たすものが理想であるということを意味する. 今回は, このような性質を満たすデータ構造として「ハッシュテーブルを用いた連想配列」を利用する. 連想配列は基本的に新たな添字を使用するときに始めてその添字で利用する領域を確保するため, これを利用して頻度値が 1 以上となる階級の情報のみを保存し, 計算時間や空間量の削減を図る. キーと値の組を保存し, キーに当たる値の参照や全体の走査の際はリストをたどるような形になるため, それほどヒストグラムが疎でない場合にはメモリ空間量や計算時間共に通常の配列よりも悪化するが, 殆どが頻度 0 であるというような場合には確実に効果を発揮するものと考えられる. また, 配列サイズが特徴量の次元数に依存しないため, 高次元特徴量でも利用可能になることが期待できる.

3.2　ベクトル量子化手法の改良

従来の時系列アクティブ探索法でのベクトル量子化手法は, 各次元の値を閾値で B 値化したあと, それを B 進数による非負整数値表現と見なすものである. しかし閾値は実験的に決定する必要があり, あるデータベースでその値を適切に設定したとしても, 他の全てのデータに対応できる保証はない.

また, この手法が全ての特徴量に対して最適であるとは限らない. 例えばクロマベクトルにおいて B=2 のベクトル量子化を行うことを考える. 各次元は各音名の音量であるため,2 値化後

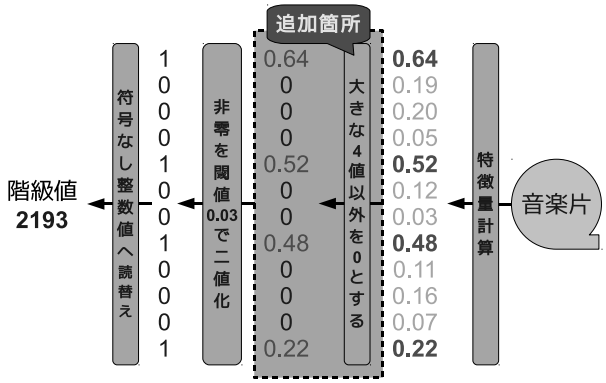

図 6　改良したベクトル量子化手順の一例

に 1 となるものが鳴っており ,0 は鳴っていない状態であると見なせる. この様な音の組み合わせからコードネームが判別可能であり , ベクトル量子化後の数値はこれに対応する. しかし , 閾値による 2 値化の場合には値が 1 に偏重する事が考えられ, これは全ての音が鳴っていることを意味するが, 通常の音楽ではありえない.

よって, より合理的な量子化方法を考える. 例えばクロマベクトルでは「 一般的に使用される和音は 3 または 4 つの音から構成される」という事を利用し ,「 数値の大きな 4 次元を 1, その他を 0」とすることによって二値化を行うことが合理的であると考えられる. このような順位付けによる方法では, 無音地帯でも 0 以外の値を出力する事が考えられるが, この対策として閾値による 2 値化を併用すると良いと考えた.

まとめると ,2 値化の手順は以下の通りになる.

（ 1） V 次元特徴量ベクトルにおいて各次元の値を大きさ順で比べ, 数値が大きい次元 N 個以外では値を 0 とする.

（ 2） 残った次元それぞれで, 数値が閾値よりも小さければ 0, 大きければ 1 とする.

（ 3） 各次元が 2 値化されたベクトルをビット列による非負整数表現と見なし , そのまま量子化した値とする.

本手法で閾値 0.03,V=12,N=4 としたときの一例を図 6 に示す.

本手法において N=V ならば, 従来手法で B=2 を設定した場合と同様の処理となる. また, そもそもこれはクロマベクトルを念頭においた処理であるが, 一応, 他の特徴量においても利用可能な二値化手法である. 閾値を用いた方法による悪影響を除去したい場合には有効となる可能性があるが, 特に値の大小について傾向が掴めている場合を除き ,N の値は試行錯誤で決定する事となる.

