大規模音楽データベースのハミング検索システム

小杉尚子[†] 小島明[†]
片岡良治[†] 串間和彦[†]

本稿では，我々が研究開発しているハミングを用いた音楽検索システム（ハミング検索システム）について述べるとともに，このシステムを構築するために採用している技術やパラメータの値について，その効果を定量的に評価する．音楽データベースに対して，音情報をキーにした検索は直感的で非常に有効である．しかし人の歌唱は曖昧で，検索キーとして使用するのは難しい．そこで本システムでは類似検索技術を用いて，ハミングに似ている部分を持つ曲を似ている順にリストアップしたものを検索結果とする．目標は，ハミングされたフレーズを含む曲（正解）を，類似度順の曲名リストの1位に出力することである．従来のハミング検索システムに比べて本システムがきわめて優位である点は，データの処理の基本単位を「音符」ではなく「拍」にしていることと，多次元特徴ベクトルを用いたインデクスを検索に使用していることに起因する．これによって，様々なエラーを含むハミングを検索キーにしても，精度の高い高速な類似検索を実現している．実験では1万余曲を登録したデータベースを構築し，検索時間は約1秒というレスポンスを達成した．また人が聞いて分かるレベルのハミングの約70%については5位以内に正解を出力することを確認した．

A Query-by-humming System for a Large Music Database

Naoko Kosugi,[†] Akira Kojima,[†] Ryoji Kataoka[†]
and Kazuhiko Kushima[†]

A music retrieval system that accepts hummed tunes as queries is described. Technologies and the values of parameters that are used in the system are also described and their effectiveness is quantitatively evaluated. Retrieval using sound information as queries for a music database is intuitive and very useful. However, it is difficult to use hummed tunes as queries because they are often unclear. Thus, the system employs a similarity retrieval technique to overcome this problem. The retrieval result is a ranked list of songs that has a part which is similar to the hummed tune according to the closeness of the match. Our goal for the system is to retrieve the correct song at the top of the song list. The most significant ways in which our system is superior to general query-by-humming systems are that 1) musical data is processed based on “beats” instead of “notes”, and 2) the retrieval is done through the use of indices based on multi-dimensional feature vectors. These features allow the system to retrieve songs quickly and precisely even if erroneously hummed tunes are used as queries. The database currently holds over 10,000 songs, and the retrieval time is about one second. The system is able to recognize the song and rank it within the first five places on the list for about 70% of hummed tunes that are recognizable to human beings as a part of a song.

1. はじめに

性能の良い低価格なパーソナル・コンピュータやネットワークの普及にともなって，マルチメディア・データの使用は広く一般ユーザに浸透している．したがって，大規模で効率的なマルチメディア・データベースの構築，またそのようなマルチメディア・データベースに対する正確で高速な検索方式の確立は重要な課題である．

検索システムといえば，必要な情報に関するキーワード（文字列）を入力（検索キー）とするものが一般的である．しかし，マルチメディア・データに対する検索では，検索キーを文字で表現することが難しい場合が多い．そこで，その解決策の1つとして内容検索（content-based retrieval）に対する関心が高まっている[1)~3)]．

内容検索システムでは，検索キーとしてデータベース内のデータと同じメディアのデータを使用する．たとえば，画像データベースに対しては写真や絵などを，

† NTTサイバースペース研究所
NTT Cyberspace Laboratories

また音楽データベースに対しては人の歌唱などを検索キーにすることができる．内容検索システムは，ユーザが探したいものをより直接的に表現することができるので，マルチメディアデータの検索システムをより使いやすいものにすることができる．

内容検索システムでは，エラーを含んだ曖昧な検索キーが使用される可能性があるので，そのような検索キーを用いたマッチングで，正解だけを的確に検索するのは困難である．したがって，内容検索では一致検索技術よりも，マッチング・スコアを利用した類似検索技術が有効である．類似度の判定方式はいろいろ提案されているが，検索結果は類似度順にソートされたランキング・リストの形で与えられることが多い．ユーザは，類似検索技術を利用することで，必ずしも正確な検索キーを入力しなくとも，提示された複数の回答案のなかから自分が意図していたものを選びだすことができる．

このように，類似検索技術は内容検索を行う場合には欠かせない技術であるが，一致検索と違ってマッチング・スコアを利用するため，検索のコストが非常に大きいので，高速化が重要な課題である．そこで我々は，複数の多次元特徴ベクトルを使った距離計算に基づく多次元空間検索エンジン，*HyperMatch*[4]を研究開発してきた．HyperMatch は VAM Split R-tree[5]に基づく検索エンジンで，この検索エンジンは，木構造インデクスを用いることによって検索速度を高めている．HyperMatch による検索処理では，ノードを枝刈りすることによる効率的な検索を可能にするだけではなく，複数種類の特徴量をそれぞれ別々の多次元ベクトル空間に保持し，クエリの指定する特徴量に応じて，それぞれの特徴ベクトル空間で類似検索を行い，それらの結果を統合して最終的な検索結果を出力することができるという柔軟性も持っている．

本稿では，HyperMatch を利用して，高速で精度の高いハミング検索システムを実現するための技術を提案し，それらの技術を用いている，現在研究開発中のハミング検索システム—*SoundCompass*を紹介する．提案する各技術や採用するパラメータ値は，SoundCompass を用いて，定量的に評価する．

2. 関 連 研 究

音楽内容検索技術の中でも，ハミングなど，人の歌唱を検索キーにする音楽内容検索に関する研究は海外[3),6),7)]より，日本[8)~12)]の方がさかんである．

ハミングを検索キーにすることの最大の難点は，やはり入力としての曖昧さであろう[8)]．たいていの人間は，記憶を頼りに歌う場合，楽譜どおりに正しく歌うことはできない．したがって，ハミングデータには音符の挿入や削除などのエラーが存在することをふまえて，それらに柔軟に対応しなければならない[6)]．一方ハミングするユーザは，曲名や歌手名などがはっきりとは分からないなりにも，ある確実な検索結果（曲名など）を期待している．したがって，ハミングを用いた音楽検索システムとは，入力にエラーが多いわりには，要求条件の厳しい検索システムであるといえる．このような厳しい要求をクリアするために，様々なマッチング方法が提案されている．

たとえばよく用いられるのは，文字列マッチングを応用した手法である．音楽は音符の並びなどで表現され，個々の音符は楽譜上では音価☆と音高の２つの情報を表現している．したがって，１つ１つの音符の情報をある種の文字にあてはめることで，音符の列を文字列で表現することができる．また，文字列どうしを比較することによって，ある曲と別の曲がどのくらい似ているかを調べることができる．この場合文字列には，音高そのものよりも，前の音符との音高の差[9),10)]，あるいは，音高の変化の方向（U(up)，D(down) など）を使用するものが多い[3),6),7),13)]．これは，マッチングの際に，データベース内の曲のキーと入力データのキーの違いに影響されないようにするためと，特に人のハミングを検索キーにする場合には，ハミングに音程のミスや揺らぎが多いことから，できるだけそのようなエラーに左右されない検索を実現するためである．しかし，このような情報だけでは，大規模なデータベースに対して精度の高い検索システムを実現することはできない．そこで，リズム情報や音高を使った検索を，高機能検索として提供しているものもある[6)]．また音長情報を用いたり，音高変化の度合いによってより細かく柔軟に音高の変化の方向を記述する方法[10),13)]も提案されている．

