Vol.2011-MUS-90 No.4
2011/5/13

オーディオ指紋検索に適した高速なハミング空間検索

北　　研　二[†1]　　肖　　清　梅[†1]

楽曲検索では，ノイズ等による楽曲の劣化や，検索質問曲の演奏開始位置が不明であることから，一般に膨大な検索空間を探索する必要がある．多くの楽曲検索システムでは，楽曲の音響的・知覚的な特性に基づくオーディオ指紋 (audio fingerprint) を特徴量として用いているが，本稿では，オーディオ指紋に適した高速検索手法を提案する．また，8,740 曲の楽曲データベースを用いた評価実験を行い，提案手法の有効性を示す．

Fast Hamming Space Search for Audio Fingerprinting Systems

Kenji Kita[†1] and Qingmei Xiao[†1]

In music information retrieval, a huge search space has to be explored because a query audio clip can start at any position of any music in the database, and also a query is often corrupted by highly significant noise and distortion. Audio fingerprints have attracted much attention recently in music information retrieval, because they provide compact representation of the perceptually relevant parts of audio signals. In this paper, we propose an extremingly fast method for exploring a huge hamming space that is suitable for audio fingerprinting systems. The effectiveness of the proposed method has been confirmed by evaluation experiments using a database of 8,740 songs.

†1 徳島大学
The University of Tokushima

1. はじめに

指紋 (fingerprint) が個人の特定に用いられるように，オーディオ指紋 (audio fingerprint) は楽曲の特定や正真性を確認するための特徴量である．オーディオ指紋の主な用途は楽曲の認識・識別であり，楽曲情報データベースと組み合わせることにより，楽曲の一部から曲名，アーティスト名，アルバム名などを自動的に取得するために用いられている．Gracenote[1] や Midomi[2] 等の商用検索サービスが有名であり，これらのサービスでは，数秒間程度の楽曲をパソコンや携帯電話のマイクに拾わせることにより，ユーザに曲名やアーティスト名を通知したり，該当する楽曲をダウンロードすることなどを可能としている．また最近は，音楽の著作権を保護するための手法としても，オーディオ指紋が注目されており，ネットワーク上で流通している著作権を侵害している楽曲の検出等にも用いられている．

一般に楽曲検索においては，検索質問（クエリ）として与えられた曲片が，楽曲中のどの位置から始まるかが不明であるため，検索処理では，楽曲データベースに含まれるすべての曲に対して，任意の開始位置からの照合を考慮する必要がある．楽曲検索の検索空間は非常に膨大なものとなるため，効率的な検索手法が必要となる．オーディオ指紋に基づく効率的な検索手法として，ハッシュ表を用いる手法[3),5)]，木構造を用いた探索手法[4)] などが提案されている．

また，楽曲検索では，検索質問曲にノイズが混入していたり，音質が劣化している場合も多々あるため，不完全な一致に基づく検索を行う必要があり，検索質問曲と類似したものも検索の対象とする必要が生じる．この意味で，楽曲検索は一種の類似検索である．近年，高次元の大規模データに対する類似検索の手法として，Locality-Sensitive Hashing (LSH) が注目を集めており，画像検索を始めとするさまざまな情報検索分野に適用されてきている[6)–8)]．

本稿では，LSH からヒントを得て，オーディオ指紋に適したハミング空間の高速検索手法を提案する．LSH に基づくハミング空間の検索手法は，従来，いくつか提案されているが，本稿で提案する手法は従来手法に比べ，記憶容量がきわめて少ないという特徴を持っている．また，検索手法の有効性を 8,740 曲を対象とした楽曲検索実験において評価する．以下，2 節では，オーディオ指紋による楽曲検索の概要を説明する．3 節では，オーディオ指紋に対する高速な検索手法を提案し，4 節において提案手法の評価について述べる．最後に，本稿のまとめを 5 節で述べる．