なお, 本手法における N=V でのヒストグラム全体のビン数 (階級の総数)$L_{N=V}$ は以下の通りである.

$$L_{N=V} = 2^V \tag{2}$$

$N \neq V$ の時は, ベクトル全体で 1 の個数が N 個よりも多くなることがないため, 同じ次元数でも実質的に使用するビン数は減少する. この数 L_N を式で表すと以下の通りである.

$$L_N = \sum_{x=1}^{N} {}_V\mathrm{C}_x \tag{3}$$

表 1　実験機のスペックおよび条件

モデル名	DELL PRECISION M6500
プロセッサ	Core i7 Q720 1.6GHz 8core
メモリ	3.4GiB
OS	Ubuntu 10.10
コンパイラ	GCC 4.5.2(g++)
オプション	-march=native -O3 std=gnu++0x

表 2　作成・比較したデータベース

データベース名	特徴量	N	量子化閾値
mp3_ch.db	Chroma	12	0.15
mp3_mfcc.db	MFCC	12	0.15
mp3_ch_4of12.db	Chroma	4	0.03
mp3_mfcc_6of12.db	MFCC	6	-100

ただし , 閾値を意味のない値 (特徴量が取る事の無い値) に設定した場合, ビンの数は実質的に $\tilde{L}_N = {}_V\mathrm{C}_N$ となる.

4. 評　　価

改良を加えた探索法を , 表 1 のマシンにて実装し , 実験を行った. また表 1 の事項以外に, すべての実験では以下の項目が共通している.

- 連想配列には C++/0x(TR1) における std::unordered_map を使用 (ハッシュテーブルによる実装)
- 特徴量抽出に OpenSMILE ver.1.0.1 を使用
- 全ての特徴量はフレームサイズ 100[msec] フレームステップ 80[msec] の間隔で取得
- クロマ・ベクトルの取得には FFT を使用し , 使用する窓関数は $\sigma = 0.4$ のガウス窓とする. また最低周波数を 55[Hz] とし , それより 6 オクターブ間 (3520Hz まで) で取得する.
- MFCC の取得にはパワースペクトルを用い , 使用する窓関数はハミング窓とする.
- データベース内の曲データは市販 CD からリッピングした mp3 形式 (サンプルレート 44.1kHz, ビットレート 192kbps, ステレオ) のもの 1959 曲を使用
- SN 比は, 信号データと雑音データのそれぞれにおける 2 乗平均 (0 を中心とした分散の平均) の比とする.

さらに, 表 2 に示す通り , データベースは同じ曲に関して四種類のものを作成し , 各々において比較している.

4.1　雑音に対する検索結果の変化

以下の手順で記録したデータを元に, 各種考察を行う.

（ 1） データベースに登録された楽曲から重複を許容して 1000 曲選び出す

（ 2） それぞれにおいてランダムな箇所で 20 秒を選択する

（ 3） 選択された 20 秒の範囲に指定された SN 比で各種雑音を重畳し , それをクエリとして各データベースでの検索を行う.

（ 4） クエリの原曲の順位等の情報を記録する

雑音の SN 比は 100.0,10.0,7.5,5.0,2.5,0.0,-2.5,-5.0,-7.5,-10.0,-12.5,-15.0[dB] という 12 種類の値を取った. また, 雑音の種類は白色雑音および研究室内で取得した会話雑音の 2 種類を用意した.1 曲の同じ地点から切り出したクエリに関して,12 種類の

SN 比 ×2 種類の雑音 ×4 種類のデータベース=96 パターン用意して実験することとなる. なお, 全てにおける探索閾値は 0.25 とした.

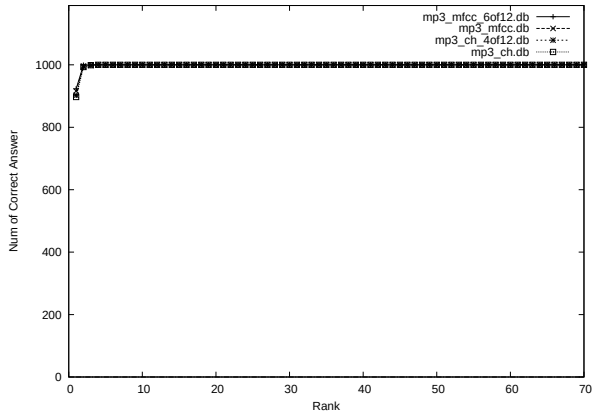

図 7　SN 比 100dB 白色雑音重畳 (雑音重畳なしと同等) での検索結果 (横:順位, 縦:その順位までの正解数)

図 7 は,SN 比 100.0dB における検索結果である. この図は白色雑音重畳時の実験結果であるが, 会話雑音における実験結果と全く変わらなかったためそちらの図は省略する. 実験したクエリは SN 比毎に 1000 個ずつであるため, 正解数そのままの数値でパーミル単位の正解率と読み替えることも可能である.