入力キーのエラーを考慮した文字列マッチング問題とは，ある文字列の中から，ある類似度の基準の下で，入力パターンに類似していると判断される部分をすべて見つけ出すことである．入力キーのエラーを考慮した文字列マッチングにおいて，最もよく知られた方法は，ダイナミック・プログラミングによるマッチング（以下 DP マッチングと呼ぶ）である．また最もよく知られた「類似度」は，編集距離（edit distance）と呼ばれるもので，ある文字を別の文字に変換する場合のコストを指す．それぞれのコストは自由に決めるこ

☆ 音符や休符が表す長さ．絶対的な時間の長さではない．

とができるので，編集距離を使った類似度判定は非常に柔軟である．したがって，ハミングを入力キーとする音楽検索のように，曖昧なキーを使い，かつ類似度の算出に自由度が必要な音楽検索システムには有効な手法である．しかし，検索精度を高くするためには，検索キーは長い方が有利であるが，DP マッチングでは，検索対象データの長さを N，入力キーの長さを M としたとき，1 回の検索コストは $O(NM)$ になるので，大規模なデータベースに対する検索には不向きである．

そこで文献 6) では，入力キーのエラーを考慮した高速なテキスト検索方式の 1 つであるステート・マッチング[14]を音楽検索に適用することを試みている．ステート・マッチングでは簡単なビット演算のみが行われるので，大規模なデータベースでも高速に検索できる．しかし編集距離のように，各々のエラーに対してそのコストを自由に設定することができないので，適切な類似度を計算するのは難しいという問題がある．

検索速度という観点からは，やはり大規模なデータベースの検索システムにはインデクスの利用が有効である．文献 15) ではあらかじめ DP マッチングでインデクスを作ることにより検索の柔軟性と高速化の両方を実現している．

3. 課題と我々の解決方針

ハミング検索システムでは，ユーザは楽譜を見ながらしっかりハミングするわけではないので，データベース中の音楽（楽譜データ）とハミングとのマッチングをする際に，以下のような曖昧さを考慮しなければならない．

問題 1 ユーザが曲のどこから歌い出すか分からない 一般的には，曲の出だしやサビから歌い出す場合が多いが，つねに誰もがそうであるとは限らない．また，たとえそうだとしてもサビの部分を自動的に検出する技術もまだ確立されていないので，ユーザがハミングする部分をあらかじめ特定することはできない．

問題 2 音高が正しいとは限らない 音高が正しくないとは，ハミングのキー（調）が楽譜と違う（調の違い）という意味と，歌っている間に正しくない音高が出現するという意味がある．また，後者も単発的に正しくない音高が現れる場合と，歌っている間にだんだん音高が下降/上昇するという場合がある．

問題 3 テンポが正しいとは限らない どんなにテンポをしっかりとりながら歌える人でも，何のガイドもなしに，厳密な意味で一定のテンポで歌うのは難しい．これも「問題 2」と同じように，ある音を伸ばしすぎたり，あるいは伸ばしきれなかったりという単発的なテンポミスの場合と，徐々に早くなっていったり遅くなっていったりするという継続的なテンポミスの場合がある．

問題 4 ハミングを採譜した結果が正しいとは限らない 抽出されたピッチや有音/無音区間の判定が正しいとは限らない．よく知られているように，人の声からのピッチ抽出に関して，いまなお決定的な方式が確立されていない[16]．特に歌唱の場合は会話と違って短時間でピッチが激しく変化するのでさらに難しい．

SoundCompass では，問題 1 は，メロディデータを細かく分割してデータベースに登録し，歌い出しの自由度を増すことで解決する（6 章参照）．

問題 2 の「調の違い」に関しては，マッチングの際に音高ではなく，ある音を基準にした相対的な音高を用いることで解決する（7.1 節参照）．

問題 3 には，メトロノームを導入して，一定のテンポでのハミングを支援することで対応する．

問題 4 には，直前/直後の音符との音高の差を利用した誤抽出ピッチの修正法を提案する（8 章参照）．

またこれらの問題は，たとえばハミングに余計な音符が挿入されたり，必要な音符が削除されたり，また，ある音符が別の音高の音符に置換されたり，1 つの音符が複数の音符に分割されたり，逆に複数の音符が 1 つの音符に結合されたり，といった音符レベルのエラーとなって現れる．したがって，音符をベースにデータを処理する場合，2 章で述べたように DP マッチングなどエラーを考慮したマッチング技術を使用することになるが，長い入力キーを用いて，大規模なデータベースに対して高速な検索を実現するのは難しい．

そこで我々は先に述べた多次元空間検索エンジン HyperMatch をベースに，多次元空間インデクスを使用した高速なハミング検索システムを実現するアプローチをとる．多次元空間検索エンジンをハミング検索に応用するうえでの最大のポイントは，どのような特徴ベクトルを定義・利用するかである．また音符レベルのエラーの影響を軽減するために，従来最もよく使われてきた音符ベースのシステム構築を離れ，拍ベースのシステム構築を提案する．すなわち,「拍」や「メトロノームの打拍」を単位に音楽データやハミングデータを処理する．これは，曲をどんなに早く演奏してもゆっくり演奏しても 1 拍は 1 拍であるという事実に着目した結果[17]である．

