大規模な音楽指紋データベースの高速検索における クエリの歪みへの頑健性向上に関する調査研究

北陸先端科学技術大学院大学
情報科学研究科

福田 真啓

平成 27 年 9 月

修 士 論 文

大規模な音楽指紋データベースの高速検索における クエリの歪みへの頑健性向上に関する調査研究

1310203　　福田 真啓

主指導教員　井口 寧
審査委員主査　井口 寧
審査委員　田中 清史
　　金子 峰雄

北陸先端科学技術大学院大学

情報科学研究科

平成 27 年 8 月

概 要

近年のインターネットは，音楽の流通を活発にする場を提供している一方で，不正コピーの温床にもなっている。これに対し，エンドユーザが音楽データを通信するだけで自動的にライセンス・課金情報を受信者に提示できるという，手軽で合法的な音楽共有システムが提案されている。しかし高速化を続けるネットワーク機器内で音楽をリアルタイム処理で検索するのは決して容易ではない。データベース (DB) が千万曲規模になるとなおさらである。本研究の目的は，千万曲以上の DB の高速かつ高精度な音楽検索を組込みシステムとして実現することである。

音楽の高速検索において重要な要素技術は音楽指紋である。検索を高速化・省メモリ化できるため，組込みシステムを前提とした音楽の高速検索には必須の技術である。しかし単に音楽指紋を高速検索できるだけでは，手軽で合法的な音楽共有システムは実用化できない。本稿では指紋検索の大規模 DB 化と歪み対策に焦点を当てる。指紋検索の先行研究である Yang の Staged LSH は指紋 DB とハッシュテーブルに大容量のメモリを必要とするため，300 曲までしか実験がなされていない。別の問題として，ハッシュ探索をしているため指紋のビットエラーにも弱く，検索精度と検索速度が悪化する。

そこで本稿ではいかにして高速化と高精度化を両立できるかを調査した。大規模 DB 化しつつ検索速度を落とさない工夫として，複数種類のメモリを用いた階層化 Staged LSH を提案する。併せて，検索速度と検索精度のトレードオフを柔軟に調整可能な隣接 LSH 探索も提案する。これらの手法により，指紋 DB を千万曲規模に拡大しつつ，検索速度を 27.0 倍に向上，検索精度を維持することに成功した。

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

近年，インターネット上の音楽の流通が盛んになっている。Apple iTunes Store のように大規模な音楽配信サービスも現れており，その配信の場としてインターネットが利用されている。一方でインターネットは不正コピーの温床にもなっており，そのため 2012 年には著作権法が改正され違法ダウンロードが刑罰化された。インターネットで音楽を利用できることは魅力的であるが，常に著作権法を意識しなければならないのは窮屈である。

不正コピーの対策には複数の方法があるが，いずれも課題が残されている。電子透かしの埋め込みは，保存形式などを変更すると埋め込んだ情報が失われやすいし，消費者にとっても自分が不正コピーをしたことに気づきやすくなるわけではない。デジタル著作権管理 (DRM) は暗号化技術などにより第三者によるコンテンツの利用を防ぐ方法であるが，専用機器や専用ソフトウェアでしか再生できないため消費者にとって不便である。音楽や動画の投稿サイトの巡回･監視については，個人個人がメールなどで音楽データを送受信するケースに対応できない。

インターネットで音楽を気軽に利用したいが，気づかぬうちに著作権法を犯すことは回避したい。この要求に応えるため，図 1.1 のような，著作権保護された音楽をインターネットで通信すると，インターネットサービスプロバイダ (ISP) のネットワーク機器が自動的にその音楽を検出して，ライセンスや課金情報を受信者に提示するシステムが提案されている。これが実現すると，世界中の消費者はメールや P2P で音楽を送受信するだけで合法的に音楽を共有できる。

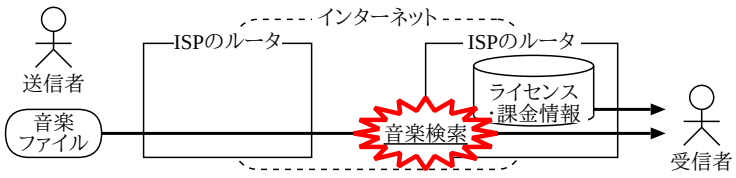


図 1.1: ネットワーク機器内での音楽検索

注意点として，このシステムは暗号化通信に対しては無力である。目的は著作権者側による利用者の監視ではない。利用者が音楽データを手軽かつ合法的に共有することが目的である。このシステムを有効に利用するためには，通信は平文で行う必要がある。

1.2 研究目的

手軽で合法的な音楽検索システムの実用化には，ルータ内で高速かつ高精度に検索できる組込みシステムが必要である。しかしルータの転送速度は 40Gbps，100Gbps，400Gbps と高速化を続けており，音楽データが通過する間にリアルタイム処理で高精度の検索ができている研究は多くない。しかもインターネットには数千万曲以上の音楽が存在する。少なくとも 10GB 以上のデータベース (DB) を格納する記憶装置が必要であるが，低速な HDD や SSD は使用したくないので，実験をすることすら難しいのが現状である。

本研究の目的は，千万曲以上の DB の高速かつ高精度な音楽検索を組込みシステムとして実現することである。音楽検索の高速化と省メモリ化に不可欠な指紋技術を利用し，回路の変更が容易な FPGA で実装・評価実験をする。先行研究で Yang が Staged LSH という音楽指紋検索アルゴリズムを FPGA に実装・実験しているので，これを改良して大規模 DB 化などを目指す。本研究の独創的な点は，千万曲の DB の格納に HDD や SSD ではなく PCI Express(PCIe) 経由の大容量の DRAM を利用している点であり，DDR3 経由と PCIe 経由の 2 種類のメモリにどのようにデータを配置するかを工夫した。精度についても，高速化と高精度化のトレードオフを柔軟に調整できる手法を考案した。

1.3 本論文の構成

本稿の第 1 章では研究の背景と目的を述べた。第 2 章では音楽の高速検索における重要な要素技術である指紋やその歪み率の定義を示した上で，指紋を用いた音楽検索の先行研究を紹介し，問題点を明らかにする。第 3 章では本研究の提案手法および実装について述べ，先行研究の Staged LSH の問題点を解決する。第 4 章では FPGA を用いて提案手法の評価実験を行ったので，その実験環境と実験結果を示した上で考察をする。最後に第 5 章でまとめと今後の課題で締めくくる。付録 A は提案手法の検索速度と検索精度の理論式の導出である。

第2章　音楽指紋とその関連研究

2.1　音楽指紋とは

音楽の高速検索において重要な要素技術は音楽指紋である (単に指紋とも言う)。指紋とは，録音した音楽のメロディやリズムなどの聴覚的な特徴をまとめた小サイズのデータである。指紋は図 2.1 の方式による音楽の高速検索に活用されている。この方式は図 2.2 のような元の音楽データ同士の比較による素朴な検索に比べて次のような利点がある。

1. 小サイズなので比較回数が少ない (高速)
2. 小サイズなので高速・小容量のメモリを有効活用できる (高速・省メモリ)
3. 同じ曲の MP3，AAC などの複数の保存形式を DB に用意する必要がない (省メモリ)

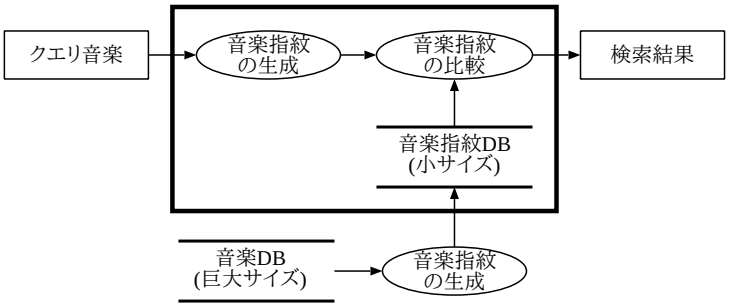


図 2.1: 音楽指紋を用いた高速検索

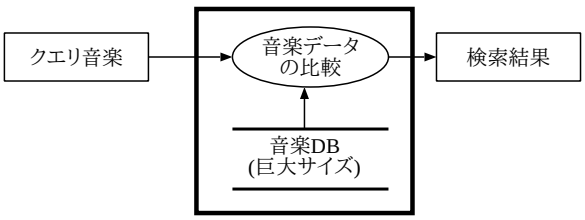


図 2.2: 元の音楽データを用いた高速でない検索

実は前ページの図 2.1 だけでは，1 章で示したネットワーク機器内での音楽指紋検索システムは実現できない。本研究の範囲をより明確にするため，他に必要な処理も合わせた全体像を図 2.3 に示す。本研究の主な焦点は “指紋検索” 部分であるが，他の処理についても本章で簡単に説明する。

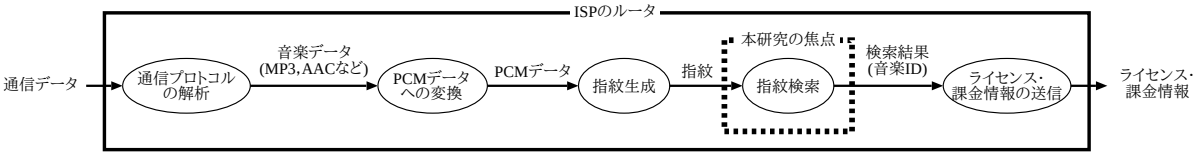


図 2.3: ネットワーク機器内での音楽指紋検索システムに必要な処理

2.2 通信プロトコルの解析

ネットワーク機器内で音楽指紋検索するにあたり，まず最初に “通信プロトコルの解析” をする必要がある。これは ISP のネットワーク機器に入ってくる通信データ (フレームやパケット) から音楽データを抽出する処理である。アプリケーション層の多様なプロトコル (HTTP，SMTP など) や符号化方式 (Base64 など) やそれらのバージョンアップなどにどこまで対応するべきかという点が主な課題であるが，多くは仕様が公開されているのでサポートするプロトコルを限定すれば開発自体は容易である。

2.3 PCM データへの変換

音楽データを抽出したら，次は “PCM データへの変換” を行う。これは MP3 や AAC で圧縮されている音楽データを，非圧縮の PCM(パルス符号変調) データに変換する処理である。この変換処理については，さらなる高速化の研究も考えられるが，既に一定の高速化が達成されている。例えば lame 3.99.5 というフリーソフトを Intel®Xeon®プロセッサで実行してみたところ，5.19MB の MP3 ファイルを 0.837 秒でデコードした。これは変換速度が $8 \times 5.19\text{MB} \div 0.837 = 49.6\text{Mbps}$ であることを意味する。なお本研究では後段の処理工程である “指紋生成” に HiFP2.0 を利用するため，“PCM データへの変換” が出力する PCM データはサンプリング周波数 44.1kHz，量子化ビット数 16 とする。

本処理工程では歪みの問題が発生することがある。歪みの問題とは図 2.4 のように，元々が同じ音楽データであるにも関わらず，保存形式・圧縮率の変更による非可逆圧縮によって聴覚情報が歪むことである。聴覚情報が歪むと後で生成する指紋にも多少のビットエラーを生じさせてしまう。これは現時点のどの指紋生成アルゴリズムでも回避できない問題なので，“指紋検索” の処理工程においては完全一致の探索ではなく類似度を利用した最近傍探索が必要になる。なお本研究では主に音楽 CD からコピーした音楽の特定を目指しているため，録音時の雑音や録音開始のタイミングのずれなどによる歪みは考えない。