ノイズ重畳プログラムでは 16bitPCM によって音楽データを扱っているが, これにおけるダイナミックレンジの限界は $20log_{10}(\frac{2^{16}}{1}) \cong 96[dB]$ であるため, 実質的に SN 比 100[dB] においてはノイズが重畳されないと言える. しかし図より, 一切雑音を乗せて無い場合でも原曲が 1 位に出現せず,2 位または 3 位に出現するものが複数個存在することが分かる.

これは, データベース内に同じ曲が 2 個以上存在したためであると考えられる. 把握している限り,2 個存在するものが 78 曲,3 個存在するものが 7 曲,4 個存在するものが 1 曲であった. 同率一位といった場合, 検索結果のトップで名前順によって結果がソートされるなどしてクエリが原曲からそのまま切り出したデータであっても 1 位に原曲が出てこないことがある. 実際, 図 7 における検索結果 1 位に出現しないデータを調べ上げた結果, 全て同じ曲に紛れて 2 位以下となっていた. よってこの結果は正しいと考えられ, ここからどのように順位が変化していくのかを同様の図によって観察する. スペースの関係で全て示せないため, 変化の大きかった SN 比-2.5dB から 2.5dB における図のみを示す.

図 (8) から図 (13) の全てにおいて,SN 比が下がりノイズが大きくなれば各種検索結果において正解楽曲が上位に出てくることも少なくなってくるが, いずれの状況においても N=4 のクロマベクトルによって作成されたデータベースでは他の特徴量よりも優秀な結果が得られている事が分かる. またノイズが変化したときにも特徴量によって異なる傾向の変化が見られるが, いずれのノイズでも結果が良いのは N=4 のクロマベクトルによるデータベースのものである. クロマベクトルにおいて, ベクトル量子化の改良は成果を上げたと言える.