2. オーディオ指紋による楽曲検索

オーディオ指紋 (audio fingerprint) は，メッセージダイジェスト（一方向ハッシュ関数）の一種であり，楽曲の音響的・知覚的特性に基づき，元の楽曲データを比較的コンパクトなビット列表現に変換するための技術である．認証やデジタル署名等で用いられている MD5 等のメッセージダイジェストでは，元のデータが少しでも異なると，まったく異なったハッシュ値が得られるが，オーディオ指紋の場合には，楽曲にノイズが混入したり，楽曲が劣化した場合にも，類似したハッシュ値が得られるように設計されている．

オーディオ指紋を用いた楽曲検索では，どのような種類のオーディオ指紋を用いるか，また，どのようにしてオーディオ指紋の照合および検索を行うかという要素技術が重要である．以下では，オーディオ指紋の抽出，照合，検索について，概要を述べる．

2.1 オーディオ指紋の抽出

オーディオ指紋抽出アルゴリズムには，さまざまなものが提案されており，フーリエ係数特徴に基づく手法[9]，メルケプストラム係数特徴に基づく手法[10]，スペクトル扁平特徴に基づく手法[11] などが研究されている．以下では，Haitsma および Kalker による周波数帯域間のエネルギー差の特徴を用いたオーディオ指紋抽出アルゴリズムの概要を示す．

Haitsma-Kalker アルゴリズムでは，入力された楽曲データをフレームに分割後，(オーバーラップした）各フレームからサブ指紋 (sub-fingerprint) と呼ばれる 32 ビットの特徴量を抽出する．サブ指紋は，実際には周波数領域において計算する．各フレームを FFT により周波数領域に変換後，重なりのない 33 個の周波数帯域に分割し，帯域間のエネルギー差の符号（プラスあるいはマイナス）からサブ指紋を求める．Haitsma および Kalker では，フレーム長 0.37 秒（フレームの重なり度は 31/32）であり，ハニング・ウィンドウ (Hanning window) により重み付けした後に，周波数領域に変換している．したがって，11.6 ミリ秒ごとに 1 つのサブ指紋が抽出されることになる．

具体的なサブ指紋の計算は，フレーム n の周波数帯域 m におけるエネルギーを $E(n,m)$ とするとき，フレーム n の第 m ビット $F(n,m)$ を以下の式により求める．

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

$$ED(n,m) = E(n,m) - E(n,m+1) - (E(n-1,m) - E(n-1,m+1)) \tag{2}$$

Haitsma および Kalker は，周波数領域におけるエネルギー差の符号から得られる特徴が，楽曲の識別・同定において有効であり，しかも圧縮や伸長等の多くの処理に対して頑健であることを実験的に示している．Haitsma-Kalker アルゴリズムは，オーディオ指紋抽出の各ステップが単純な算術演算で実装でき，生成されるオーディオ指紋がコンパクトであるという特徴を持っている．

2.2 オーディオ指紋の照合

サブ指紋は楽曲中の 1 フレームから得られる 32 ビットの特徴量であり，サブ指紋 1 つだけでは，楽曲を識別・同定するための十分な情報を含んでいない．このため，オーディオ指紋の照合においては，サブ指紋のブロック (fingerprint-block) を用いる．なお，サブ指紋のブロックとは，一定の長さのサブ指紋の時系列データである．Haitsma および Kalker では，ブロック長を 256 としている．

また，2 つのサブ指紋ブロックの同一性を判定するための尺度として，ビットエラー率 (bit error rate) を用いる．いま，2 つの楽曲片 A および B から抽出されたサブ指紋をそれぞれ $F_A(n,m), F_B(n,m)$ とするとき，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，以下のようにして計算することができる．

$$BER(A,B) = \frac{\sum_{n=1}^{N}\sum_{m=1}^{32}[F_A(n,m) \wedge F_B(n,m)]}{32\,N} \tag{3}$$

なお，演算子 $\wedge$ は，ビット演算 XOR(exclusive or) を表している．上式の分子は，2 つのサブ指紋ブロック間のハミング距離 (Hamming distance) を計算しており，サブ指紋ブロックのビット長 $32\,N$ で除算することにより，1 ビット当たりのエラー率を計算している．