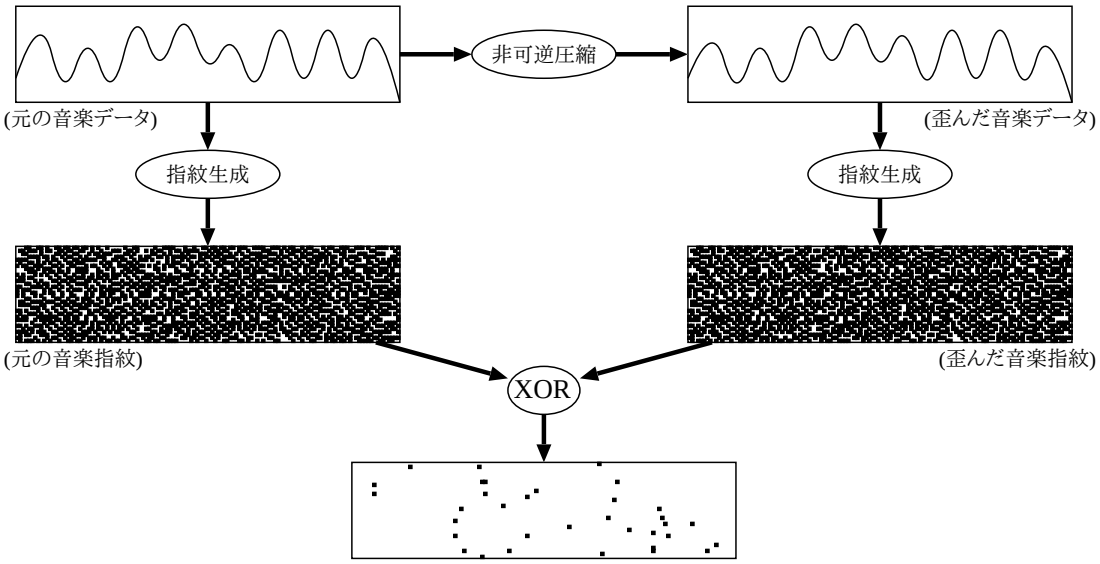


図 2.4: 音楽指紋の歪み

2.4 指紋生成

音楽データの PCM データを得たら，次は “指紋生成” をする。指紋生成アルゴリズムは複数提案されているが，音楽の検索に向いているのは，高速に小サイズの指紋を生成でき，かつ歪みの問題が発生しても同じ音楽か異なる音楽かを確実に見分けられるものである。

Haitsma らは論文 [2] でフーリエ変換を基本とした指紋を生成する方法を提案した。論文には再生時間 11.6 ミリ秒ごとに 32 ビットのサブ指紋を生成し，一万曲では 2.5 億のサブ指紋になると記載されている。すなわち一万曲ですら 1GB の指紋 DB とハッシュテーブルが必要であり，そのまま千万曲に大規模化すると 1TB 以上になる。また，DFT や FFT などのフーリエ変換の計算は，前節の HiFP2.0 に比べハードウェア処理が低速である。

Schreiber らは論文 [3] で Haitsma の方法を発展させ，指紋の中から歪みにくい部分だけを DB に格納する戦略を取ったが，それでも千万曲で 22GB という見積もりである。

荒木らは文献 [1] でウェーブレット変換に基づく HiFP2.0 を提案した。Haitsma らの方法に比べてハードウェア処理に向いている上に，指紋は 1 曲あたり 4096 ビット (千万曲で 5.12GB) で，保存形式· 圧縮率の変更によるビットエラーも (論文の実験の範囲内では) 最大でも 15%未満と小さいことが確認されている。本研究ではこの HiFP2.0 を利用する。

HiFP2.0 の詳細なアルゴリズムを図 2.5 と図 2.6 に示す。図 2.5 は Harr ウェーブレット変換を用いたマルチレベルサブバンド分解 (MHWT) のアルゴリズムであり，図 2.6 は MHWT を用いた HiFP 2.0 の全体の指紋生成アルゴリズムである。HiFP2.0 の処理は大まかに MHWT と特徴抽出に分けられており，図 2.6 の 4 行目が MHWT，8〜12 行目が特徴抽出である。

```
 1:  MHWT(wav[] ← 入力信号,
 2:                n ← 入力信号のサンプル数,
 3:                m ← 出力のサンプル数) {
 4:     for (; n > m; n /= 2) do
 5:        for (i = 0; i < n/2; i++) do
 6:           Hi[i] ← (wav[2 × i] − wav[2 × i + 1])/2;
 7:           Lo[i] ← (wav[2 × i] + wav[2 × i + 1])/2;
 8:        end for
 9:        wav[] ← Lo[];
10:     end for
11:     return (Hi, Lo);
12:  }
```

図 2.5: Haar ウェーブレット 変換を 用いたマルチレ ベルサブバンド 分解アルゴリ ズム

```
 1:  HiFP2.0(wav[] ← PCM データ ) {
 2:     n ← PCM データのサンプル数;
 3:     m ← 出力のサンプル数;
 4:     Hi[], Lo[] ← MHWT(wav[], n, m);
 5:     j ← 0
 6:     for (i = 0; i < m − 4; i += 4) do
 7:        tmp ← Lo[i] − Lo[i + 4]
 8:        if (tmp > 0) then
 9:           FPID[j] ← 1;
10:        else
11:           FPID[j] ← 0;
12:        end if
13:        j++
14:     end for
15:     FPID[m/4 − 1] ← 0;
16:     return FPID;
17:  }
```

図 2.6: 音楽指紋の生成アルゴリ ズム (HiFP 2.0)

HiFP2.0 のアルゴリズムを図解したものも図 2.7 に示す。要するに HiFP2.0 は，音楽データの先頭から約 2.97 秒分を取り，8 サンプルずつ算術平均を取って Lo 配列を作り，Lo 配列を 3 つ飛ばしに大小比較をすることで 4096 ビットの FPID(指紋) を生成する。なお HiFP2.0 ではサンプリング周波数が 44.1kHz であることを前提としており，2.97 秒は 131,072 サンプルに相当する。整数の加算，シフト 演算，比較処理だけで済むのでハードウェア処理に適している。

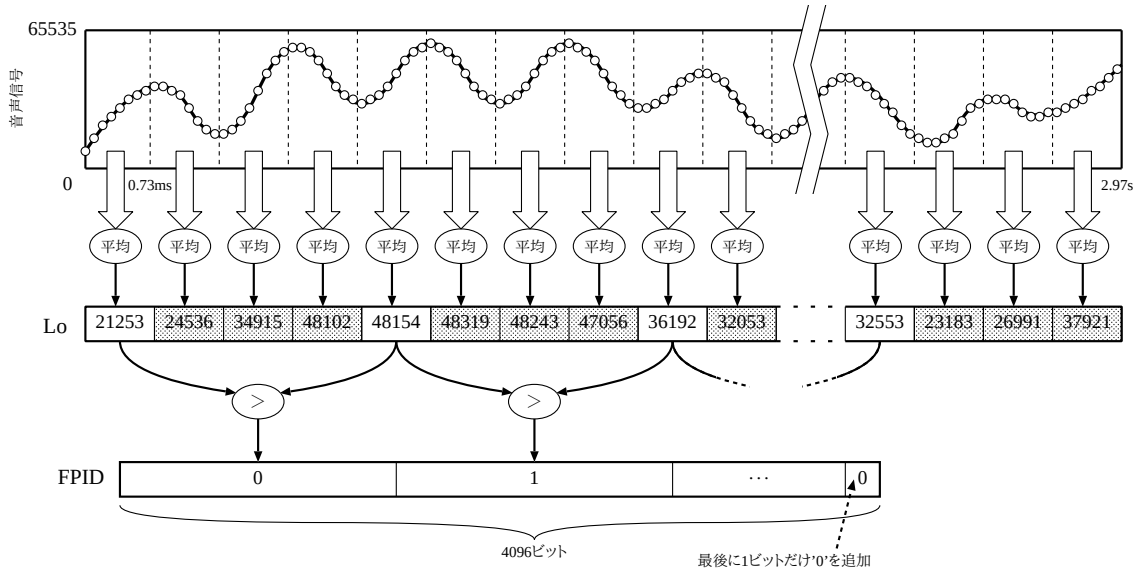


図 2.7: HiFP 2.0 の図解

音楽指紋の歪みは通常，それほど大きくはならない。図 2.8 に，現実に存在する 300 曲の HiFP2.0 の指紋の歪みのヒストグラムを示す。最小 1，平均 80，最大 453 であった (最大歪み率 11.1%)。歪みは lame 3.99.5というフリーソフトを用いて付加した。その時のコマンドは以下の通りである。“-b 128” はビットレートが 128kbps であることを意味する。

```
lame -b 128 --resample 44.1 オリジナルの WAV ファイル MP3 ファイル
lame --decode MP3 ファイル 歪んだ WAV ファイル
```

同様にビットレートを 192kbps にすると，最小 1，平均 34，最大 333 であり，最大歪み率 8.1%であった。ビットレートが 256kbps だと，最小 0，平均 18.3，最大 164 であり，最大歪み率 4.0%であった。

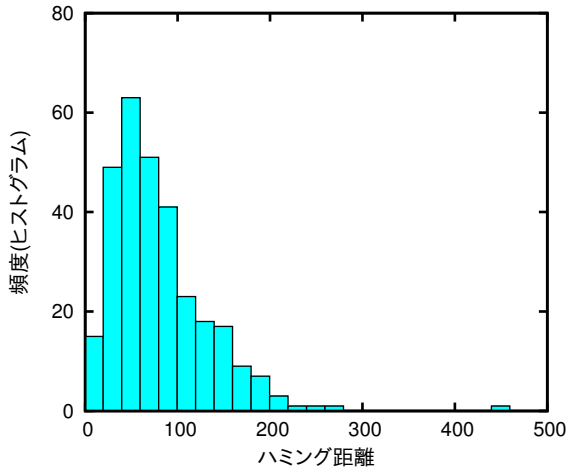


図 2.8: HiFP2.0 の歪みのヒストグラム

2.5 指紋検索

指紋を生成したら “指紋検索” をする。これが本稿の主要テーマである。指紋がどの音楽由来なのかを特定し，検索結果として音楽 ID を返す。ここで音楽 ID とは，指紋 DB に登録済みの音楽または指紋を一意に識別できる番号である。

2.5.1 Yang の研究成果の概要

指紋検索に関しては Yang の研究が組込みシステム向けである。というのも指紋の生成には HiFP 2.0 を用いており，かつ音楽指紋検索システムを FPGA に実装しているからである。Yang は文献 [4] にて，ハミング距離を計算するハードウェアアクセラレータを開発し，局所性鋭敏型ハッシュ (Locality Sensitive Hashing; LSH) を用いたハッシュ探索を改良した。

2.5.2 ハミング距離のハードウェアアクセラレータ

Yang による成果の 1 つはハミング距離を計算するアクセラレータである。これは図 2.9 のようなハードウェアであり，指紋を 32 ビットずつ入力すると，1〜6 段目で 32 ビットのハミング距離を計算し，7 段目でそれまで計算したハミング距離を累積する。パイプライン化しているので，計算時間は表 2.1 のように短い。

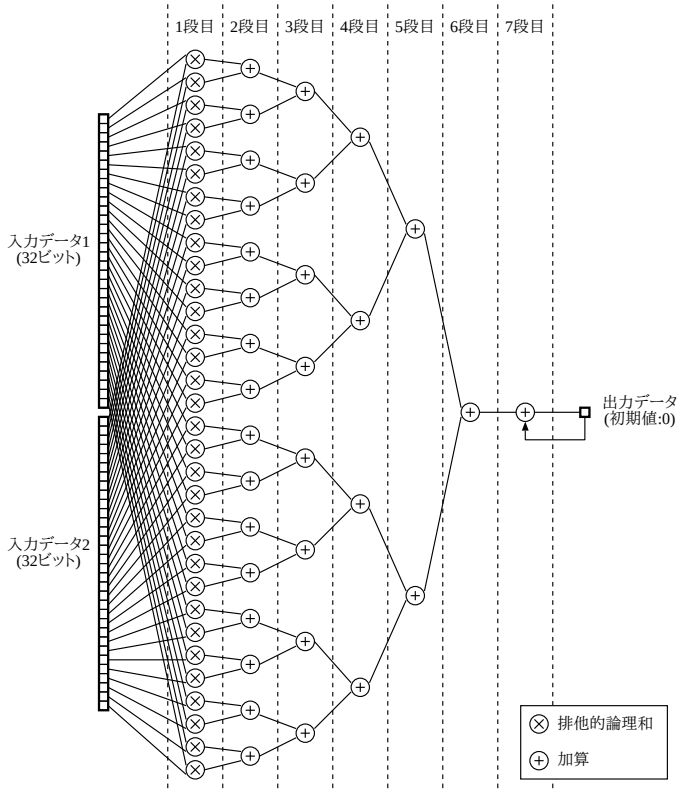


図 2.9: ハミング距離を 計算する ハード ウェア

表 2.1: ハミング距離の計算時間

入力ビット 数	所要クロック数
32	7
64	8
96	9
⋮	⋮
4196	134

2.5.3 Staged LSHとその検索アルゴリズム

Yangによるもう1つの成果はStaged LSHである。これはLSHを用いたハッシュ探索の改良版である。4096ビットの指紋のハミング距離をいきなり計算せず，まずはもっと少ないビット数のハミング距離を計算してスクリーニング処理する方法であり，比較処理の回数を減らすことにより検索を高速化する。