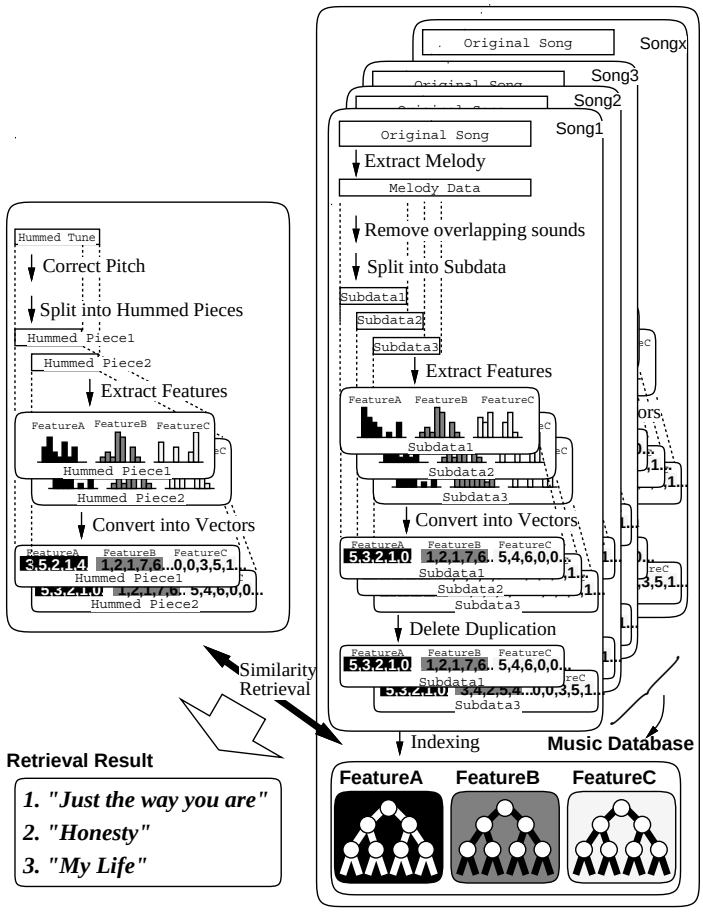

図 1 データベース構築過程とハミング処理過程，およびマッチング
Fig. 1 Steps in music database construction, hummed tune processing and matching.

4. SoundCompass

本章では，我々が研究開発中のハミング検索システム SoundCompass におけるデータベースの構築やハミングの処理について処理の流れを説明する（**図 1** 参照）．

4.1 データベースの構築

データベースには，MIDI 形式のメロディデータを使用しており，現在 10,069 曲を登録している．10,069 曲の中には様々な分野の曲が含まれている．童謡や民謡のように短くて単純な曲もあるし，複雑なポップスやロックなども含まれている．日本はカラオケがさかんなので，最新のヒット曲を含む多数の MIDI データを入手することができる．

音楽データベースを構築するために，まず最初にメロディデータを抜き出す．ほとんどのカラオケデータでは，単一のチャネルにメロディデータが格納されているので，メロディデータだけを抜き出すのは容易である．人は，曲をメロディで認識することが多いので，現在はメロディのみを検索対象としてデータベースに登録している．

次に，同時に 2 つ以上の音が存在する区間に対して，1 つの音だけを残して後を削除する処理を行う．この段階では，すでに伴奏データに見られるような複雑な和音は存在せず，たとえば演奏効果などを狙って前の音と次の音が若干重なっているという程度のものしかない．しかし，1 人で行うハミングでは，同時に 2 音以上を発声するのは難しいので，本処理でハミングとデータベース中の曲の特徴量抽出上の整合性をとる．具体的には，後から出現する音を優先して残す．すなわち，ある音 A が鳴っている最中に別の音 B も鳴り始めた場合，B の音が鳴り始めた時点で A の音は鳴り終わったものとして，2 音の重複区間を排除する（**図 2** 参照）．2 つの音がまったく同時に鳴り出す場合は，和音と考えられる．この場合は，和音が鳴ったとき優勢に聞こえる高い方の音を選択する．

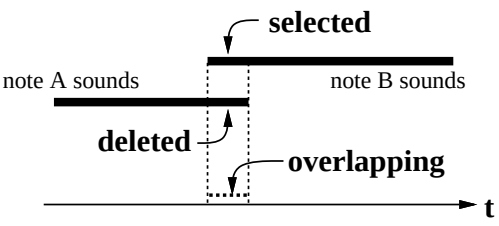

図 2 音の重なりの排除
Fig. 2 Removing overlapping sounds.

次にメロディデータを細かく分割する（6 章参照）．分割されたデータを,「部分音楽片（subdata）」と呼ぶ．

各部分音楽片からは特徴を抽出し（7 章参照），多次元の特徴ベクトルに変換する．

最後にすべての特徴ベクトルをロードして，特徴量ごとに多次元空間インデクスを作成する．

4.2 ハミングの処理

本節では，ハミング検索キーの作成方法について述べる．検索キー作成のほとんどの過程は 4.1 節で述べた音楽データベースの作成過程と変わらないが，人のハミングのように曖昧な情報から検索キーを生成しなければならないので，修正が必要となる．

4.2.1 ハミングの仕方

ハミングはマイクを通して入力し，市販の採譜ソフト[18]を用いて MIDI に変換する．ここでいうハミングとは，いわゆる一般的なハミングではなく,「タ」を使ってはっきり歌ったものである．これはできるだけハミングを正しく採譜するためで，特に「タ」は，発音の前に完全な閉鎖区間が存在する無声破裂音（t）と，広い周波数領域わたってエネルギーがバランスしている中舌母音（a）の組合せ[19]なので，採譜するという目的に対して最も効果的である.「ダ」や「ラ」を使ったハミングも採譜してみたが，精度はあまり良くなかった．

また，メトロノームに合わせてハミングする必要がある．これは，人は何のガイドもなしに一定のテンポで歌い続けるのは困難なのでそれを支援するためと，SoundCompass では「拍」と「メトロノームの打拍」を処理の単位として使用しているので，ハミングがどのテンポで歌われたのかを把握する必要があるからである．ただし，メトロノームのテンポはユーザの歌いやすい速さに調節してかまわない．

4.2.2 検索キーとしての修正と整形

最初に，MIDI に変換されたハミングデータに対して，誤抽出ピッチの修正（8 章参照）を行う．

次にハミングデータを細かく分割する．分割されたデータを「部分ハミング片（hummed piece）」と呼ぶ．

その後，部分音楽片から抽出されたものと同種の特徴量を各部分ハミング片からも抽出し，特徴ベクトルを作成する．マッチングはこの特徴ベクトルを用いて行う．

5. 類似度の算出と類似検索

SoundCompass ではハミングから生成される特徴ベクトルに似ているベクトルを，データベース内の各特徴ベクトル空間から探し出す．ウィンドウ長より長いハミングを収録した際には，複数の部分ハミング片が生成されるが，検索にはその中の中央部分の部分ハミング片 1 つを使用する．すなわち，検索に使用するのは，総部分ハミング片数を h_n としたとき，$\left\lceil \frac{h_n+1}{2} \right\rceil$ 番目の部分ハミング片のみである．このようにハミングデータの中央部分を検索キーとして使用するのは，歌い始めや歌い終わりよりも，ハミングが安定している部分だからである．

部分音楽片と部分ハミング片との類似度は，各特徴ベクトル空間で特徴量ごとの類似度を計算した後，それらの重み付き線形和の形で算出する．すなわちある部分音楽片（M）とある部分ハミング片（H）との類似度 S は，特徴量が N 種類の場合，i 種類目の特徴量による M，H の特徴ベクトル（次元数 d_i）を $M_i = (m_{i,1}, m_{i,2}, \cdots, m_{i,d_i})$，$H_i = (h_{i,1}, h_{i,2}, \cdots, h_{i,d_i})$，重みを α_i とすると，

$$S = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{d_i} |m_{i,j} - h_{i,j}| \right) \quad (1)$$