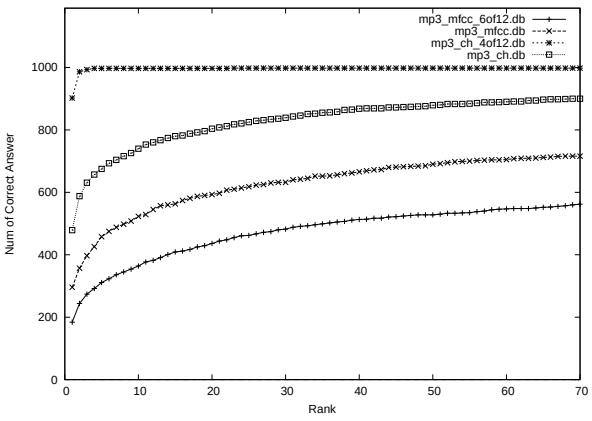

図 8　SN 比 2.5dB 白色雑音重畳時の検索結果

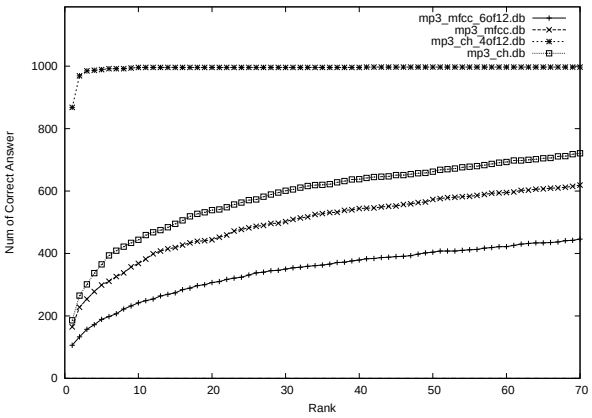

図 9　SN 比 0.0dB 白色雑音重畳時の検索結果

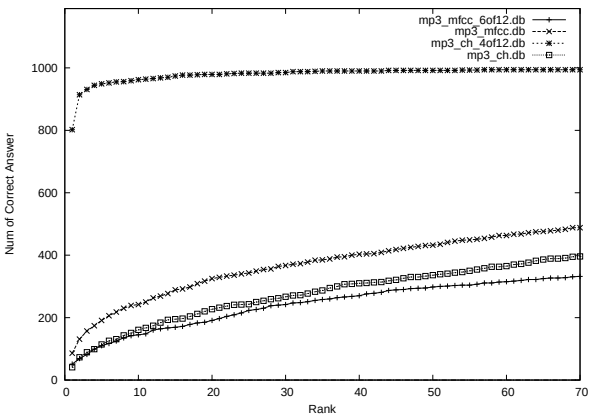

図 10　SN 比-2.5dB 白色雑音重畳時の検索結果

4.2 ヒストグラムの最適化

以下の手順で記録したデータを元に, 各種考察を行う.

（ 1） データベースに登録された楽曲から重複を許容して 1000 曲選び出す

（ 2） それぞれにおいてランダムな箇所で 20 秒を選択する

（ 3） 選択された 20 秒の範囲にをクエリとし, 検索閾値および通常配列と連想配列の使用を切り替えながら各データベースでの検索を行う.

（ 4） クエリの原曲の順位等の情報を記録する

検索閾値 θ は 1.00,0.80,0.60,0.50,0.40,0.35,0.30,0.25,0.20,0.15,

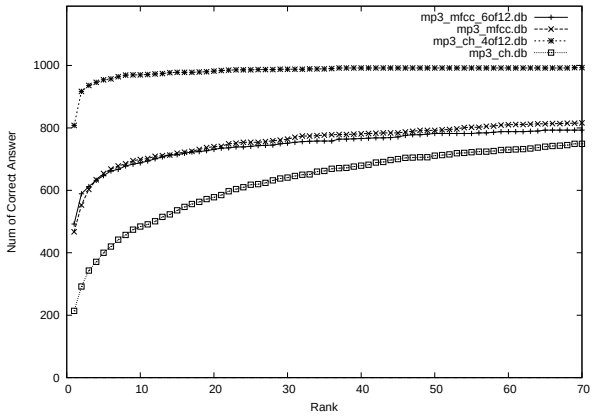

図 11　SN 比 2.5dB 会話雑音重畳時の検索結果

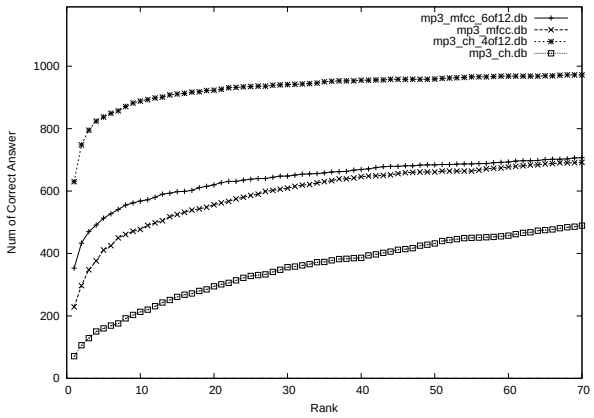

図 12　SN 比 0.0dB 会話雑音重畳時の検索結果

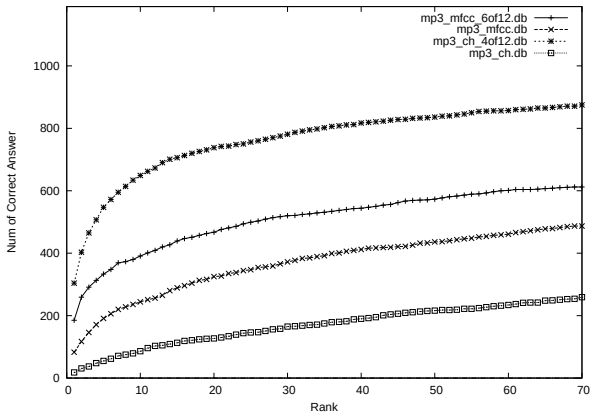

図 13　SN 比-2.5dB 会話雑音重畳時の検索結果

0.10,0.05,0.00 という 13 種類の値を取った. よって同じクエリに関して,13 種類の検索閾値 ×2 種類のデータ構造 ×4 種類のデータベース=104 種類の条件下で実験することとなる.