検索全体のアルゴリズムは図2.10の通りである。6行目の“スクリーニング”，8行目の“精査”については，Yangの文献[4]ではそれぞれ“coarse search”，“exact search”と表記されている。以降では，このアルゴリズムについて詳しく説明する。

```
 1: for フレーム t ∈ クエリ指紋 Q do
 2:     ハッシュ値 ← ハッシュ関数で t を処理
 3:     for エントリ i ∈ ハッシュテーブル
 4:         if ハッシュ値 == i.ハッシュ値 then
 5:             for j ∈ i.フレームリスト do
 6:                 if スクリーニング (t, j) ≤ 閾値 1 then
 7:                     P ← j の元となる指紋
 8:                     if 精査 (P, Q) ≤ 閾値 2 かつ P < BestMatch then
 9:                         BestMatch ← P
10:                     end if
11:                 end if
12:             end for
13:             return BestMatch
14:         end if
15:     end for
16: end for
17: return “一致なし”
```

図 2.10: Staged LSHのアルゴリズム

アルゴリズムの内容に入る前に“フレーム”について説明する。Yangは図2.11のように4096ビットの指紋を32ビットずつ128個の“サブ指紋”として区切り，3つの連続したサブ指紋をフレームと呼んでいる。1つの指紋には126フレームある。

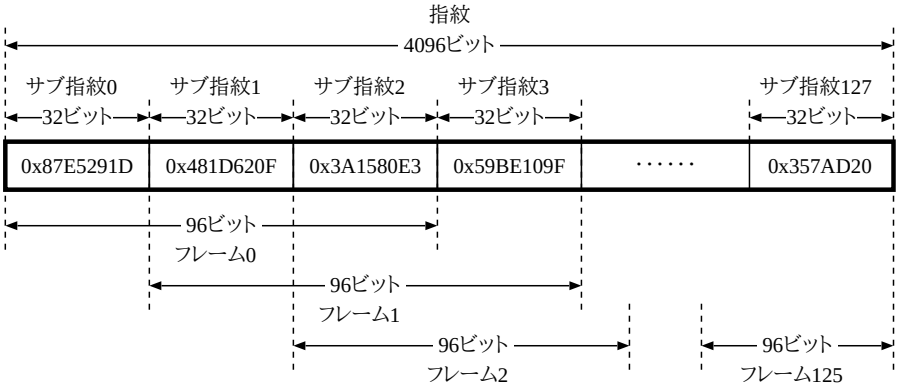


図 2.11: 指紋とサブ指紋とフレーム

Staged LSH の検索アルゴリズムには大きくハッシュ探索，スクリーニング，精査の 3 つの段階がある。図 2.12 のようにこの 3 つの段階を通して徐々に指紋 DB の探索範囲を狭めていく。

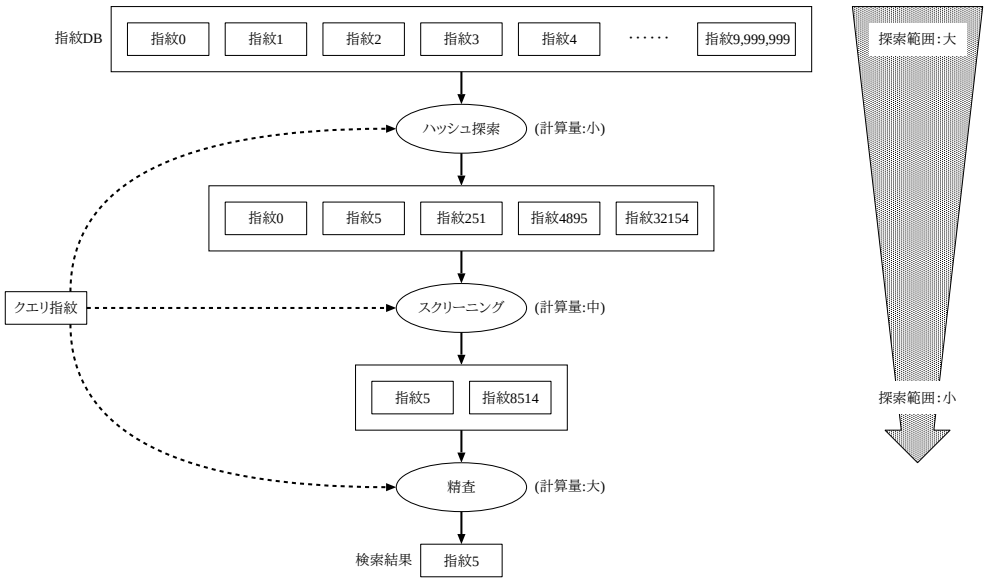


図 2.12: Staged LSH で徐々に指紋 DB の探索範囲を狭めていく様子

ハッシュ探索

Staged LSH の最初の段階はハッシュ探索である。ハッシュ探索は他の多くの音楽指紋検索の研究でも採用されており，線形探索や二分探索などと比べて次の特徴を持つ。

- 長所: DB のサイズによらず計算量が一定である。
- 短所: ハッシュテーブルの格納にメモリ容量が必要である。
  (図 2.14 のデータ形式では千万曲で約 5GB)
- 短所: ハッシュ値の衝突発生時の対応が必要である。

Staged LSH で用いるハッシュ関数は，図 2.13 のように 96 ビットのフレームを入力すると，13 ビットのハッシュ値を出力する局所性鋭敏型ハッシュ (Locality Sensitive Hashing; LSH) である。LSH はビット位置の交換だけであるから，ハードウェア処理ではわずか 1 クロックでハッシュ値を出力できる点が強みである。ハッシュ探索は，このハッシュ値が同じになる指紋だけを指紋 DB から選び取る処理であり，例えば図 2.14 のようなハッシュテーブルを用いると処理が即座に完了する。ハッシュ値の衝突発生時は，同じハッシュ値になるフレームをリストにまとめておき，検索時にそのリスト内のフレームを全て次段階のスクリーニングに渡す。

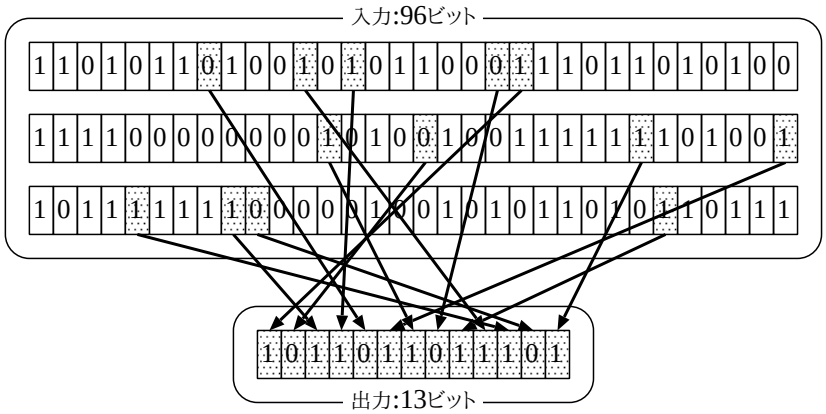


図 2.13: 局所性鋭敏型ハッシュの生成方法

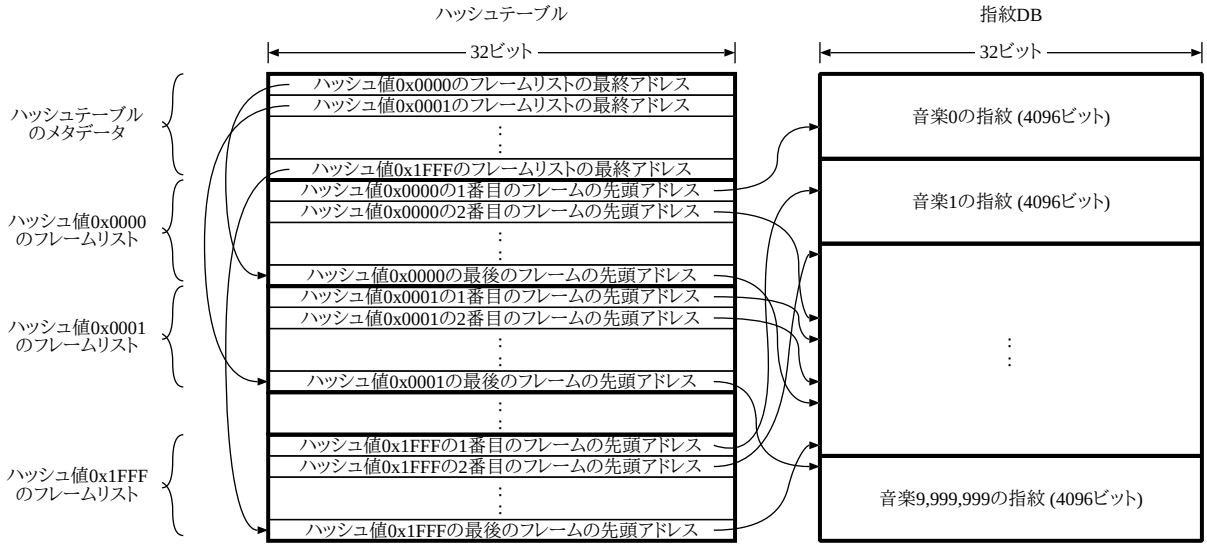


図 2.14: ハッシュテーブルと指紋 DB のデータ構造

スクリーニング (coarse search)

ハッシュ探索の次はスクリーニングである。スクリーニングも精査もハミング距離に基づいて正解の指紋を探し当てるが，ハッシュ探索完了時点ではまだ多数の候補が残ることがある。そこで，いきなり 4096 ビット同士のハミング距離を計算せず，まずは 96 ビットのフレームだけでハミング距離を計算し，一定の閾値以上の指紋をふるい落とす。これが Staged LSH の最大の特徴のスクリーニングである。

精査 (exact search)

精査ではスクリーニングでふるい落とされなかった指紋のみ，4096ビットのハミング距離を計算し，閾値以下でかつ最小の指紋を検索結果として返す。検索結果を返せなかった場合は，クエリ指紋の次のフレームについてハッシュ探索からやり直す。さらにクエリの最終フレームでも検索結果を返せなかった場合は“一致なし”を返す。以上がStaged LSHのアルゴリズムである。

2.5.4 Staged LSHの問題点

Staged LSHはハッシュ探索とスクリーニングのおかげで高速な検索が可能であるが，この方法でもハッシュテーブルのサイズが大きい点は他の先行研究と同様である。ハッシュテーブルの構造を図2.14に示したが，これには全フレームのアドレスが格納されている。そのせいでハッシュテーブルは指紋DBと同程度のサイズになる。例えば千万曲だと，ハッシュ値のビット数を $L_0 = 13$ として

$$\text{指紋DBのサイズ} = 10,000,000 \times 4096 \div 8 = 5.12\text{GB} \quad (2.1)$$

$$\text{ハッシュテーブルのサイズ} = 2^{L_0} \times 4 + 10,000,000 \times 126 \times 4 = 5.04\text{GB} \quad (2.2)$$

となる。Yangの評価実験は300曲までにとどまっているが，これらのデータを格納するメモリを確保するのが難しかったからだと思われる。いずれにせよサポート対象の曲数を将来的に例えば一億曲まで拡大しようとすると，指紋DBとハッシュテーブルで100GB以上必要になる。これは組込みシステムとしてもDRAMだけで対応するのはふさわしくない。低速ではあるがSolid State Drive(SSD)やStorage Class Memory(SCM)の使用を検討するべきである。