で表される．SoundCompass では，距離計算にはシティブロック距離を使用しており，ベクトル間の距離が小さければ小さいほど，それらが似ているとする．

最終的な検索結果は，ハミングしたメロディに似ている部分を持つ曲名の，類似度順のランキングリストとして表示する．

6. スライディング・ウィンドウ方式を用いたデータの分割

3 章の「問題 1」の解決策として，ユーザが曲のどこを歌っても検索できるように，曲のデータをスライディング・ウィンドウ方式[20]で分割する（**図 3** 参照）．分割の際に，スライド幅をウィンドウ長より短くすることで，連続する各部分音楽片はお互いに重なりのある冗長なデータになり，ハミングする部分に関していっそう自由度が増す．

同様にハミングデータもデータベース作成時のウィンドウ長とスライド幅で，スライディング・ウィンド

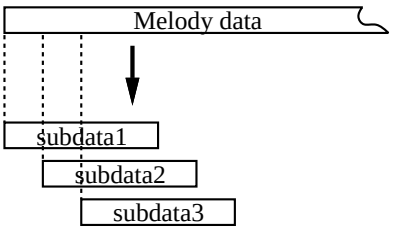

図 3 スライディング・ウィンドウ方式によるメロディデータの分割
Fig. 3 Melody data splitting with the sliding-window method.

ウ方式で分割する．

このように，曲のデータを細かく切り刻んで，それぞれを類似度演算の対象にすることで，「ハミングに似ている部分を持つ曲を似ている順にリストアップするハミング検索システム」を構築することができる．また，データベース内の各データが保持している情報量と，各部分ハミング片から作られるデータが保持している情報量が「拍」と「メトロノームの打拍」を単位に等しくなるので，効率的な類似度演算が可能になる．

ウィンドウ長と検索精度の関係は 9.2 節で定量的に評価する．

7. 特徴量と特徴ベクトル

本章では，各部分音楽片や各部分ハミング片から抽出する特徴量と，作成する特徴ベクトルについて述べる．目標は，できるだけ少ない特徴量データで，高い精度で正解を検出できるようにすることである．

7.1 音高推移特徴ベクトル

人間は，ある曲の一部をハミングで聞かされたとき，自分がすでに知っている曲ならば，多少音程が外れていたりテンポが一定でなくても，たいていはその曲の曲名をあてることができる．このことから，人間は絶対的な音高やテンポではなく，音高の時間的な遷移を曲の特徴としてとらえていると考えることができる．したがって，本稿でも曲の特徴を表現するためにある音を基準とした音高差の時間的な変化を表す特徴ベクトルを考案する．「音高差」とは，ある音とある音の音高の差で，最小単位を半音とし，同音高を 0，半音高い音を +1，半音低い音を −1 と表現する語と定義する．このような音高差の時間的推移を表す特徴量を音高推移特徴量と呼び，生成されるベクトルを音高推移特徴ベクトル[12]と呼ぶことにする．

音高推移特徴ベクトルは，連続するある一定の拍（拍粒度）内の代表音の音高の列で表す．これが我々のシステムを「音符ベース」ではなく「拍ベース」と表現する由来である．ベクトルの各要素は単位拍（拍粒度）ごとの優勢（代表）音をとる．したがって，単位拍の代表音に吸収される範囲の速い音価を持つ音符によるエラーはベクトル生成に影響を与えない．代表音はその単位拍内で最も長く出ている音とする．

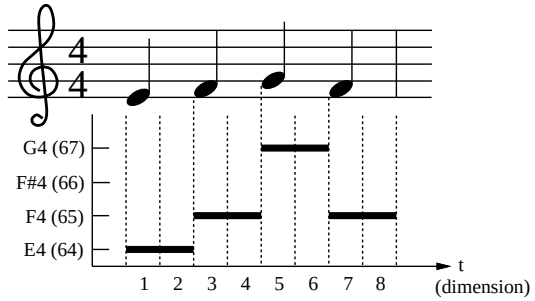

図 4 音高推移特徴ベクトルの生成
Fig. 4 Generating a pitch-transition feature vector.

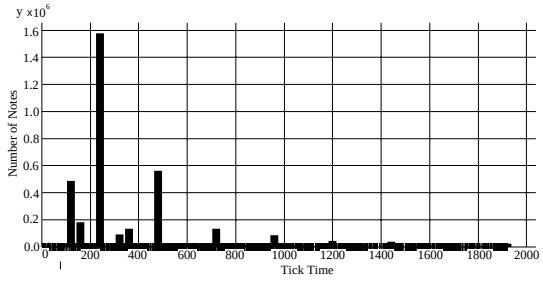

図 5 全曲中の音価の分布
Fig. 5 Note length distribution of all songs.

図 4 は，あるメロディの音高推移特徴ベクトルの生成イメージを表している．E4，F4，G4 は音コードで，その右の数字は MIDI の音高番号である．したがって，たとえば拍粒度を 8 分音符にした場合，この 4 拍の旋律からは 8 次元の特徴ベクトル（64, 64, 65, 65, 67, 67, 65, 65）を作ることができる．また，ある音（基準音と呼ぶ）を基準とした音高差をベクトル値にすると，データベース内の楽譜と異なるキーでハミングされた場合にも対応することができる．これにより，3 章の「問題 2」のハミングと楽譜のキーの違いによる問題を解決できる．たとえば，上記のメロディの中で最も頻出している音（F4(65)）を基準音とし，その他の音を基準音との音高差で表現すると，(−1, −1, 0, 0, 2, 2, 0, 0) という 8 次元の特徴ベクトルを生成することができる．

7.2 音高推移特徴量抽出の拍粒度について

我々は音高推移特徴ベクトルの効果的な拍粒度について検討するために，全 10,069 曲中のすべての音符の音価の分布を調べた．**図 5** に結果を示す．横軸はティックタイムを表しており，縦軸は音符数を表して

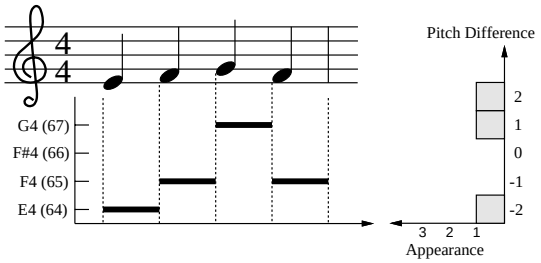

図 6 音高差分布特徴ベクトルの生成
Fig. 6 Generating a pitch-difference distribution feature vector.