2.3 オーディオ指紋の検索

オーディオ指紋に基づく楽曲の検索は，基本的に，以下のように行う．まず，検索対象である楽曲データベース中の各楽曲から複数のサブ指紋ブロックを抽出する．1 つの楽曲データに対し，開始フレーム 1 から得られるサブ指紋ブロック，開始フレーム 2 から得られるサブ指紋ブロック，開始フレーム 3 から得られるサブ指紋ブロックというように，非常に多数のサブ指紋ブロックが得られる．検索質問として楽曲片が与えられると，この楽曲片からも複数のサブ指紋ブロックが得られる．楽曲検索は，検索質問から得られたサブ指紋ブロックとの距離を最小にするような楽曲データベース中のサブ指紋ブロックを見つける問題となる．

オーディオ指紋の検索空間は一般に非常に膨大である．たとえば，Haitsma-Kalker アルゴリズム（上述の分析条件）により，1 曲あたりの平均長を 5 分と仮定し，10,000 曲からオーディオ指紋を抽出すると，総計で 2.5 億のサブ指紋ブロックが得られることになる．検

索質問から得られる複数のサブ指紋ブロックとの照合を考えると，単純な方法 (brute-force search) だと，2.5 億の数倍から数十倍の距離計算を行う必要がある．

上で述べたように，逐次的に距離計算を行う単純な線形探索では，楽曲データベースの規模に比例した計算量が必要となるため，効率的な検索手法が求められる．Haitsma および Kalker[3] は，サブ指紋に対するハッシュ表（ルックアップ・テーブル）を構成することにより効率的に検索する手法を提案している．Miller ら[4] は，データベース中の楽曲に対するサブ指紋を木構造で表現することにより，効率的に検索する手法を提案している．また，Wang[5] は，周波数領域におけるピーク値と 2 つのピーク値間の時間差を組み合わせたハッシュ表を用いた検索手法を提案している．しかし，これらの手法では，検索質問とデータベース中の楽曲とのビットエラー率が大きくなるにつれ，ハッシュ表の大きさが急激に増大するという欠点がある．

3. オーディオ指紋に対する高速検索

本節では，オーディオ指紋に対する高速な検索手法を提案する．基本的な仮定として，オーディオ指紋は 0 と 1 のビット列で表現されており，オーディオ指紋間の距離としてハミング距離 (Hamming distance) を用いるとする．以下，3.1 節で Locality-Sensitive Hashing に基づくハミング空間における検索手法の概要を述べ，3.2 節でオーディオ指紋に適した検索手法を提案する．

3.1 Locality-Sensitive Hashing に基づく検索

Locality-Sensitive Hashing (LSH) は，特定のアルゴリズムというよりも，大規模な高次元データに対する確率的な検索手法の総称であり，ビット・サンプリング (bit sampling) を用いたハミング距離に対する LSH[6]，Min-wise Independent Permutation を用いた Jaccard 距離に対する LSH[12]，Random Projection を用いた cosine 距離に対する LSH[13]，p-安定分布 (p-stable distribution) を用いた L_p 距離に対する LSH[14] など，多数の検索アルゴリズムが提案されている．いずれのアルゴリズムも，高次元のベクトルデータをハッシュ値に変換することを基本としており，変換後のハッシュ値は元データの距離が近い場合には高い確率で同じ値になるという性質を持っている．一般に，高次元データに対するハッシュ関数で，近傍データのみを一致させるような関数を一意に決めることはできず，LSH では，複数のハッシュ関数を用いることで，一定の検索精度を保つようにしている．

ハミング距離に対する LSH は，Indyk および Motwani[15] が PLEB (Point Location in Equal Balls) 問題に帰着させたアルゴリズムを提案しており，その後，Charikar[13] や Ravichandran ら[16] がビット列のランダム置換 (random permutation) を用いた，より洗練されたアルゴリズムを提案している．