図 14 は, 連想配列によって検索した場合での, 各検索閾値における配列サイズ (テーブルサイズ) の平均値をプロットし, 最小二乗法によって特徴量毎に線を引いたグラフである. 探索閾値によって値が大きく変動することは無く, また各々の特徴量に対して一定の値を取っている事が分かる.

クエリの長さが 20 秒であり, また特徴量取得の回数が 0.08sec につき 1 回であることを考慮すれば,20 秒で量子化した特徴量

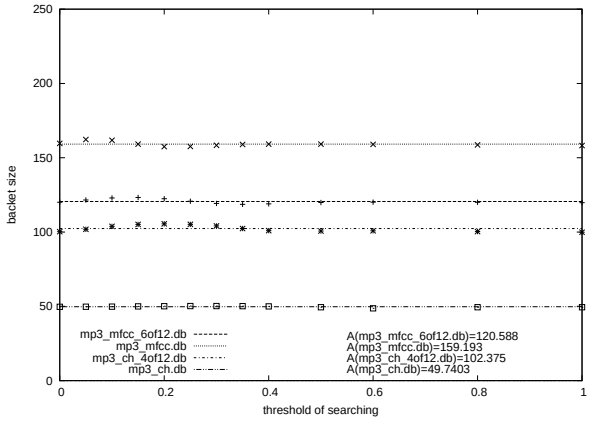

図 14　連想配列による検索時の配列サイズ (横軸:検索閾値, 縦軸:サイズ)

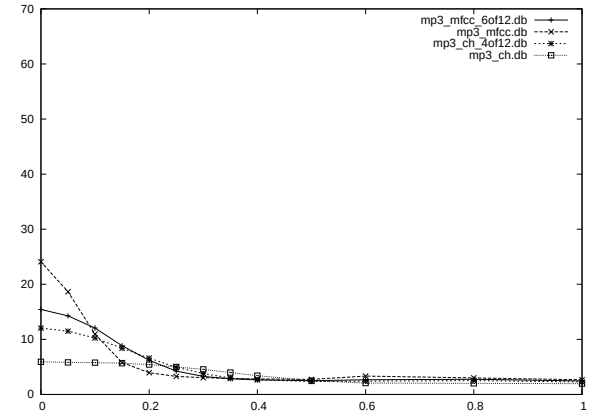

図 15　連想配列使用時の検索閾値と速度 (横軸:検索閾値, 縦軸:実検索時間)

の値は 250 個得られる. テーブルサイズは理論上それが最大値であるが, 実測値ではおよそ 50 から 160 個程度となっていた. 当然, 通常配列で処理した場合には取りうる特徴量値全ての添字に該当する領域を確保しなければならないため, 今回の場合だと 4096 個で固定となる. よって, 単純な配列のサイズで判断した場合には最低でも $\frac{160}{4096} = 0.039 = 3.9[\%]$ 程度のサイズまで, 空間計算量が減少したと言える. 連想配列において, 要素一つに対してキーと値を保存しなくてはならないことや, ハッシュテーブルを構築しなければならないことを考慮しても, 連想配列が要素毎に通常配列の 25 倍のメモリ領域を消費するとは考えにくいため, 実メモリの消費量も十分に削減できていると考えられる.

なお, このテーブルサイズ値は, クエリの長さが変動しないことを考慮すれば, 同じ特徴量 (特徴量ベクトルの量子化後の値) が多く出現すれば低下する事が明らかである. 故にクロマベクトルでは同じ特徴量が多く出現し, MFCC においては様々な特徴量の値が出現したことが分かる.