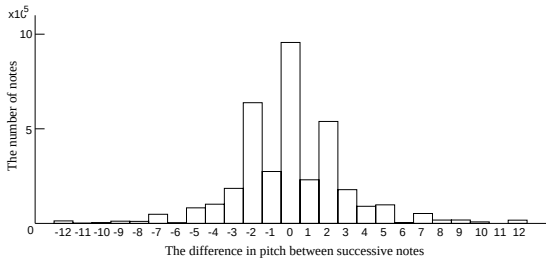

図 7 直前の音符との音高差の分布
Fig. 7 Distribution of pitch-difference between successive notes.

いる．全 10,069 曲において 4 分音符が 480 ティックタイムで表記されているので，図 5 は楽曲中で最も頻繁に使用される音価は 8 分音符（240 ティックタイム）であるということを示している．

図 5 の結果に基づいて，音高推移特徴ベクトルの最適な拍粒度は 9.3.1 項で定量的に評価する．

7.3 音高差分布特徴ベクトル

本節では，7.1 節とは別の尺度から特徴を抽出する音高差分布特徴ベクトル[20]について述べる．効果的な数種の特徴ベクトルを使用して，効率的に正解を限定するのが狙いである．

本稿では，直前の音符との音高差の分布（ヒストグラム）を使用する．**図 6** は，図 4 でも用いたメロディの音高差のヒストグラム作成イメージである．このヒストグラムから，たとえば音高差が −2 の音符の出現回数を 1 次元目の値，音高差が +2 の音符の出現回数を 5 次元目の値とし，各音高差の出現回数をカウントすると，(1,0,0,1,1) という 5 次元の特徴ベクトルを生成することができる．

以上，本節で提案したそれぞれの特徴量の検索に与える効果については，9.3.2 項で定量的に評価する．

8. 誤抽出ピッチの修正

ハミングの誤採譜が検索システムの精度に与える影響は大きい．多くの場合，特徴量の改良などではカバーできないからである．そこで，誤採譜の傾向をつかみ，それらを修正するのは検索システムの精度を上げるのに有効であると考えられる．

本システムで使用している採譜ソフト[18]では，発声したはずの音高の 1 オクターブ下の音としてピッチが誤抽出されてしまう現象が多く観察された．そこで，この 1 オクターブ下がる誤抽出ピッチの修正方針を決定するために，直前の音符との音高差の分布を調べた．

一般的には，大部分の曲において隣接音どうしの音高はあまり激しく変化しないことが知られている．実際に隣接する音符どうしの音高差を，全 10,069 曲の MIDI データについて調べてみると**図 7** のような分布になった．横軸は直前の音符との音高差を表しており，縦軸は音符数を表している．MIDI で音高を表すノートナンバは，半音の違いは数値で 1 の差になるので，+12 は 1 オクターブ上を，−12 は 1 オクターブ下を意味する．

図 7 から分かるように直前の音符との音高差は 0，−2，+2 に集中している．これは，ある音符は直前の音符に対して同じ高さか，あるいは全音高いか低いかの，いずれかの音高の音符であることが多いということを意味している．また，約 97.5%の音符の音高差が −12 から +12 の間にあることも分かった．それ以上の急激な音高の変化はフレーズが切り替わるときなどに起きているが，ゲームのつもりでハミング検索を利用するなどではない限り，ユーザがフレーズをまたがってハミングすることは少ないと考えられる．よって，ある音符 A の次の音符 B の音高は，たいていの場合 A の音高そのもの，または A の音高 ±2 の音高のはずなので，直前の音符との音高差が −12 − 2 から −12 + 2 の間にあり，かつ直後の音符との音高差も −12 − 2 から −12 + 2 の間にある音符は，ピッチの誤抽出の可能性が高いと考えられる．本稿では，直前/直後の音符との音高差が上記の範囲にある音符を，1 オクターブ上の音高の音符に修正するという方法で誤抽出ピッチの修正を試みる．

この方式の有効性は 9.4 節で定量的に評価する．

9. 実験結果と考察

本章では SoundCompass を用いて，6 章から 8 章で考案した各技術の効果を定量的に評価する．あわせて，インデクスを用いた検索による検索速度評価も行う．

9.1 実験環境

実験では，10,069 曲を登録したデータベースを用い

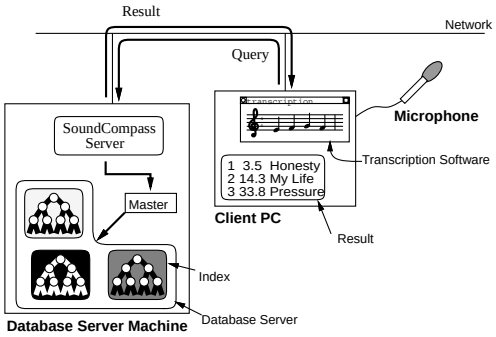

図 8 実験システム構成
Fig. 8 Experimental system structure.

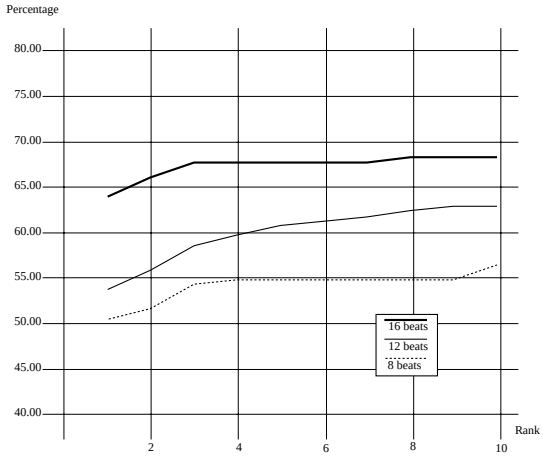

図 9 ウィンドウ長が 8，12，16 拍の場合の検索精度
Fig. 9 Precision for window lengths of 8, 12, and 16 beats.

て精度評価・検索速度評価を行った．

精度評価では，25 人（男 21，女 4）の被験者から 258 のハミングデータを収集し，そのうち人が聞いて曲名が分かるレベルの採譜結果 186 ハミング（72%）を本評価実験用検索キーとして採用した．186 のハミングはすべてデータベースに登録されている曲の一部である．被験者の中には合唱部に属している者も数人含まれている．

検索速度評価ではデータベースサーバマシンとして，Sun Ultra80（UltraSPARC-II 450 MHz×4，主記憶 4 GB）1 台を使用した．

図 8 に本実験のシステム構成を示す．ネットワーク経由でデータベースサーバマシンとクライアント PC がつながっている．サーバマシン側には，SoundCompass サーバ（SoundCompass Server），検索要求を受け付けるマスタ（Master）とデータベースサーバ（Database Server）が動いている．データベースサーバは内部に特徴量ごとのインデクスを保持している．クライアント PC にはマイクがつながっていて，ハミング検索用 GUI（図中には検索結果例を示した）と採譜ソフトが動いている．

マイクから入力された歌声は，採譜ソフトで MIDI に変換されてサーバに送られる．サーバではハミングを 4.2 節に記載した方法で処理し，クエリをマスタ経由でデータベースサーバに送信する．データベースサーバは検索終了後，結果を SoundCompass サーバに返送し，SoundCompass サーバは検索結果を整形加工して，クライアント PC に返信する．