また，ハッシュ探索はハッシュ値に誤りがあると探索が失敗するという問題がある。クエリの先頭フレームでハッシュ探索に失敗しても，もし2フレーム目や3フレーム目でハッシュ探索に成功し，スクリーニング，精査も成功すれば，全体としては検索成功となる。しかしそれだけ検索時間が長引くし，実際にYangの論文[4]でも指紋に歪みがあると検索時間が長くなるという実験結果が出ている。

2.6 ライセンス・課金情報の送信

指紋の検索結果(音楽ID)を得た後は，“ライセンス・課金情報の送信”をする。これは音楽IDに対応する音楽のライセンス・課金情報を，音楽のダウンロード先へ送信する処理である。ライセンス・課金情報を送信するための通信のプロトコルは，通信の信頼性やセキュリティなども考慮の上で決定または作成する必要がある。または本処理を“関連パケットを遮断”や“管理会社に報告”に変更・拡張することも考えられる。しかし大部分はプロトコルや運用方法のポリシーの問題であり技術的に困難な問題ではない。

2.7 まとめ

本章では音楽指紋とその関連研究について述べた。ネットワーク機器内の音楽検索システムを実現するには大きく分けて5つの処理を考える必要がある。その中でも指紋生成のHiFP 2.0と指紋検索のStaged LSHについては詳しく説明した。本研究の対象は5つの処理工程の中でも指紋検索の部分であり，その重要な課題の1つが大規模DB化，もう1つの課題が高速化と高精度化の両立である。

第3章　階層化Staged LSHによる大規模音楽指紋検索システムの実装

3.1　組込みシステムとしての音楽指紋検索

本研究ではFPGAで音楽を検索する。というのも，本研究で実現したいアプリケーションは，図3.1のようなネットワーク機器内での音楽検索を必要とする。すなわち組込みシステムであり，以下のような要求が想定されるためである。

- 小型化･軽量化
- 計算リソース･金銭的コストの削減
- 低消費電力化

ネットワーク機器内に組み込めるのは，スーパーコンピュータや高性能のサーバ機器ではなく，マイコン，ASIC，FPGAなどである。その中でも本研究でFPGAを選んだのは，上述の要求を満たしつつ高性能化を狙え，しかも繰り返し実機で検証できるためである。

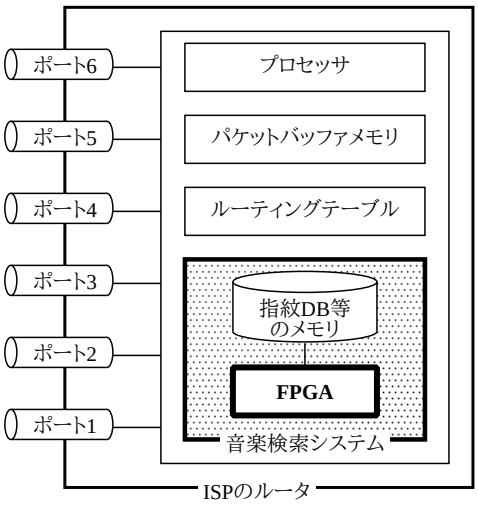


図 3.1: ネットワーク機器内での音楽指紋検索

FPGAとは図3.2のような構造を持つプログラマブルデバイスである。内蔵SRAMに格納されている回路情報をもとにLUTやSMの結線をして動作する。集積回路ではあるが，エンドユーザでも容易に回路を変更して実機で動作させることができる。手順は，まずデジタル回路をVerilog HDLやVHDLなどのハードウェア記述言語で設計し，Xilinx社やAltera社などのFPGAベンダが提供しているツールで“論理合成”，“配置&配線”，“ビットストリーム生成”を行い，その出力ファイルをUSBなどでJTAGを経由してSRAMへ転送すれば良い。

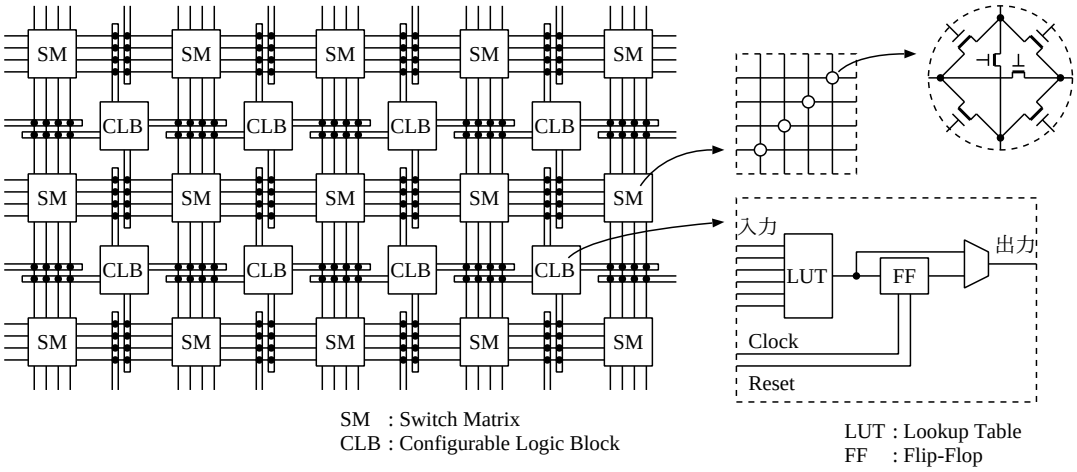


図 3.2: FPGAの基本的な構造の例

検索時間の目標についても検討する。望ましいのは，ネットワーク機器内でリアルタイム処理できることである。これは，もし音楽の通信が同時に多数発生しても，後続の音楽の検索を遅延させないためである。最近のネットワーク機器(有線)は10Mbps〜100Gbpsの通信速度をサポートしており，400Gbpsも仕様策定が進められている。音楽データのサイズは，保存形式･圧縮率にもよるが典型的には1分あたり1〜2MBである。そこで，例えば1MBの音楽データが1Gbpsのネットワーク機器を通過するまでの時間を計算すると，

$$\text{転送時間} = \frac{8 \times (1 \times 10^6)}{1 \times 10^9} \text{秒} = 8 \text{ミリ秒} \tag{3.1}$$

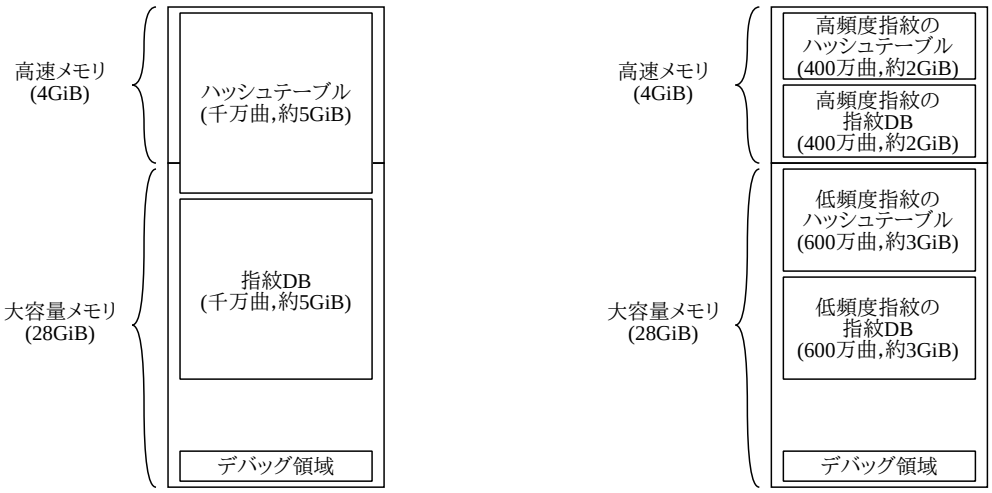
である。すなわちこの数値例では1曲の検索を8ミリ秒以内に抑える必要がある。本研究で使うFPGAボードでは200MHzあるいは250MHzのクロックが利用可能であるから，8ミリ秒はそれぞれ1.6×10^6クロック，2×10^6クロックに相当する。

3.2 階層化 Staged LSH による大規模指紋 DB の検索の高速化

Staged LSH の課題の 1 つは，指紋 DB とハッシュテーブルのサイズが大きいことであった。しかし，だからと言って HDD や SSD を使うとデータ転送速度に限界があるし，今後別種類の二次記憶装置が登場する可能性もある。そこで本研究では高速・小容量の DRAM に加えて，PCIe でアクセス可能な低速・大容量なメモリを別途用意することにした。

通常，コンピュータシステムの記憶装置 (例:一次キャッシュ，二次キャッシュ，DRAM，SSD，HDD など) は，価格が同じであれば，高速なものほど記憶容量が小さい。例えば DRAM の高速性 (例:10GB/s) と HDD の大容量性 (例:1TB) を兼ね備えた記憶装置を，特に組込み用途向けに用意するのは現時点では難しい。このような場合は，頻繁にアクセスするデータを高速メモリに，そうでないデータを大容量メモリにそれぞれ格納し，平均的な転送速度の向上を狙うのが一般的である。

Yang の研究は FPGA のブロック RAM と外部の DRAM しか想定していないので，これ以上メモリを追加すると Yang の実装をそのまま実行できなくなる。素朴な拡張方法は，図 3.3(a) のようにハッシュテーブルの先頭から一部を高速メモリに，ハッシュテーブルの残りと指紋 DB を大容量メモリに格納することである。



(a) Yang の手法の単純拡張　　(b) 階層化 Staged LSH

図 3.3: 指紋 DB とハッシュテーブルのメモリ格納方法

しかし，これだとどの指紋を検索する際も必ず大容量メモリにアクセスしなければならない。そこで，現実には人気の音楽や流行の音楽があり，インターネット上を通信する音楽に偏りがあることを踏まえ，図 3.3(b) のようにデータを格納し，高頻度で検索要求される指紋を高速メモリだけで検索完了可能にするために図 3.4 のフローチャートの方法を採用した。これが階層化 Staged LSH である。ここで “高頻度指紋” とは人気があり検索要求されやすい音楽の指紋の集合であり，“低頻度指紋” とはそれ以外の音楽の指紋の集

合である。

検索時間は付録 A.3 より，指紋 DB のサイズ N_{DB} および 32 ビットあたりのメモリアクセス時間 T にほぼ正比例する。N_{DB} の増大は回避不能として，人気の音楽や流行の音楽に関して T を削減することで，平均的な検索時間を短縮するという狙いである。

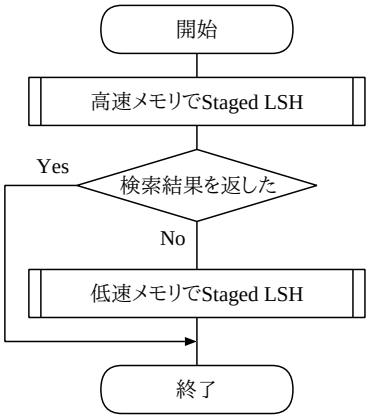


図 3.4: 階層化 Staged LSH のアルゴリズム (フローチャート)

3.3 隣接 LSH 探索による高速化・高精度化のトレードオフの調整

指紋の歪みによりハッシュ値が 1 ビットでも変化すると，基本的にはハッシュ探索は失敗する。しかし LSH では歪みが小さければハッシュ値が変化しないことがあり，その場合ハッシュ探索は成功する。

LSH のハッシュ探索の成功確率は，ハッシュ値が衝突しない確率とトレードオフの関係にある。いま指紋の各ビットが 0 と 1 である確率はどちらも 1/2 であるとし，指紋の歪み率 (ビットエラーの発生確率) を p_{ber} とすると，ハッシュ値が L_0 ビットであれば，

$$\text{ハッシュ探索の成功確率} = (1 - p_{ber})^{L_0} \tag{3.2}$$

$$\text{ハッシュ値の衝突確率} = \frac{1}{2^{L_0}} \tag{3.3}$$

となる。すなわち L_0 を小さくすればハッシュ探索は成功しやすくなるが，ハッシュ値も衝突しやすくなる。衝突が発生すると線形探索する対象が多くなるので検索速度の低下につながる。かと言って L_0 を大きくするとハッシュ探索が失敗しやすくなり，精度が低下する。