ランダム置換を用いた手法の概要を，以下に示す．いま，検索対象として n 個の k 次元のデータ集合 $D = d_1, d_2, \cdots, d_n$ があるとする．ここで，各 d_i は 0 あるいは 1 から成る k ビットの列である．置換 σ とは集合 $\{1, 2, \cdots, k\}$ 上の全単射 (bijection) のことであり，ビット列に対する置換は，ビット列 $b_1, b_2, \cdots, b_k$ をビット列 $b_{\sigma(1)}, b_{\sigma(2)}, \cdots, b_{\sigma(k)}$ に置き換えたものとして定義することができる．集合 $\{1, 2, \cdots, k\}$ 上の置換の総数は $k!$ 個あり，ランダム置換はこれらの置換の中からランダムに 1 つを選択したものである．

さて，データ集合 D 中の各要素に対して，ランダム置換 σ を施したデータ集合 D_σ を作成する．検索質問のビット列 q に対しても，同様に，ランダム置換 σ を施したビット列 q_σ を作成する．データ集合 D_σ の中から q_σ に近いデータを探すために，D_σ をソートしておき，ソート列に対し 2 分探索 (binary search) を適用する．多数のランダム置換に対し，上記のことを実行すれば，検索されるビット列データは，元の検索質問ビット列の近傍にあることが期待される．以上が，Charikar[13] や Ravichandran ら[16] が提案したハミング空間における近傍検索の概要である．なお，この手法の理論的および実験的解析に関しては，Charikar[13],Manku ら[17] を参照せよ．

3.2 オーディオ指紋に対する高速ハミング空間検索

ランダム置換に基づくハミング空間検索の考え方はきわめて単純である．ビット列の一致検索 (exact match) でよければ，元のデータ集合に対する 2 分探索のみで十分である．また，下位ビットのみが一致しない場合の近傍検索も 2 分探索で十分であろう．問題は，上位ビットに不一致がある場合であり，この際には単純な 2 分探索では検索漏れを起こしてしまう．このような事態に対処するために，複数のランダム置換によるビットの入れ替えを考えているのである．一般に LSH では，複数のハッシュ関数を用いるが，ランダム置換に基づくハミング空間検索では，ランダム置換がハッシュ関数に相当していると考えることができる．

ランダム置換に基づく検索の最大の欠点は，膨大なデータ容量を必要とすることである．効率的な検索を行うためには，元の検索対象データ集合にあらかじめランダム置換を施したデータ集合を作成しておく必要がある．また，複数のランダム置換を考える必要があるため，元データ集合の数倍〜数十倍程度の容量が必要となる．

ランダム置換に基づく検索では，元のデータ集合を（複数のランダム置換により）多重化することにより，ビット列データに対する近傍検索を実現しているということができる．も

し仮に，検索質問のみを多重化することができれば，元のデータ集合に対する多重化は不要になり，少ない記憶容量でハミング空間の検索が可能になると考えられる．我々は，検索質問のみの多重化を実現するために，検索質問と類似したビット列データを複数作成するという手法を提案する．

図 1 にランダム置換に基づく検索の模式図を，また**図 2** に検索質問多重化による検索の模式図を示す．ランダム置換に基づく検索では，検索漏れの問題を解消するために，元のデータ集合および検索質問双方に対する複数のランダム置換（図中，$\sigma_1, \sigma_2, \sigma_3$）を考えている．一方，検索質問多重化による検索では，検索質問と類似したビット列データを作成する関数（図中，$\varphi_1, \varphi_2, \varphi_3$）により，検索質問のみを多重化している．

オーディオ指紋による楽曲検索は，一般にサブ指紋の照合に基づき実行されるが，検索質問曲からは，開始フレームが少しずつ異なる複数のサブ指紋が抽出できる．さらに，サブ指紋の時系列において，隣合ったサブ指紋の間には大きな類似性があり，時間が進むにつれ，少しずつ異なったサブ指紋が得られるという特徴がある．このような開始時間の異なる複数のサブ指紋を用いて検索質問の多重化を行うことにより，元のデータ集合に対するランダム置換を考慮せずに検索を行うことが可能となる．

我々の検索アルゴリズムでは，まず最初に，検索質問曲のサブ指紋を用いて楽曲データ中の候補位置を絞り込み，その後，サブ指紋ブロックに対するハミング距離（ビットエラー率）を計算する．ただし，サブ指紋 1 つだけでは，楽曲を識別・同定するための十分な情報を含んでいないので，実際には，サブ指紋の短い時系列を用いる．なお，我々の楽曲検索システムでは，サブ指紋の短系列の長さを 3（96 ビット）としている．