9.2 ウィンドウ長の最適化

スライディング・ウィンドウ方式による音楽データの分割に関して，ウィンドウ長が検索精度に与える影響を調べるために，ウィンドウ長が 16 拍の場合と，12 拍の場合と，8 拍の場合の検索精度比較実験を行った．スライド幅は 4 拍とした．検索には，拍粒度が 8 分音符の音高推移特徴ベクトル（7.1 節参照）を使用した．結果を**図 9** に示す．太い実線はウィンドウ長が 16 拍の場合を，細い実線はウィンドウ長が 12 拍の場合を，点線はウィンドウ長が 8 拍の場合を示している．この図は，ある順位までの正解率を示しており，横軸は順位，縦軸はその順位以内に正解が検出された曲数の割合を表している（図 10〜13 も同様）．たとえば，ウィンドウ長を 16 拍にすると，186 ハミングの約 66%について，正解を 2 位以内に検出できたということを示している．

実験結果より，ウィンドウ長が長くなるほど情報量が増えて検索精度が良くなることが分かる．しかし，ウィンドウ長は最短ハミング長であり，16 拍を超えるハミングは被験者から「長すぎる/辛い」と不評だった．よって，ハミング検索システムという観点では，16 拍程度が限界だと思われる．

一方過去の実験から，データベースに登録されている曲数が少なければ，ウィンドウ長が 8 拍程度でも十分に高い検索精度を達成できることが分かっている[21]．このことから，データベースの曲数が増えるにつれて，短いハミングでは正解を限定するのが困難になることも分かった．

9.3 特徴ベクトルと検索精度の関係

本節では，7 章で提案したそれぞれの特徴ベクトルが検索精度に与える効果を定量的に評価する．また，9.2 節の結果をうけて，本節以降の評価試験で使用するデータベースとハミングデータはすべて，ウィンドウ長は 16 拍で分割した．スライド幅は 4 拍とした．

9.3.1 音高推移特徴ベクトルの拍粒度の最適化

図 5 より全曲において最頻出するのは 8 分音符であ

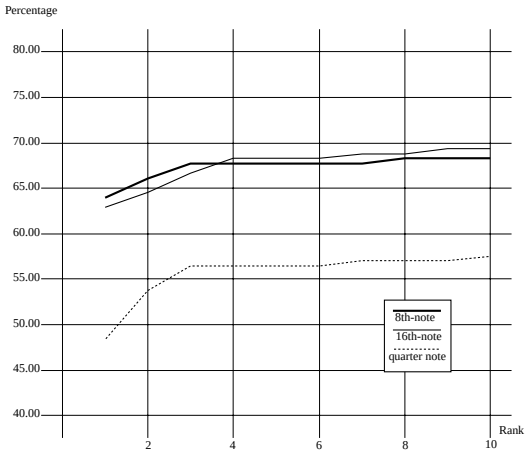

図 10 音高推移特徴ベクトルの拍粒度が 4 分音符，8 分音符，16 分音符の場合の検索精度

Fig. 10 Precision where the beat-resolution for pitch-transition feature vectors are a quarter note, 8-th note and 16-th note.

ることが分かった．そこで，音高推移特徴ベクトルのための効果的な拍粒度を決定するために，検索に使用する特徴ベクトルの拍粒度が 4 分音符の場合と，8 分音符場合と，16 分音符の場合の検索精度比較実験を行った．結果を**図 10** に示す．太い実線は拍粒度が 8 分音符の場合を，細い実線は拍粒度が 16 分音符の場合を，点線は拍粒度が 4 分音符の場合を示している．

実験結果より，特徴ベクトルの拍粒度は 4 分音符より 8 分音符の方が検索精度が良くなることは明らかである．これに対し，拍粒度が 16 分音符の場合と 8 分音符の場合とでは精度にあまり差がない．しかし，拍粒度が倍になると特徴ベクトルの次元数も倍になるのでデータベースのサイズも大きくなるし，類似度演算のコストも増大する．したがって，この程度の精度の差であれば，音高推移特徴ベクトルの拍粒度としては 8 分音符を使用するのが適切であると考えられる．

9.3.2 複数種類の特徴ベクトルの組合せ効果

ここでは，音高推移特徴ベクトルと音高差分布特徴ベクトルを，それぞれ単独で検索に使用した場合と，組み合わせて検索に使用した場合について検索精度比較実験を行った．音高推移特徴ベクトルの拍粒度は 8 分音符とした．よって音高推移特徴ベクトルの次元は $16 \times 2 = 32$ 次元である．音高差分布特徴ベクトルに関しては，直前の音符との音高差が 1 オクターブ（±12）を超える場合は，1 オクターブとしてカウントした．これは 8 章での調査より，直前の音符との音高差は約 97.5%が 1 オクターブ以内であることによる．したがって，本稿で使用した音高差分布特徴ベクトル

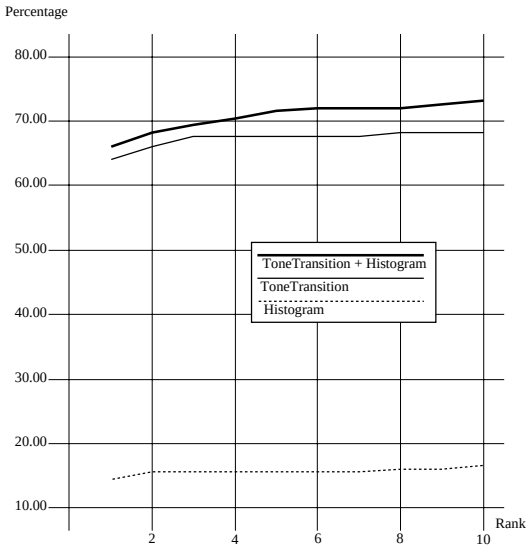

図 11 各特徴ベクトルを単独で使用した場合と，それらを組み合わせて使用した場合の検索精度

Fig. 11 Comparison of retrieval precision obtained using each feature vector and a combination of the vectors.