そこで検索速度と精度の両立のため，ハミング距離的に隣接するハッシュ値である “隣接ハッシュ”(図 3.5) も探索対象に含める “隣接 LSH 探索” 手法を提案する。これは概念的

には図 3.6 のようなアイデアであり，ハッシュ値のビット数 L_0 を増やして衝突確率を下げながら，隣接ハッシュを探索することでハッシュ探索の成功確率を上げる。

付録 A.2 では，指紋の各ビットの誤り率が一定以下であると仮定し，ハッシュ値を何ビットにして何隣接ハッシュまでを探索するべきかを確率計算により考察した。結果は，20 ビット，1 隣接である。

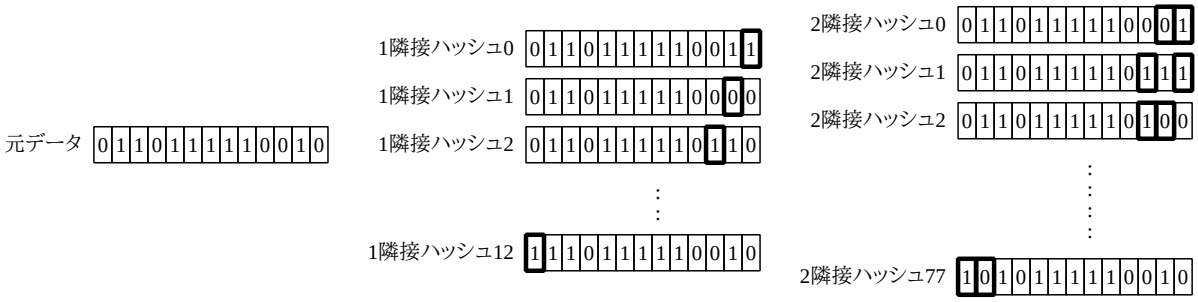


図 3.5: 隣接ハッシュの例

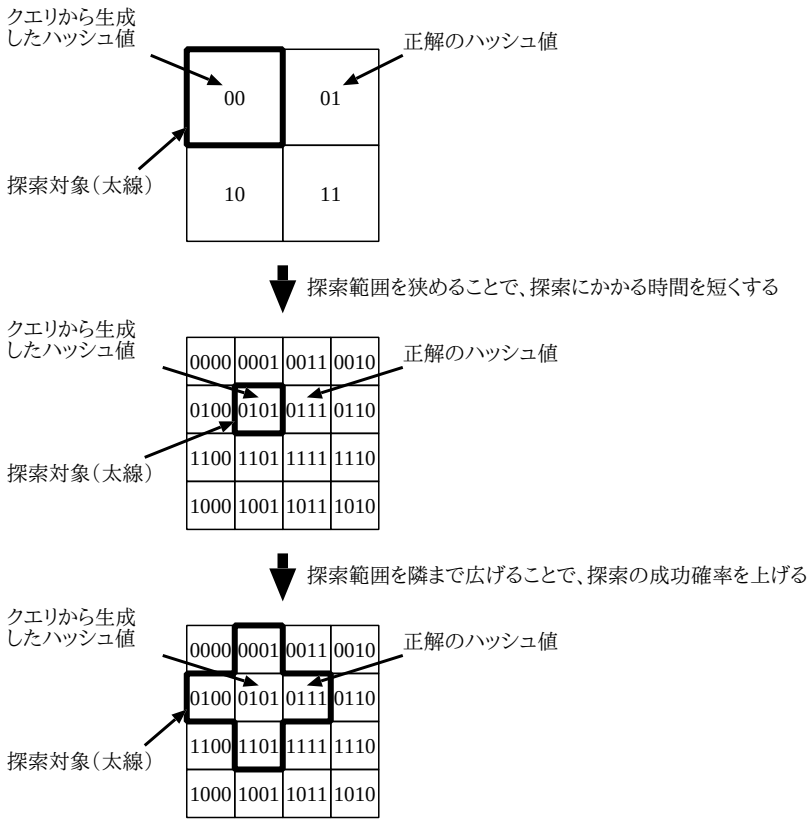


図 3.6: 隣接 LSH 探索のアイデア

3.4　提案システムの実装

図 3.7 に本研究の音楽指紋検索システムの内部構成図を，表 3.1 に構成図中の各モジュールの説明を，それぞれ示す。提案手法では，階層化 Staged LSH と隣接 LSH 探索の両方を “Staged LSH” のブロックに機能追加する。“デバッグ用” で囲んだ信号線は Ethernet(MII) または PCIe のインタフェースを制御するモジュールと接続しており，外部のパソコンが検索開始指示を出し検索結果を受け取るために使用した。Ethernet や PCIe は一般的なメモリ用のインタフェースではないが，FPGA からはメモリ制御モジュールの特定アドレスにデータを読み書きすることでこれらのインタフェースを制御可能とした。

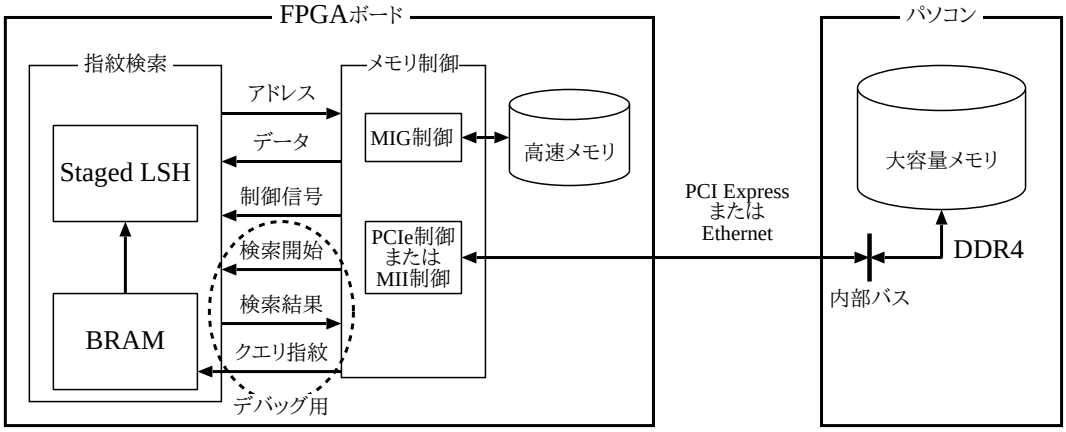


図 3.7: 音楽指紋検索のシステムと内部モジュールの構成図

表 3.1: 音楽指紋検索システムの内部モジュール一覧

モジュール名	説明
指紋検索	クエリ指紋を受け取り検索結果を返す。
Staged LSH	Staged LSH 手法によりクエリ指紋を検索する。 提案手法では隣接 LSH 探索も含む。
BRAM	クエリ指紋や一時データを格納する。
メモリ制御	DDR3 の高速メモリを 64 ビットアドレスに一元的にマッピングする。
MIG 制御	DDR3 経由で FPGA ボード内蔵の DRAM(高速メモリ) のデータを読み出す。Xilinx のライブラリを含む。
PCIe 制御 または MII 制御	PCIe または Ethernet 経由でパソコンから指紋 DB・ハッシュテーブル・クエリ指紋，そして検索開始指示を受け取る。検索完了後は結果を返す。

この構成図において指紋を検索するのは FPGA である。デスクトップ PC は，CPU を用いて指紋検索させることはなく，基本的にはデバッグと大容量メモリとしてのみ使用する。より正確には次の 3 つの用途である。

- FPGA へ検索開始を 指示する
- FPGA から 検索結果を 受け取り，画面に 表示する
- FPGA へ指紋 DB などを 転送する (検索前& 検索中)

3.5 まとめ

Yang の Staged LSH は音楽検索のハード ウェア 化と 高速化において 優れた 方法であるが，実際に 実装すると 大容量のメモリが 必要であり，そのままでは千万曲規模の音楽DB には適さなかった。そこで高速・小容量の DRAM に加えて 低速・大容量の DRAM を 用意することで，千万曲分の指紋 DB とハッシュテーブルを 用いた 検索を 実現するとともに，高精度の音楽に 関するデータを 高速・小容量の DRAM に格納することで平均的な検索速度の向上を 図った。併せて 検索速度と 検索精度の両立という 課題に対応するため，隣接ハッシュまで探索対象を 広げる 隣接 LSH 探索を 考案し，より 柔軟に 検索速度と 検索精度を 調整可能にした。そして 階層化 Staged LSH と 隣接 LSH 探索も 含めた Staged LSH を FPGA に実装し，音楽指紋検索システムを 開発した。

第4章　評価実験

提案手法である階層化 Staged LSH と隣接 LSH 探索を評価するための実験を行う。

4.1　現実の音楽から生成した指紋 DB での評価

まずは現実の音楽から生成した指紋を用いて，実装した音楽指紋検索システムの，特に隣接 LSH 探索による検索速度と検索精度への影響を評価する。この評価では，音楽指紋 DB のサイズは 300 曲までであり，複数メモリを用いた実験も実施しない。

4.1.1　開発環境と実験環境

FPGA の開発環境を表 4.1 にまとめる。回路合成からビットストリーム生成まで通常は数十分以上かかるため，高性能のサーバを使用することでビルド時間を短縮した。開発用ツールおよびライブラリは，現時点で最新のものを使用した。

表 4.1: FPGA の開発環境

項目	説明
CPU	AMD Opteron(TM) Processor 6238 (48コア)
メモリ	DDR3 SDRAM 271GB
ストレージ	ST320DM000-1BC14C (SATA 3.0, 320GB)
OS	Scientific Linux 6.1
カーネル	2.6.32-504.30.3.el6 (64ビット版)
開発ツール	ISE Design Suite 14.7
MIG ライブラリ	Memory Interface Generator 3.92

表 4.2 に，実験に用いた FPGA ボードの特徴を簡単に示す。これらの機材は市販のものをそのまま使用しており，未改造の状態である。

表 4.2: FPGA ボードの特徴

項目	説明
製品名	Xilinx 社 Virtex®-6 FPGA ML605 評価キット
FPGA	Virtex-6 XC6VLX240T-1FFG1156
クロック	200MHz (MIG ライブラリが出力)
搭載メモリ	DDR3 SO-DIMM，512MB

4.1.2 隣接 LSH 探索の評価

実験パラメータ

実験条件· 実験パラメータは表 4.3 の通りである。ハッシュ値のビット数 L_0 と隣接 LSH 探索のハミング半径 r_n については，Yang の手法は文献 [4] のものであり，提案手法は付録 A.2 で求めた最適値である。Yang の手法のスクリーニングと精査の閾値は文献 [4] の 5.2.2 節を参考にして，25%までの歪み率を想定して決定した。

表 4.3: 隣接 LSH 探索の評価実験の条件

項目	Yang の手法	提案手法
試行回数	300	300
クエリ指紋の歪み率 p_{ber}	0〜0.5	0〜0.5
スクリーニングの閾値 ε_1	24	24
精査の閾値 ε_2	1024	1024
ハッシュのビット数 L_0	13	20
隣接 LSH 探索のハミング距離 r_n	0	0

なお指紋の元となった音楽は Yang と同じものであり，300 曲を指紋 DB 化した。クエリ指紋の歪み (ビット誤り) についても，Yang と同様に乱数で機械的に付与した。

検索速度

現実の音楽データでまずは検索速度を評価する。Yang が用いた 300 曲から HiFP2.0 で指紋 DB とハッシュテーブルを構築し，メモリに全データを格納する。その後クエリ指紋として，300 曲の指紋にランダムに歪みを付与したものを検索システムに入力し，検索時間を測定する。

表 4.4，図 4.1 が実験結果である。提案手法の方が平均検索速度において約 6.3 倍高速であった。

表 4.4: Yang の手法と 隣接 LSH 探索の検索時間

歪み率	Yang の手法の 検索時間 (クロック数)	提案手法の 検索時間 (クロック数)
0.00	3,459	781
0.05	6,495	970
0.10	12,850	1,597
0.15	26,838	3,259
0.20	64,244	8,610
0.25	324,776	75,706
0.30	410,223	71,466
0.35	403,908	66,413
0.40	403,561	66,211
0.45	403,553	66,208
0.50	403,553	66,208