図 15 および図 16 は, 連想配列及び通常配列における, 各探索閾値での実検索時間の平均をプロットしたものである. 通常配列では探索時間はどの特徴量によっても変わらないが, 連想配列では探索時間が特徴量によって大きく異なる. なお, 別途各し

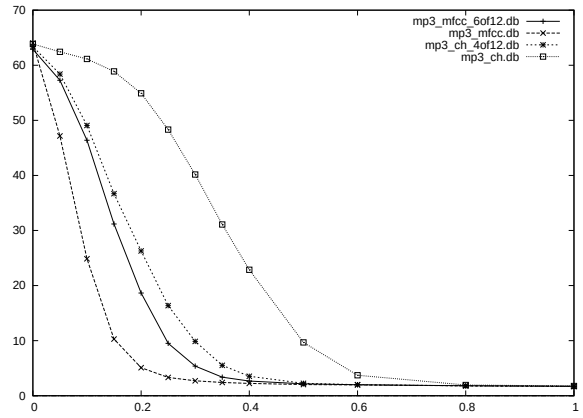

図 16　通常配列使用時の検索閾値と速度 (横軸:検索閾値, 縦軸:実検索時間)

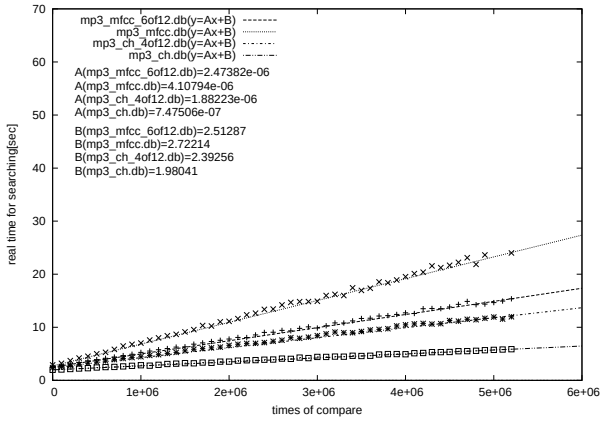

図 17　連想配列使用時の比較回数と速度 (横軸:比較回数, 縦軸:実検索時間)

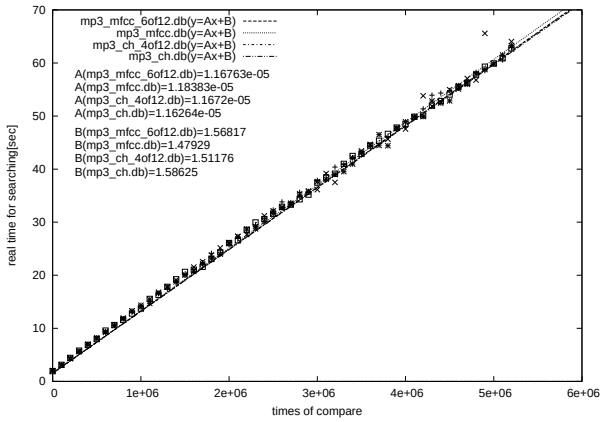

図 18　通常配列使用時の比較回数と速度 (横軸:比較回数, 縦軸:実検索時間)

きい値におけるヒストグラム比較の回数を調べてみると, 連想配列と通常配列で全く同じであった. よってこの探索時間の違いは, 各特徴量における連想配列の大きさが異なり, 重なり率の計算時間等に差異が生じた事が原因であると考えられる. また, 連想配列では閾値を下げても探索速度が遅くなりにくく, 閾値が低い状態ではほとんど通常配列より高速であることが分かる.

ヒストグラム比較回数と実検索時間の関係を確認するため, 連想配列及び通常配列での各特徴量において, 比較回数 10 万回単位でまとめた中における実検索時間の平均値をプロットし, その上に元データから計算した近似直線を引いたグラフが図 17 および図 18 である. 通常配列では各特徴量間の大きな差異は認められない. 特徴量が異なっても配列サイズは変わらず, 重なり率計算時間やその他の処理においても特に変化が無かったからだと考えられる. 一方, 連想配列では特徴量によって明確に異なる. これは, 特徴量によって配列の大きさが変化し, 重なり率等の各種計算量も変化したからであると考えられる. 理論的には配列サイズが大きくなれば各種計算時間も長くなるが, 図 14 から読み取れる配列サイズにおける特徴量の大小関係と, 図 17 から読み取れる実探索速度における特徴量の大小関係を比べると, 実際に一致していることが分かる.