サブ指紋の短系列の検索について説明する（**図 3** 参照）．短系列の長さを m，楽曲データベース中の全曲から得られたサブ指紋の系列を $FP = FP_1, FP_2, \cdots, FP_n$ とするとき，FP 中の長さ m の全系列のソート位置を表す 1 次元配列 $S = S_1, S_2, \cdots, S_n$ を考える．配列 S は，接尾辞配列 (suffix array)[18] と同様に，系列 FP へのインデックスを格納した配列であり，次を満たすようにする．

$$S_j = i \text{ iff } FP_i, FP_{i+1}, \cdots, FP_{i+m-1}\text{がソート順で } j \text{ 番目の短系列} \quad (4)$$

検索処理では，検索質問曲から得られたサブ指紋の短系列に対し，配列 S 上を 2 分探索すればよい．また，2 分探索された位置の前後を調べることで，サブ指紋短系列の近傍検索を行うことができる．

上で述べた配列 S が，楽曲検索用のインデックスとなる．検索インデックスの大きさは，楽曲データベース中のサブ指紋系列の長さに比例した大きさであり，ランダム置換を用いる手法に比べ，はるかに少ない容量である．実際の楽曲検索は，以下のように行われる．

(1) 検索質問曲からサブ指紋の時系列を抽出する．
(2) すべてのサブ指紋短系列に対して，配列 S 上を 2 分探索し，検索候補位置を求める．
(3) 検索候補位置を開始位置とするサブ指紋ブロックのハミング距離（ビットエラー率）

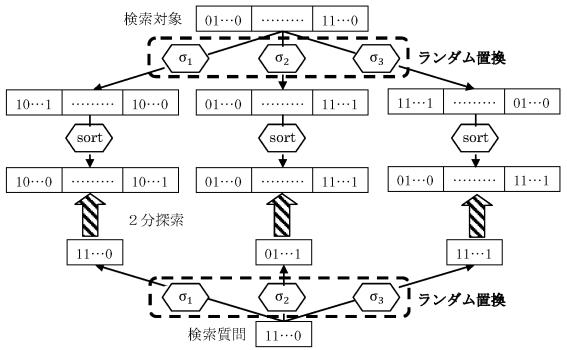

図 1 ランダム置換に基づく検索
Fig. 1 Schematic diagram of search based on random permutation

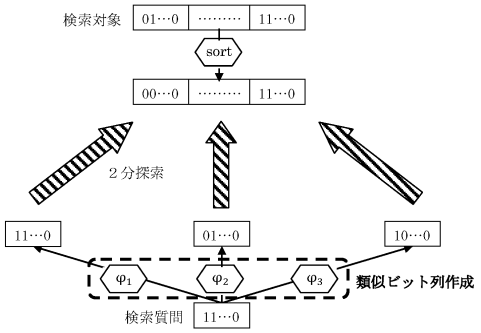

図 2 検索質問多重化による検索
Fig. 2 Schematic diagram of search based on query multiplexing

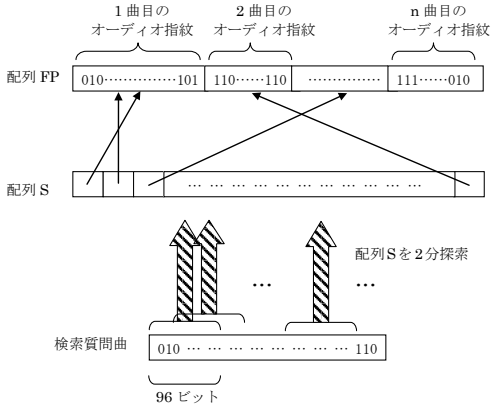

図 3 サブ指紋系列の検索
Fig. 3 Schematic diagram of sub-fingerprint sequence search

を計算する．

(4) ハミング距離に基づき，検索結果をソートし，上位のものを検索結果とする．

4. 評 価 実 験

提案した楽曲検索アルゴリズムの有効性を検証するために，実際の楽曲データを用いた評価実験を行った．評価実験で用いたオーディオ指紋抽出は，基本的には Haitsma-Kalker アルゴリズムと同様の手法を用いたが，音楽分析条件等は異なっている．