の次元は $12 + 1 + 12 = 25$ 次元である．また本実験では，特徴量ごとの類似度の和をとる際に使用する重み（式 (1)）の値は，すべて 1.0 とした．結果を**図 11** に示す．太い実線は音高推移特徴ベクトルと音高差分布特徴ベクトルを併用した場合を，細い実線は音高推移特徴ベクトルのみを使用した場合を，点線は音高差分布特徴ベクトルのみを使用した場合を示している．

音高差分布特徴ベクトルは，音楽の特徴を大まかにとらえる特徴量と考えられ，単独で使用したのではあまり検索精度は高くない．一方，音高推移特徴ベクトルは，単独でもかなり高い検索精度を実現しているように見えるが，複数の同じ類似度を持った検索結果を出力する場合（同率 1 位）もあった．これに対し，音高推移特徴ベクトルと音高差分布特徴ベクトルを併用すると，検索精度が向上することが分かる．

以上より，音高推移特徴ベクトルと音高差分布特徴ベクトルの 2 種類の特徴ベクトルを組み合わせて使用すると，検索精度を向上させるうえで有効であるといえる．

9.4 誤抽出ピッチの修正の効果

ここでは，8 章で提案した誤抽出ピッチの修正処理の効果を調べるために，ハミングデータに対してピッチ修正処理をした場合と，ピッチ修正処理をしない場合とで検索精度比較実験を行った．特徴量は拍粒度が 8 分音符の音高推移特徴ベクトルを使用した．結果を**図 12** に示す．実線はハミングデータにピッチ修正処

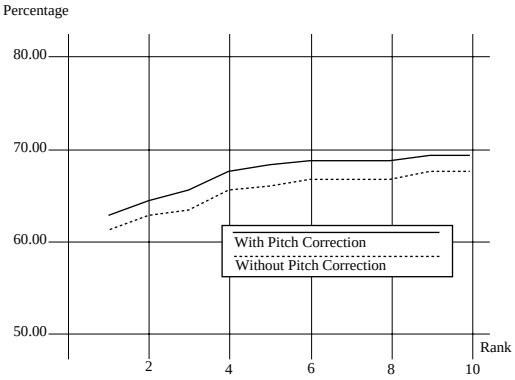

図 12 誤抽出ピッチを修正した場合としなかった場合の検索精度
Fig. 12 Precision where the pitch correction has/has not occured.

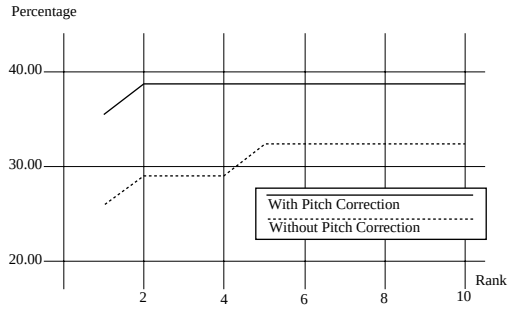

図 13 騒音環境下で収録したハミングに対して，誤抽出ピッチを修正した場合としなかった場合の検索精度
Fig. 13 Precision where the pitch correction has/has not occured for hummed tunes recorded in a noisy place.

理を行った場合を，点線はピッチ修正処理を行わなかった場合を示している．

また，ピッチの誤抽出は，周りの騒音が増えるにつれて増加する傾向があるので，本精度評価で用いたハミングデータとは別に，騒音環境下で収録したハミングデータを用いた精度評価実験も行ったので，その結果もあわせて簡単に報告する．

本実験用に，背後で大きな音で CD を再生しながら 31 のハミングデータを収集した．検索精度比較実験の結果を**図 13** に示す．実線はハミングデータにピッチ修正処理を行った場合を，点線はピッチ修正処理を行わなかった場合を示している．

この 31 個のハミングデータの採譜結果には全部で 1,492 個の音符があり，この中で明らかに 1 オクターブ下の音にピッチを誤抽出した考えられる音符の数は，85 個（5.7%）であった．この 85 個のピッチを誤抽出された音符に対し，本稿で提案している方式でピッチを修正できたのは 46 個（54.1%）であった．逆に，誤

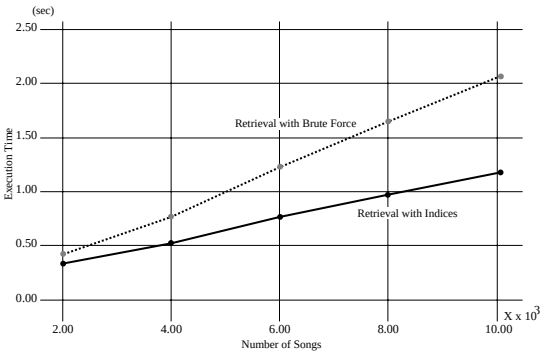

図 14 検索にインデクスを使用した場合と使用しなかった場合の検索時間
Fig. 14 Retrieval time evaluation where the indices are/are not used.

修正してしまった音符は 1 個（1.18%）で，これは騒音を誤って採譜してしまった音符に対して行われた．

全体的に検索精度が大きく低下しているのは，このように CD 再生による騒音も採譜されてしまったことが原因である．これら 31 のハミング・データの中で人間が聞いても曲名が分からない採譜結果はなかったが，マイクを ON にしてから実際にハミングするまでの間に誤採譜された，騒音による音符が特に検索精度に悪影響を及ぼしていた．

これらの実験結果より，ピッチ修正処理を行った方が検索精度が良くなり，また本方式による誤修正の危険性は低いので，8 章で提案したピッチ修正方式は有効であるといえる．

9.5 検索速度について

本節では，検索にインデクスを用いた場合と全特徴ベクトルに対する逐次照合した場合の検索時間を計測し，インデクスの効果を調べる．検索時間の結果を**図 14** に示す．実線はインデクスを用いた場合を，点線は全特徴ベクトルに対する逐次照合を行った場合を示している．横軸はデータベースのサイズ（曲数），縦軸は検索時間を表している．検索時間を測定するための，登録曲数が少ない各データベースは，ランダムに選択した曲を用いて作成した．検索時間として，ハミングによる検索キーがデータベースサーバに送信されてから，検索終了後，データベースサーバからの回答を受信するまでを測定した．

実験結果より，データベースのサイズが大きくなればなるほど，インデクスを使用した方が効率良く検索でき，10,000 曲を超えてもデータベースサーバは約 1 秒で検索結果を出力できることが分かる．

10. まとめ

本稿では高速で精度の高い，ハミングによる音楽検索システム SoundCompass について述べるとともに，このシステムを構築するために採用している技術やパラメータの値についてその効果を定量的に評価した．

現在，データベースは 10,069 曲を保持しており，検索システムは約 1 秒で検索結果を出力する．このシステムでは，曲のどの部分をハミングしても曲名を検索することができる．精度は，人が聞いて曲名が分かるレベルのハミングの約 71%について，正しい曲名を 5 位以内に出力することができる．もちろん正しい曲名だけではなく，ハミングした部分を含んだ曲を一部に含むメドレーも検出することができる．

我々はこの性能を実現するために，拍ベースの音楽データ処理を提案し，多次元特徴ベクトルによるインデクスを使用した高速類似検索技術を導入した．さらに上記の技術をベースに，メロディデータをスライディング・ウィンドウ方式で分割してデータベースに登録することで，自由な歌い出しによる検索を可能とした．また，音価の分布や隣接する音符との音高差といった楽曲データ解析から得た知見をもとに，拍粒度が 8 分音符の音高推移特徴ベクトルや採譜時の誤抽出ピッチの修正方式，また 2 種類の特徴ベクトル（音高推移と音高差分布）の組合せによる検索などを考案し，それらの有効性を定量的に評価した．