結局, 図 17 と図 18 の近似直線を見比べると分かるように, 比較回数がある程度多くなれば連想配列は通常配列よりも高速となる. ただ連想配列のサイズはクエリの長さや特徴量の性能によってそのサイズが変化しやすく, クエリを極端に長くした場合や特徴量の種類によっては配列の方が高速に検索できる場合もある. また特徴量の次元をこれ以上増やした場合には, 通常配列よりも連想配列の方が高速に探索可能になることが予測される.

5. 類似音楽検索システム

5.1 概　　要

前章までの研究結果を用いて,Web 経由で利用可能な音楽検索システムを作成した.

本サービスはサーバー・クライアント方式で提供される. また近年の動向を鑑みて, 様々なプラットフォームにおけるアプリケーション開発を考慮した入出力 API も搭載している. その上で想定する利用方法は以下のような手順である.

（ 1） 外部で聞こえる音楽を携帯デバイス等を用いて録音する

（ 2） mp3 形式の録音データおよび検索条件を Web ブラウザや各種アプリケーションによってシステム提供サーバーに送信する

（ 3） サーバーから検索結果として類似度が高い曲名のリストを受け取る

本実験結果からある程度雑音に対する頑健性は強い事が分かっているため, 性能のあまり期待できない携帯デバイスにおけるマイクであっても利用は可能であると考えられる.mp3 形式で圧縮されたデータを扱う理由は,Web 経由で音楽データを扱う際の大きな転送量を緩和するためである.

5.2 インタフェース

Web 経由で提供されるシステムなので, 通常の Web ブラウザから閲覧と利用が可能なインタフェースとして Web ページを用意した.

ページは php で記述し, 通常の Web ページ同様に操作できるようにしている. 設定やアップロードファイルの指定を行った後に検索ボタンをクリックすれば, 設定にも依るが数秒後には検索結果として類似度が高い順に曲名が表示される. 曲名お

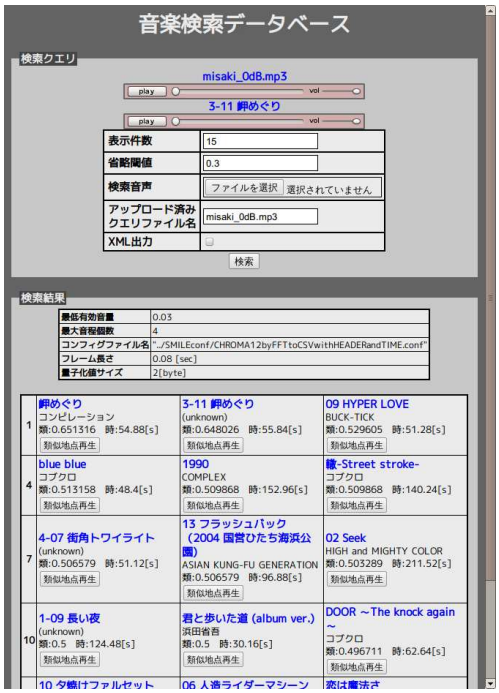

図 19　システムのインタフェースとして作成した Web ページ

よび作者名は, あらかじめ MySQL データベースにファイル名と関連付けて登録したものを用いている. また Flash を用いた mp3 プレイヤーによって各曲における類似地点からの再生を行うことができ, 実際に自分の耳で聞いて正しいかどうかを判断することができる.

5.3　入出力 API

本システムでは Web ページからの利用以外に,XML で登録曲リストや検索結果のデータ等をやりとりすることができる API を備えている. これは, 様々な環境下でのインタフェースを誰もが独自に作成する事を可能にし, 一つの Web ページだけでは対応できない様々な環境に対応するための手段として提供するものである. 典型的には, 携帯デバイス等での利用を考慮したページが表示等の面から考えて通常のパソコンからのアクセスには適さない事や, その逆のケースへの対策が考えられる. また, 録音からデータ転送の一括実行や, 検索結果の保存等といった多彩な機能を持つアプリケーションを作成すれば, より便利で簡単にシステムを利用することが可能となるだろう.