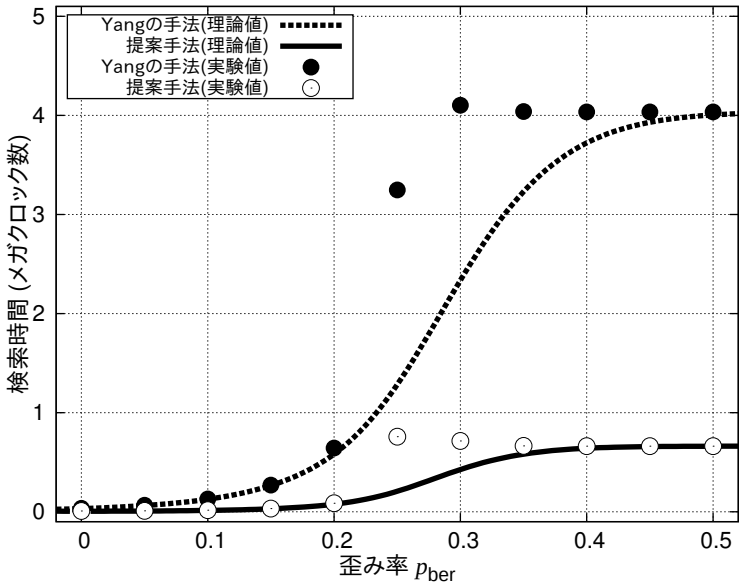


図 4.1: 歪み率と 平均検索時間

検索精度

検索精度については図 4.2 の通り Yang の手法も隣接 LSH 探索も同程度であった。なお不正解指紋を返した件数 (“一致なし” ではなく別の指紋を誤って返した件数) は 0 であったため，図表には正答率のみを記載した。

表 4.5: Yang の手法と 隣接 LSH 探索の検索精度

歪み率	Yang の手法の 正答率	提案手法の 正答率
0.00	1.00	1.00
0.05	1.00	1.00
0.10	1.00	1.00
0.15	1.00	1.00
0.20	0.41	0.42
0.25	0.00	0.00
0.30	0.00	0.00
0.35	0.00	0.00
0.40	0.00	0.00
0.45	0.00	0.00
0.50	0.00	0.00

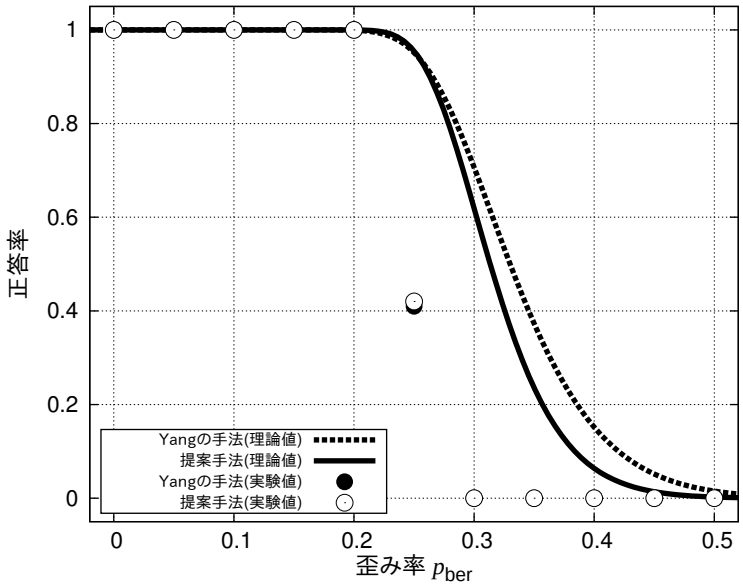


図 4.2: 歪み率と 検索精度

4.1.3 リソース使用量

実装した指紋検索モジュール (Staged LSH モジュール) のリソース使用量は表 4.6 の通りである。隣接 LSH 探索を追加実装しても最大で約 1.5%の増加であった。

表 4.6: 実装した指紋検索モジュール (Staged LSH モジュール) のリソース使用量

項目	従来手法	提案手法
Number of Slice Registers	467	471
Number of Slice LUTs	2206	2235
Number of fully used LUT-FF pairs	329	334
Number of bounded IOB	115	115
Number of BUFG/BUFGCTRLs	1	1

4.2 千万曲分の乱数指紋 DB での評価

今度は千万曲分の指紋 DB を乱数で生成して提案手法の評価を行う。指紋 DB が 300 曲の時よりも検索時間が長くなることが予想されるので，ネットワーク機器のデータ転送速度にどこまで追いつけるかを調べるとともに，階層化 Staged LSH および隣接 LSH 探索による検索速度の向上や精度への悪影響の程度を見る。

4.2.1 開発環境と実験環境

FPGA の開発環境を表 4.7 にまとめる。開発ツールが ISE Design Suite 14.7 から Vivado 2015.2 へ変更したこと以外は前節と同様である。

表 4.8，表 4.9 に，実験に用いた FPGA ボードとデスクトップ PC の特徴をそれぞれ簡単に示す。これらの機材は市販のものをそのまま使用しており，改造などはしていない。なお FPGA ボードには 4GB のメモリが 2 つあるが，実験内容を単純にするため片方のみ使用した。デスクトップ PC についてはメモリと PCIe の速度に特化したものを購入した。

4.2.2 階層化 Staged LSH の評価

階層化 Staged LSH 単独の評価を行う。このために隣接 LSH 探索を無効化する。千万曲分の指紋 DB を乱数で生成し，それを元にハッシュテーブルを構築する。クエリ指紋については各指紋が検索要求される頻度は Zipf 則に従うとし，一定の試行回数だけ検索システムに入力して平均検索時間と精度を測定する。その他の実験条件は表 4.10 の通りである。

表 4.7: FPGA の開発環境

項目	説明
CPU	AMD Opteron(TM) Processor 6238 (48 コア)
メモリ	DDR3 SDRAM 271GB
ストレージ	ST320DM000-1BC14C (SATA 3.0, 320GB)
OS	Scientific Linux 6.1
カーネル	2.6.32-504.30.3.el6 (64 ビット版)
開発ツール	Vivado 2015.2
MIG ライブラリ	Memory Interface Generator 7 Series Version 2.3 (Rev. 2)
PCIe ライブラリ	Virtex-7 FPGA Gen3 Integrated Block for PCI Express Version 4.0 (Rev. 1)

表 4.8: FPGA ボード の特徴

項目	説明
製品名	Xilinx 社 Virtex®-7 FPGA VC709 コネクティビティキット
FPGA	Virtex-7 XC7VX690T-2FFG1761C
クロック	250MHz (PCIe ライブラリ が出力)
搭載メモリ	DDR3 SO-DIMM，2 個，各 4GB，理論性能は 10.4GB/s
PCIe	PCI Express 3.0 x8，理論性能は 6.8GB/s

表 4.9: デスクトップ PC の仕様

項目	説明
CPU	Intel®Xeon®CPU E5-1620 v3 (4 コア)
メモリ	DDR4 33.6GB (実験では 28GiB 使用)
ストレージ	DT01ACA050 (SATA 3.0, 500GB)
I/O	PCI Express 3.0 x8 など
OS	Ubuntu 14.04.2 Long Term Support
カーネル	3.16.0-45-generic (64 ビット版)

表 4.10: 階層化 Staged LSH の評価実験の条件

項目	Yang の手法の単純拡張	提案手法
試行回数	300	300
クエリ 指紋の歪み率 p_{ber}	0〜0.25	0〜0.25
スクリーニングの閾値 ε_1	24	24
精査の閾値 ε_2	1024	1024
ハッシュのビット 数 L_0	13	13
隣接 LSH 探索のハミング距離 r_n	0	0
高速メモリ からの 32 ビット の平均読出時間 T_H	19 ナノ 秒	19 ナノ 秒
大容量メモリ からの 32 ビット の平均読出時間 T_L	580 ナノ 秒	580 ナノ 秒
メモリ 格納方法	図 3.3(a)	図 3.3(b)

表 4.11，図 4.3 が実験結果である。階層化 Staged LSH 有効の方が平均検索速度において歪み無し 時に 1.3 倍高速，最大で 4.3 倍高速であり，精度は互角であった。なお図中の “1Gbps の壁” および “10Gbps の壁” とは，1MB の音楽データを 1Gbps，10Gbps のネットワーク機器を 通過するまでの時間であり，3.1 節の方法で計算した。“256kbps”，“192kbps”，“128kbps MP3 の圧縮による 歪み率” とは，lame というフリーソフト を 用いて 各ビットレート の MP3 へエンコード した時の実験的な 最大歪み率であり，2.4 節の方法で 求めた。

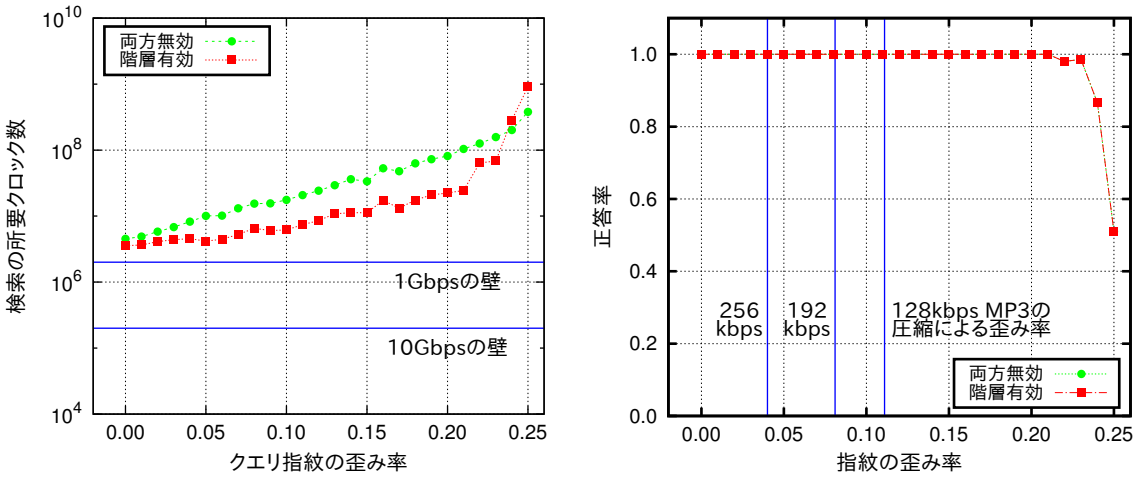


図 4.3: 階層化 Staged LSH の評価の歪み率と 平均検索時間・検索精度

表 4.11: 階層化 Staged LSH の評価実験の結果

歪み率	Yang の手法の単純拡張		提案手法	
	平均検索時間 (クロック)	正答率	平均検索時間 (クロック)	正答率
0.00	4.53×10^6	1.000	3.57×10^6	1.000
0.01	4.90×10^6	1.000	3.65×10^6	1.000
0.02	5.83×10^6	1.000	4.11×10^6	1.000
0.03	6.81×10^6	1.000	4.40×10^6	1.000
0.04	8.24×10^6	1.000	4.55×10^6	1.000
0.05	1.01×10^7	1.000	4.18×10^6	1.000
0.06	1.02×10^7	1.000	4.46×10^6	1.000
0.07	1.31×10^7	1.000	5.26×10^6	1.000
0.08	1.54×10^7	1.000	6.54×10^6	1.000
0.09	1.56×10^7	1.000	6.04×10^6	1.000
0.10	1.76×10^7	1.000	6.17×10^6	1.000
0.11	2.09×10^7	1.000	7.39×10^6	1.000
0.12	2.42×10^7	1.000	8.54×10^6	1.000
0.13	2.94×10^7	1.000	1.08×10^7	1.000
0.14	3.63×10^7	1.000	1.13×10^7	1.000
0.15	3.36×10^7	1.000	1.13×10^7	1.000
0.16	5.29×10^7	1.000	1.73×10^7	1.000
0.17	4.79×10^7	1.000	1.32×10^7	1.000
0.18	6.29×10^7	1.000	1.75×10^7	1.000
0.19	7.27×10^7	1.000	2.09×10^7	1.000
0.20	8.15×10^7	1.000	2.24×10^7	1.000
0.21	1.04×10^8	1.000	2.44×10^7	1.000
0.22	1.26×10^8	0.980	6.39×10^7	0.980
0.23	1.58×10^8	0.986	6.86×10^7	0.986
0.24	2.02×10^8	0.866	2.85×10^8	0.866
0.25	3.80×10^8	0.510	9.12×10^8	0.510