11. 今後の課題

我々はこのシステムが，高速に，高い精度で，メモリ使用量がより少ないデータベースで，使いやすく，正解曲を検出できるように現在も研究を進めている．

今後の課題として考えられることは，さらに有効な特徴量の実装といった技術的なことから，音楽の類似性とは何かといった根本的なことまで，きわめて幅が広いが，ここでは特に技術的な観点から今後の課題を整理しておく．本稿では，検索にインデクスを使用することを前提に，有効な特徴ベクトルの考案に重点を置いた．今後は，さらに新しい特徴量を考案するとともに，それらをどのような組合せで使用するか，またそれぞれの特徴をどのような重みをつけて利用するべきかといった検討が必要である．また，本稿ではハミングデータの中央部分を検索キーに使用したが，ハミングのどの部分を，あるいはハミング全体をどのように検索キーとして利用すべきなのかも検討が必要である．さらに，本稿ではハミングに含まれるエラーの中でも，継続的な音高の上昇/下降や継続的なテンポのミスという問題は解決していない．今後は，これらに対しても有効な手法を考案しなければならない．もちろん，ウィンドウ長（=最低ハミング長）もさらに短くてすむように工夫しなければならない．

人は，自分の知っている曲ならその曲のハミングが多少音程やテンポがずれていてもその曲名をあてることができる．我々の検索システムは，ハミングに対して少々の音程のずれやテンポのずれは吸収することができるが，人が自分の知っている曲に対して，そのハミングから柔軟に曲名を識別するようなレベルには達していない．我々の究極の目標は，この検索システムを,「数十万曲の曲をよく知っている人」のようにすることである．

参考文献

1) Ma, W.Y., Manjunath, B.S., Luo, Y., Deng, Y. and Sun, X.: *NETRA: A Content-Based Image Retrieval System*. http://maya.ece.ucsb.edu/Netra/
2) Smith, J.R. and Li, C.S.: Image classification and querying using composite region templates, *Journal of Computer Vision and Image Understanding* (1999).
3) Ghias, A., Logan, J. and Chamberlin, D.: Query By Humming, *Proc. ACM Multimedia 95*, pp.231–236 (1995).
4) Curtis, K., Taniguchi, N., Nakagawa, J. and Yamamuro, M.: A comprehensive image similarity retrieval system that utilizes multiple feature vectors in high dimensional space, *Proc. International Conference on Information, Communication and Signal Processing*, pp.180–184 (1997).
5) White, D. A. and Jain, R.: Similarity Indexing: Algorithms and Performance, *Proc. SPIE IV*, Vol.2670, pp.62–75 (1994).
6) McNab, R.: Interactive Applications of Music Transcription, Master's thesis, Computer Science at the University of Waikato (1996).
7) Rolland, P.Y., Raskinis, G. and Ganascia, J.G.: Musical Content-Based Retrieval: an Overview of the Melodiscov Approach and System, *Proc. ACM Multimedia 99*, pp.81–84 (1999).
8) 山本順人：音楽データベース，情報処理，Vol.29, No.6, pp.599–607 (1988).
9) 蔭山哲也，高島洋典：ハミング歌唱を手掛かりとするメロディ検索，電子情報通信学会論文誌，Vol.J77-D-II, No.8, pp.1543–1551 (1994).
10) 園田智也，後藤真孝，村岡洋一：WWW 上での歌声による曲検索システム，信学技報，pp.25–32,

電子情報通信学会 (1998).
11) 橋口博樹，西村拓一，岡　隆一，赤坂貴志：鼻歌の旋律と歌詞をクエリーとする楽曲信号のスポッティング検索，信学技報 PRMU200-107〜118, Vol.100, No.443, pp.79–86 (2000).
12) Kosugi, N., Nishihara, Y., Sakata, T., Yamamuro, M. and Kushima, K.: A Practical Query-By-Humming System for a Large Music Database, *Proc. 8th ACM International Conference on Multimedia*, pp.333–342 (2000).
13) Blackburn, S. and DeRoure, D.: A Tool for Content Based Navigation of Music, *Proc. ACM Multimedia 98* (1998).
14) Wu, S. and Manber, U.: Fast Text Searching Allowing Errors, *Comm. ACM*, Vol.35, No.10, pp.83–91 (1996).
15) Sonoda, T. and Muraoka, Y.: A WWW-based Melody-Retrieval System — An Indexing Method for A Large Melody Database, *Proc. ICMC 2000*, pp.170–173 (2000).
16) 古井貞熙：ディジタル音声処理，東海大学出版会 (1992).
17) Nishihara, Y., Kosugi, N., Kon'ya, S. and Yamamuro, M.: Humming Query System Using Normalized Time Scale, *Proc. CODAS'99* (1999).
18) Wildcat Canyon Software: AUTOSCORE. http://www.wildcat.com/Site/Homepage.htm
19) 鹿野清宏，中村　哲，伊勢史郎：音声・音情報のディジタル信号処理，昭晃堂 (1997).
20) 小杉尚子，西原祐一，紺谷精一，山室雅司，串間和彦：ハミングを用いた音楽検索システム，情報処理学会研究報告 99-DBS-119, pp.49–54, 情報処理学会 (1999).
21) Kosugi, N., Nishihara, Y., Kon'ya, S., Yamamuro, M. and Kushima, K.: Let's Search for Songs by Humming!, *Proc. ACM Multimedia 99* (*Part 2*), p.194 (1999).

(平成 13 年 5 月 30 日受付)
(平成 13 年 12 月 18 日採録)

小杉　尚子

1993 年慶應義塾大学理工学部電気工学科卒業．1995 年同大学院理工学研究科計算機科学専攻修士課程修了．同年，日本電信電話（株）入社．以来，リアルタイム OS の研究を経て，現在は音楽検索システムの研究開発に従事．

小島　　明

1988 年東京大学工学部計数工学科卒業．1990 年同大学院工学系研究科計数工学専攻修士課程修了．同年日本電信電話（株）入社．以来，映像データベースシステム・電子図書館システム等の研究開発に従事．

片岡　良治（正会員）

1985 年千葉大学工学部電子工学科卒業．1987 年同大学院電子工学専攻修士課程修了．同年，日本電信電話（株）入社．以来，トランザクションの並行処理制御方式の研究，マルチメディア情報検索システムの研究開発に従事．電子情報通信学会会員．

串間　和彦（正会員）

1980 年京都大学工学部電子工学科卒業．2001 年博士（情報学）京都大学．1980 年日本電信電話公社（現 NTT）入社．以来，知識処理用プログラミング環境の研究，大規模クライアントサーバシステムの実用化等を経て，現在はマルチメディアデータベースの研究開発に従事．