4.1 楽曲データ

検索対象となる楽曲は，8,740 曲の MP3 データである．これらの曲は，CD やインターネットなどから個人的に収集した曲であり，曲によって MP3 の圧縮率は異なっている．楽曲のジャンルも，和洋ポップス，クラシック，民族音楽など，多岐に渡っている．

検索質問となる楽曲は，YouTube にアップロードされたものを用いた．PV やライブ映像など，多様な形態の動画から音声部分のみを抽出した．前後に数十秒程度の無音があるもの，拍手，歓声，セリフなどを含んでいるもの，ノイズを含んでいるものなど，低品質な楽曲データも数多く存在する．評価用のデータは全 268 曲であり，これらのデータは，データ収集者による分類がなされ，楽曲の状態に関するコメントが付与されている（**表 1** 参照）．

表 1 評価データの分類および正解率
Table 1 Results on evaluation data

分類		説 明	楽曲数	正解率
原曲	ノイズ無	PV 等の原曲に比較的忠実であると判断される曲．ノイズはないか，あっても軽微．	104	96.2%
	ノイズ有	原曲と判断される曲であるが，明らかにノイズが含まれている曲．	22	100.0%
ライブ		ライブ等で歌われた曲．ほとんどの曲に，トーク，歓声，拍手等が含まれる．ノイズが含まれる曲も多い．	142	83.1%

4.2 音楽分析条件

楽曲データは，4,000Hz にダウン・サンプリングを行った後，フレーム長 1.024 秒，フレームシフト 0.032 秒で分析を行い，ハミング窓 (Hamming window) による重み付け，FFT による周波数領域への変換を行っている．周波数領域では，33 個の周波数帯域に分割して，32 ビットのサブ指紋特徴量を抽出した．また，サブ指紋ブロックの長さは 128 とした．

上記の分析条件は，Haitsma および Kalker[3] に比べると荒いものとなっているが，これらの条件は，いくつかの予備実験を行い決定した．提案アルゴリズムでは，上記程度の分析条件でも十分な検索精度が得られることがわかった．なお，この条件で分析を行った際に得られるサブ指紋の総数は，検索対象 8,740 曲に対して，約 7 千万である．

4.3 実験結果および考察

本稿での提案手法は，オーディオ指紋に基づく楽曲の高速検索アルゴリズムであるので，まず検索速度面について説明する．評価実験を行った計算機は DELL Precision M6500（ノートパソコン），CPU は Intel Core i7 (1.73GHz)（全 8 コア），搭載メモリは 4GB である．曲によって検索時間は異なるが，1 曲あたり概ね 0.4〜0.6 秒程度で検索可能である．

検索質問 1 曲あたりのサブ指紋の系列長は，$6,000 \sim 8,000$ 程度のものが大多数を占める．提案アルゴリズムでは，検索質問曲から得られるすべてのサブ指紋短系列（長さ 3）に対して，2 分探索による検索候補位置の計算を行った後，サブ指紋ブロックに対するビットエラー率の計算を行っている．したがって，サブ指紋ブロック 1 つあたりの検索時間は 0.0001 秒以下で行われていることになり，十分，高速であるということができる．

検索精度であるが，表 1 の右欄に，第 1 位での正解率を示している．「原曲」に対する第 1 位での検索率は 96.8%，「ライブ」に対する第 1 位での正解率は 83.1%である．「原曲」に

対しては，ノイズの有無によって評価データを 2 つに分類しているが，結果的には，ノイズによる検索精度への影響はみられなかった.「原曲」と「ライブ」との間で検索精度の違いがみられるが，この理由は，ライブでは元の曲をアレンジしていたり，元の曲調とは異なっている場合があるためであると考えられる.