実験的には，階層化 Staged LSH は，検索速度を 1.3 倍 (歪み無し 時) または 4.3 倍 (歪み有りも含めた最大値) に向上し，検索精度は互角であった。もし 正解の指紋が大容量メモリにある場合には，階層化 Staged LSH は高速メモリで 126 回ものハッシュ探索とスクリーニングを行うので，むしろ図 3.3(a) の素朴なやり 方が高速である 可能性も残っている。しかし 今回の実験パラメータでは，各試行において高頻度指紋が検索要求される確率は合計 94.5%，低頻度指紋は 5.5%と大きな差があり，このことが階層化 Staged LSH を有利にしたと考えられる。

4.2.3 隣接 LSH 探索の評価

隣接 LSH 探索の単独の評価をする。このために階層化 Staged LSH を無効化する。指紋 DB，ハッシュテーブル，クエリ指紋については前の節と同じものを使用する。その他の実験条件は表 4.12 の通りである。

表 4.12: 隣接 LSH の評価実験の条件

項目	Yang の手法の単純拡張	提案手法
試行回数	300	300
クエリ指紋の歪み率 p_{ber}	0〜0.25	0〜0.25
スクリーニングの閾値 ε_1	24	24
精査の閾値 ε_2	1024	1024
ハッシュのビット数 L_0	13	20
隣接 LSH 探索のハミング距離 r_n	0	1
高速メモリからの 32 ビットの平均読出時間 T_H	19 ナノ秒	19 ナノ秒
大容量メモリからの 32 ビットの平均読出時間 T_L	580 ナノ秒	580 ナノ秒
メモリ格納方法	図 3.3(a)	図 3.3(a)

表 4.13，図 4.4 が実験結果である。隣接 LSH 有効の方が平均検索速度において歪み無し時に 5.9 倍高速，最大で 8.9 倍高速であり，精度は互角であった。

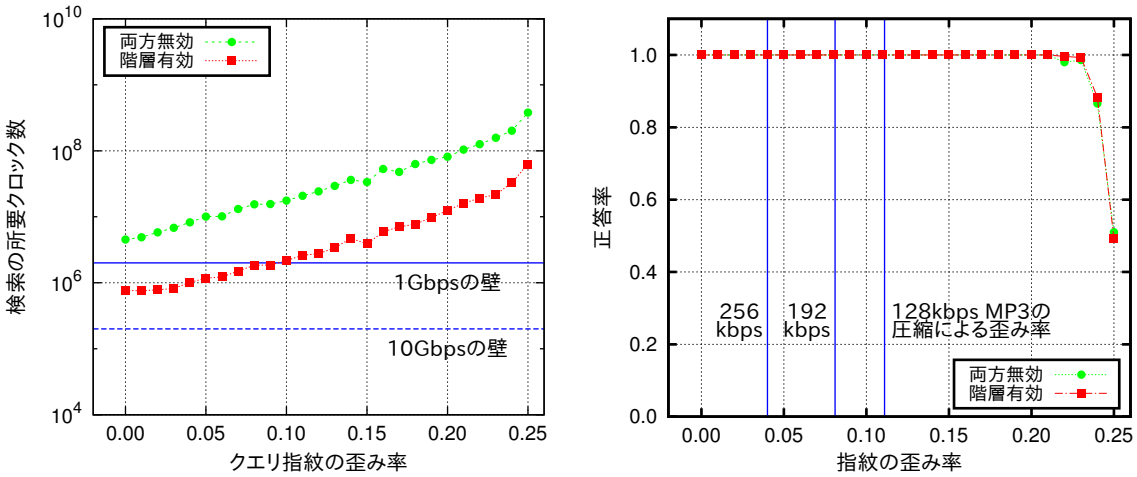


図 4.4: 隣接 LSH 探索の評価の歪み率と平均検索時間・検索精度

表 4.13: 隣接 LSH 探索の評価実験の結果

歪み率	Yang の手法の単純拡張		提案手法	
	平均検索時間 (クロック)	正答率	平均検索時間 (クロック)	正答率
0.00	4.53×10^6	1.000	7.64×10^5	1.000
0.01	4.90×10^6	1.000	7.56×10^5	1.000
0.02	5.83×10^6	1.000	7.82×10^5	1.000
0.03	6.81×10^6	1.000	8.18×10^5	1.000
0.04	8.24×10^6	1.000	1.02×10^6	1.000
0.05	1.01×10^7	1.000	1.17×10^6	1.000
0.06	1.02×10^7	1.000	1.23×10^6	1.000
0.07	1.31×10^7	1.000	1.51×10^6	1.000
0.08	1.54×10^7	1.000	1.84×10^6	1.000
0.09	1.56×10^7	1.000	1.86×10^6	1.000
0.10	1.76×10^7	1.000	2.17×10^6	1.000
0.11	2.09×10^7	1.000	2.61×10^6	1.000
0.12	2.42×10^7	1.000	2.76×10^6	1.000
0.13	2.94×10^7	1.000	3.39×10^6	1.000
0.14	3.63×10^7	1.000	4.73×10^6	1.000
0.15	3.36×10^7	1.000	3.88×10^6	1.000
0.16	5.29×10^7	1.000	5.93×10^6	1.000
0.17	4.79×10^7	1.000	7.03×10^6	1.000
0.18	6.29×10^7	1.000	7.72×10^6	1.000
0.19	7.27×10^7	1.000	9.76×10^6	1.000
0.20	8.15×10^7	1.000	1.25×10^7	1.000
0.21	1.04×10^8	1.000	1.61×10^7	1.000
0.22	1.26×10^8	0.980	1.92×10^7	0.996
0.23	1.58×10^8	0.986	2.17×10^7	0.993
0.24	2.02×10^8	0.866	3.26×10^7	0.883
0.25	3.80×10^8	0.510	6.29×10^7	0.493

今回の実験では，隣接 LSH 探索により検索速度は 5.8 倍 (歪み無し時) または 9.8 倍 (歪み有りも含めた最大値) に改善し，検索精度については互角であった。隣接 LSH 探索は，クエリ指紋の歪み率が比較的小さいという事前知識を利用して探索範囲を狭める手法である。今回の実験パラメータでは歪み率が 25%以下であり，それに合わせて実験条件 (ハッシュのビット数 L_0，隣接 LSH 探索のハミング距離 r_n) を机上の確率計算により設定することにより，検索精度をほぼ互角にしつつ，検索速度を向上することができた。

4.2.4 階層化 Staged LSH と隣接 LSH 探索の組み合わせの評価

階層化 Staged LSH と隣接 LSH 探索を両方有効化した場合の評価をする。指紋 DB，ハッシュテーブル，クエリ指紋については前説と同じものを使用する。その他の実験条件は表 4.14 の通りである。

表 4.14: 階層化 Staged LSH と隣接 LSH の評価実験の条件

項目	Yang の手法の単純拡張	提案手法
試行回数	300	300
クエリ指紋の歪み率 p_{ber}	0〜0.25	0〜0.25
スクリーニングの閾値 ε_1	24	24
精査の閾値 ε_2	1024	1024
ハッシュのビット数 L_0	13	20
隣接 LSH 探索のハミング距離 r_n	0	1
高速メモリからの 32 ビットの平均読出時間 T_H	19 ナノ秒	19 ナノ秒
大容量メモリからの 32 ビットの平均読出時間 T_L	580 ナノ秒	580 ナノ秒
メモリ格納方法	図 3.3(a)	図 3.3(b)

表 4.15，図 4.5 が実験結果である。隣接 LSH 有効の方が平均検索速度において歪み無し時に 7.5 倍高速，最大で 27.0 倍高速であり，精度は互角であった。

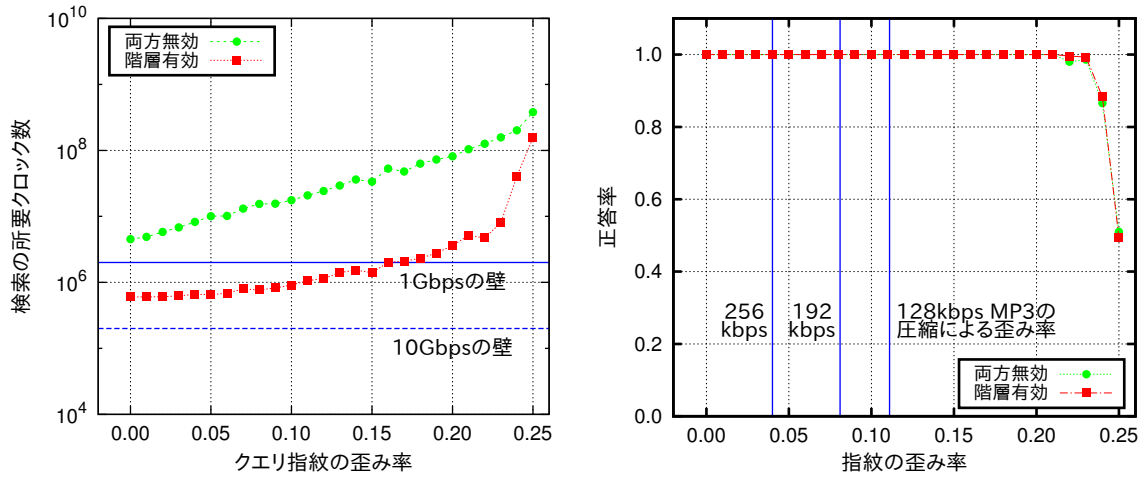


図 4.5: 階層化 Staged LSH と隣接 LSH 探索の評価の歪み率と平均検索時間・検索精度

表 4.15: 階層化 Staged LSH と隣接 LSH 探索の評価実験の結果

歪み率	Yang の手法の単純拡張		提案手法	
	平均検索時間 (クロック)	正答率	平均検索時間 (クロック)	正答率
0.00	4.53×10^6	1.000	6.03×10^5	1.000
0.01	4.90×10^6	1.000	6.04×10^5	1.000
0.02	5.83×10^6	1.000	6.07×10^5	1.000
0.03	6.81×10^6	1.000	6.26×10^5	1.000
0.04	8.24×10^6	1.000	6.58×10^5	1.000
0.05	1.01×10^7	1.000	6.52×10^5	1.000
0.06	1.02×10^7	1.000	6.86×10^5	1.000
0.07	1.31×10^7	1.000	7.97×10^5	1.000
0.08	1.54×10^7	1.000	7.69×10^5	1.000
0.09	1.56×10^7	1.000	8.24×10^5	1.000
0.10	1.76×10^7	1.000	9.11×10^5	1.000
0.11	2.09×10^7	1.000	1.06×10^6	1.000
0.12	2.42×10^7	1.000	1.14×10^6	1.000
0.13	2.94×10^7	1.000	1.41×10^6	1.000
0.14	3.63×10^7	1.000	1.54×10^6	1.000
0.15	3.36×10^7	1.000	1.40×10^6	1.000
0.16	5.29×10^7	1.000	2.01×10^6	1.000
0.17	4.79×10^7	1.000	2.09×10^6	1.000
0.18	6.29×10^7	1.000	2.33×10^6	1.000
0.19	7.27×10^7	1.000	2.78×10^6	1.000
0.20	8.15×10^7	1.000	3.62×10^6	1.000
0.21	1.04×10^8	1.000	5.12×10^6	1.000
0.22	1.26×10^8	0.980	4.72×10^6	0.996
0.23	1.58×10^8	0.986	8.03×10^6	0.993
0.24	2.02×10^8	0.866	4.03×10^7	0.883
0.25	3.80×10^8	0.510	1.55×10^8	0.493