一つのインタフェースで対応できる環境は限られている. また, アプリケーション自体に対する機能の違いも人によって考えが異なる. それ故に, このような形で最も基本的なシステム部分を汎用的かつオープンにしておくことは, 様々なメリットがあると考える.

5.4　考　　察

実際に小さなマイクから録音したもので検索した結果は概ね良好で, 検索速度も大抵の場合 5 秒以内であるため, 個人的には特にストレス無く利用できた. クエリデータとして wav ファイルを用いるシステムも作ってみたが, あまりにもレスポンスが遅く転送量が増加する上にあまり結果が良くなる事もない為, 別途変換の必要があったとしても mp3 を用いたシステムが妥当であると考えられる.

インタフェース部には改良の余地があると思われる. 現状では音楽の録音や mp3 への変換は各ユーザーに委ねられているが, 例えば Java 等を用いれば web インタフェースに録音部分を盛り込むことが可能となるだろう. また, 探索閾値の情報は利用者に対してはよりわかりやすく抽象化されているべき部分であり, 何も考えずに探索閾値を 0 としてしまうような事への対策をとる必要がある. Web 標準インタフェース開発でなく API の提供に注力すると考える場合には, より堅牢で詳細な情報を提供可能なものに改良するべきだろう.

6.　ま　と　め

本論文では, 時系列アクティブ探索法に改良を施し, 音楽検索における検索精度, 検索速度および空間計算量における改善を実験にて示した. 特徴量としてクロマベクトルを選択し, 時系列アクティブ探索法におけるベクトルの量子化方法を工夫することにより, 白色雑音 0[dB] 重畳時にも 90%近くの曲が検索に対して上位 1 位, またほぼ全ての曲が上位 10 位以内に出現し, 他の特徴量と比較しても雑音に対する高い頑健性を得た事が確認できた. またヒストグラムを連想配列によって扱う事により, 空間計算量をクエリ長の線形オーダーとし, 実検索時間の削減にも成功した. これは, 時系列アクティブ探索法において作成されるヒストグラムがほぼ全ての階級で頻度 0 となる疎な状態であった事を利用したものであり, 検索クエリが長くなり過ぎない限りは速度面においても従来手法より優位に立つ. その他, 特徴量の次元数が計算機の消費リソースにほとんど影響しなくなった事で, 特徴量選択の幅が広がったとも言える.

今後の展望としては, より検索に有効な特徴量の選択および利用方法を考え, 多様な種類の雑音に対する一層の頑健性強化を図ることや, 様々な種類の音楽に於いて適応可能であるのかどうかを調べたい. また, 本論文での改良を用いることによって始めて利用可能になるような高次元の特徴量に関して, 実験によって何らかの結果を得ておきたいと考えている. また高次元の特徴量に限らず, 例えば実用される和音の組み合わせだけで特徴量を取ってくるなどの工夫により, 特徴量の改良なども行いたい. また N の値は小さなデータベースでもっともよい結果が出るように試行錯誤で決定したが, 大きなデータベースでも同様な結果となるのかどうかを正確に検証したい. その他, 実用的なシステムとしての使いやすさについても随時検討したい.

文　　献

[1] 柏野 邦夫, ガビン スミス, 村瀬 洋: ヒストグラム特徴を用いた音響信号の高速探索法-時系列アクティブ探索法- (電子情報通信学会論文誌 D-II Vol. J82-D-II No.9 pp.1365-1373 1999 年 9 月)
[2] 後藤 真孝: SmartMusicKIOSK:サビ出し機能付き音楽試聴機
[3] 青木 直史: C 言語で始める音のプログラミング (オーム社)
[4] 荒木 正洋: フリーソフトでつくる音声認識システム (森北出版)
[5] 宮沢 幸希: Miyazawa's Pukiwiki 公開版 (http://shower.human.waseda.ac.jp/ m-kouki/pukiwiki_public/)