5. 結　　論

本稿では，オーディオ指紋に基づく楽曲検索に適した高速検索手法を提案した．提案手法は，高次元データに対する確率的な近傍検索アルゴリズムである Locality-Sensitive Hashing (LSH) にヒントを得ている．LSH では，検索精度を保つために，一般に複数のハッシュ関数を用いており，このため検索用データであるハッシュ表に相当の容量が必要となる．ハミング距離に対する従来の LSH の場合，ランダム置換が一種のハッシュ関数に相当するが，ランダム置換により複数個の検索対象データをあらかじめ用意しておく必要がある．我々の提案手法では，開始時間の異なる複数のサブ指紋の短い系列を用いて検索質問の多重化を行うことにより，検索対象データの多重化を必要としない．従来手法に比べ，検索インデックスの大きさをはるかに少なくすることが可能である．評価実験において，検索速度あるいは検索精度の面から評価を行い，提案手法の有効性を示した.

謝辞　音響処理の面で数々の教えを受けた大同大学の柘植覚氏，名古屋大学の武田一哉氏に感謝する．さらに，本研究に対し，有益なコメントを頂いた電気通信大学の渡邉恵理子氏，株式会社 Photonic System Solutions の水野潤氏，日本女子大学の北林理沙氏，日本女子大学名誉教授の小舘香椎子氏に感謝する.

参 考 文 献

1) Gracenote: available from ⟨http://www.gracenote.com/⟩
2) Midomi: available from ⟨http://www.midomi.co.jp/⟩
3) Jaap Haitsma and Ton Kalker: Highly Robust Audio Fingerprinting System, *Proceedings of the 3rd International Conference on Music Information Retrieval* (ISMIR 2002), pp.107–115 (2002).
4) Matthew L. Miller, Manuel Acevedo Rodriguez, and Ingemar J. Cox.: Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces, *Journal of VLSI Signal Processing*, 41, pp.285–291 (2005).
5) Avery Li-Chun Wang: An Industrial-Strength Audio Search Algorithm, *Proceedings of the 4th International Conference on Music Information Retrieval* (ISMIR 2003), pp.7–13 (2003).
6) Aristides Gionis, Piotr Indyk, and Rajeev Motwani: Similarity Search in High Dimensions via Hashing, *25th International Conference on Very Large Data Bases* (VLDB 1999), (1999).
7) Brian Kulis and Trevor Darrell: Learning to Hash with Binary Reconstructive Embeddings, *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems* (NIPS 2009), pp.1042–1050 (2009).
8) Brian Kulis and Kristen Grauman: Kernelized Locality-Sensitive Hashing for Scalable Image Search, *Proceedings of the 12th IEEE International Conference on Computer Vision* (ICCV 2009), (2009).
9) 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).
10) Beth Logan: Mel Frequency Cepstral Coefficients for Music Modeling, *Proceedings of the International Symposium on Music Information Retrieval* (ISMIR 2000), (2000).
11) Eric Allamanche et al.: AudioID: Towards Content-based Identification of Audio Material, *110th AES Convention*, (2001).
available from ⟨http://www.aes.org/e-lib/browse.cfm?elib=10019⟩
12) Andrei Z. Broder, Moses Charikar, Alan M.Frieze, and Michael Mitzenmacher: Min-wise Independent Permutations, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp.327–336 (1998).
13) Moses S. Charikar: Similarity Estimation Techniques from Rounding Algorithms, Proceedings of the 34th Annual ACM Symposium on Theory of Computing, (2002).
14) Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S.Mirrokni: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions, *Proceedings of the 20th Annual Symposium on Computational Geometry*, (2004).
15) Piotr Indyk and Rajeev Motwani: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, (1998).
16) Deepak Ravichandran, Patrick Pantel, and Eduard Hovy: Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering, *Proceedings of ACL*, (2005).
17) Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma: Detecting Near-Duplicates for Web Crawling, *Proceedings of the 16th international conference on World Wide Web*, pp.141–149, (2007).
18) Udi Manber and Gene Myers: Suffix Arrays: A New Method for On-line String Searches, *SIAM Journal on Computing*, Vol.22, No.5, pp.935–948, (1993).