階層化 Staged LSH と隣接 LSH 探索は同時に両方有効化できるものであり，片方だけを有効化した時よりも高速化が期待できる。今回の実験では，検索速度については 7.5 倍 (歪み無し時) または 27.0 倍 (歪み有りも含めた最大値) に改善した。これは今までのどの方法 (階層化 Staged LSH のみ有効，隣接 LSH 探索のみ有効，両方無効) よりも高速である。検索精度は隣接 LSH 探索と同じであった。

4.2.5 リソース使用量

階層化 Staged LSH および隣接 LSH 探索を含めた指紋検索モジュール (Staged LSH モジュール) のリソース使用量は表 4.16 の通りである。

表 4.16: 実装した指紋検索モジュール (Staged LSH モジュール) のリソース使用量

項目	使用量	使用可能量	使用率
FF	463	866,400	0.05 %
LUT	1,190	433,200	0.27 %
Memory LUT	304	174,200	0.17 %
I/O	0	850	0.00 %
BRAM	0	1,470	0.00 %
BUFG	0	32	0.00 %
MMCM	0	20	0.00 %
PLL	0	20	0.00 %
GT	0	45	0.00 %

4.3 まとめ

まず，大容量メモリのおかげで，千万曲分の指紋 DB とハッシュテーブル (合計 10GiB 程度) を用いた指紋検索実験が可能になった。本稿では乱数で千万曲分の指紋 DB を構築して，クエリ指紋に一定の歪みを付加して検索実験を行った。今回の実験パラメータでは，階層化 Staged LSH は検索速度を 1.3 倍 (歪み無し時) または 4.3 倍 (歪み有りも含めた最大値)，隣接 LSH 探索は検索速度を 5.8 倍 (歪み無し時) または 9.8 倍 (歪み有りも含めた最大値) にそれぞれ改善し，両方組み合わせると 7.5 倍 (歪み無し時) または 27.0 倍 (歪み有りも含めた最大値) になった。検索精度についてはほぼ互角であった。階層化 Staged LSh は検索速度を高速化する手法であり，隣接 LSH 探索は検索速度と検索精度を柔軟に調整する手法である。今回は検索精度を維持しつつ検索速度を高速化したが，これは付録 A.2 で定めた隣接 LSH 探索のパラメータの狙い通りである。

第5章　まとめと今後の課題

ギガビット以上のネットワーク機器内でリアルタイムに音楽を検索できるシステムの実用化を目指し，音楽指紋検索の高速化・高精度化・大規模DB化の研究を進めてきた。先行研究の指紋技術によって，ある程度の検索の高速化とメモリ削減は達成されているが，それでも千万曲規模になると指紋DBが10GBを超える。本研究では組込みシステムを想定しているためFPGAで実験をしているが，市販のFPGAボード内蔵のDRAM(高速メモリ)には容量不足で10GBを保存できず，実験すらままならない。

そこで本研究ではPCIe経由でデスクトップPCのDRAM(大容量メモリ)にアクセス可能にした。さらに高頻度で検索要求される音楽のデータを高速メモリに，それ以外の音楽のデータを大容量メモリに格納することで，高頻度の音楽を高速メモリだけで検索可能にする階層化Staged LSHを提案した。

加えて，検索速度と検索精度のトレードオフをより細かく調整可能な隣接LSHを考案した。確率計算に基づくパラメータのチューニングにより，検索精度を維持しつつ検索時間を向上できる。

この階層化Staged LSHと隣接LSHを組み合わせて実験したところ，クエリ指紋に歪みがない場合に検索時間を7.5倍に向上，歪みがある場合も含めると最大で27.0倍に向上できた。歪みがない場合には，提案手法無効時に平均4.18×10^6クロックであったのが，提案手法有効時に平均4.37×10^5クロックであり，3.1節に示した目標2×10^6クロックを達成した。また検索精度についても維持できていた。

今後の課題を以下に挙げる。

- 重要な課題の1つは，高速メモリと大容量メモリのデータの入れ替えるアルゴリズムの開発である。本研究では高速メモリに高頻度で検索要求される音楽のデータを，大容量メモリにそれ以外の音楽のデータを格納するとしたが，その割り当ては固定的であった。現実には流行の音楽はその時々によって変わるので，各音楽の検索頻度の移り変わりによって，どのような入れ替えアルゴリズムを適用するべきか検討する余地がある。

- 類似指紋の区別も重要な課題である。音楽指紋は音楽の聴覚的特徴をもとに生成されるため，類似の音楽は類似の指紋になる可能性がある。これをいかにして区別するか検討する必要がある。

参考文献

[1] K. Araki, Y. Sato, V. Jain and Y. Inoguchi, "Performance Evaluation of Audio Fingerprint Generation using Haar Wavelet Transform," Proceedings of the 2011 International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP11), pp.380-383, Tianjin SaiXiang Hotel, Tianjin, China, 2011.

[2] J. Haistma and T. Kalker, "A Highly Robust Audio Fingerprinting System," Proceedings of the International Symposium on Music Information Retrieval, 2002.

[3] H. Schreiber and M. Muller, "Accelerating Index-Based Audio Identification," IEEE Transactions on Multimedia, Volume 16, Issue 6, 2014.

[4] F. Yang, "Hardware Acceleration of Searching Strategy for Audio Fingerprinting System," Master's thesis of Information Science, Japan Advanced Institute of Science and Technology, 2013.

謝辞

本研究を行うにあたり，研究室のゼミやミーティングやその他の様々な場面で，手厚い研究の御指導や情報機器の説明などを受け賜りました北陸先端科学技術大学院大学の情報社会基盤センター井口 寧教授に深く感謝するとともに御礼申し上げます。

中間審査会で貴重な御助言，御意見を頂いた金子 峰雄教授，田中 清史准教授に深く感謝いたします。ハードウェアグループ合同ゼミでも金子研，田中研，金沢大学，福井大学の方々にはお世話になりましたので感謝と御礼を申し上げます。

井口研究室のゼミや普段の会話で，多くの御意見，御指導を頂いた Tan Yiyu 氏，佐々木 泰氏，中吉 達二氏，大和 良介氏，赤崎 翔太氏，河村 知記氏，Faiz Al Faisal 氏に感謝と御礼を申し上げます。OB として多くの御意見，御指導を頂いた佐藤 幸紀特任講師 (東京工業大学)，荒木 光一氏，Fan Yang 氏に感謝と御礼を申し上げます。

情報セキュリティコースでお世話になった宮地 充子教授，面 和成准教授，布田 裕一特任准教授，田中 覚特任助教，木藤 圭亮氏，STEWART, Gavin Lynn 氏，道廣 大喜氏，足立 大地氏，岡村 宗一郎氏に感謝いたします。情報セキュリティの考え方が研究室や自宅のサーバ運用の役に立っている他，確率計算や誕生日のパラドックスの考え方は本研究の理論式の導出にも活用できました。

北陸先端科学技術大学院大学の授業や事務手続きなどでお世話になった全ての方々に御礼を申し上げます。

最後に，日頃より暖かく見守ってくださった両親，親族に感謝いたします。

付 録A　理論検索時間と理論検索精度

本章では複数メモリを用いた階層化Staged LSHと隣接LSH探索の理論検索時間と理論検索精度について考察する。

A.1　変数と関数の記号

変数の一覧を表A.1に，関数の一覧を表A.2にそれぞれ示す。

表 A.1: 変数の一覧

記号	説明
N_{DB}	指紋DBの音楽数 (すなわち300または10,000,000)
p_{ber}	指紋の歪み率 (ビットエラー率)
L_0	ハッシュ値のビット数
L_1	フレームのビット数 (すなわち96)
L_2	指紋のビット数 (すなわち4096)
n_m	ハッシュ探索の最大実行回数 (すなわち126)
r_n	隣接LSH探索のハミング半径 (隣接LSH探索無効時は0)
ε_1	スクリーニングのハミング距離の閾値 (すなわち24)
ε_2	精査のハミング距離の閾値 (すなわち1024)

表 A.2: 関数の一覧

記号	説明
$\mathrm{Hash}(x_1)$	x_1のハッシュ値を求める関数
$\mathrm{HD}(x_1, x_2)$	x_1とx_2のハミング距離を求める関数

A.2 検索精度に関する考察

本節では Staged LSH におけるハッシュ探索，スクリーニング，精査の各工程を正解指紋がパスする確率を求め，検索精度に大きな影響を与えるハッシュ値のビット数について 1 つの最適解を求める。

ここでいう正解指紋とは、クエリ指紋の元となった音楽と同じ音楽から生成された指紋のことである。逆に不正解指紋とは，クエリ指紋の元となった音楽とは別の音楽から生成された指紋のことである。“一致なし”という検索結果も存在するが，これは正解指紋とも不正解指紋とも異なる。

精度には指紋生成アルゴリズムに起因するものと指紋検索アルゴリズムに起因するものが考えられるが，本稿では指紋生成に関しては十分な精度が保証されている想定で，指紋検索に関する精度を考察する。

A.2.1 不正解指紋が精査をパスしてしまう確率

まずは検索精度について考える。検索精度を確保する上では，異なる音楽の指紋のハミング距離がある程度離れていることが最低限必要である。できれば ε_2 より大きいことが望ましい。さもなくば，不正解の音楽を検索結果として返す恐れが生じる。誕生日のパラドックスによれば，乱数を大量に発生させると，同じ値が複数回発生 (衝突) している確率は直感よりも高い。ここでは 4096 ビットの HiFP2.0 の指紋について誕生日のパラドックスを考慮しても千万曲の指紋同士が互いに ε_2 より離れていることを確認しておきたい。

まず，2 指紋 R_{fp}, W_{fp} のハミング距離が ε_2 以下である確率は，指紋が乱数とみなせるならば次のように表される。

$$\begin{aligned} \overline{P}_{\overline{2},2} &= P[\mathrm{HD}(R_{fp}, W_{fp}) \leq \varepsilon_2] \\ &= \frac{1}{2^{L_2}} \sum_{i=0}^{\varepsilon_2} {}_{L_2}C_i \end{aligned} \tag{A.1}$$

これを 3 指紋，4 指紋，$\cdots$，N 指紋と増やしていくと，

$$\begin{aligned} \overline{P}_{\overline{2},N} &< 1 - \{1 - \overline{P}_{\overline{2},2}\}\{1 - 2\overline{P}_{\overline{2},2}\} \cdots \{1 - (N-1)\overline{P}_{\overline{2},2}\} \\ &\simeq 1 - \{1 - N\overline{P}_{\overline{2},2}\}^{(N-1)/2} \\ &\simeq 1 - \exp\left[\frac{-N(N-1)}{2}\overline{P}_{\overline{2},2}\right] \end{aligned} \tag{A.2}$$

となる。ここで途中計算の簡単化のため N は奇数，$N \cdot \overline{P}_{\overline{2},2} \ll 1$ とした。N が偶数の場合も最終的な式は同じである。この式を $L_2 = 4096, \varepsilon_2 = 1024, N = 10^7$ の条件で計算すると $\overline{P}_{\overline{2},N} \simeq 0$ である。つまり HiFP2.0 が生成する指紋が乱数に近ければ，千万曲の指紋の中にハミング距離が小さいものが 1 組以上存在する確率は無視できる程度に小さい。

実際には HiFP2.0 が生成する指紋は必ずしも乱数とはみなせない。特に 4096 ビットの先頭付近は 0 であることが多い。指紋 DB 内にハミング距離の近い指紋がある場合の対応については今後の課題とする。

なお数値計算には次の Mathematica スクリプトを実行した。

```
l2 = 4096;
e2 = 1024;
n = 10^7;
p22 = 2^(-l2) * Sum[l2!/k!/(l2-k)!, {k,0,e2}];
p2n = 1 - Exp[-n*(n-1)/2 * p22];
Print[N[p2n]];
```

A.2.2 正解指紋が精査をパスできない確率

歪んだクエリ指紋と正解指紋のハミング距離を考える。もしこのハミング距離が精査の閾値 ε_2 より大きいようでは，どれだけハッシュ探索とスクリーニングをチューニングしても検索は必ず失敗する。確率計算をすると，指紋は L_2 ビットであるから，歪み率 $p_{ber} > 0$ のクエリ指紋 Q_{fp} と正解指紋 R_{fp} のハミング距離が ε_2 より大きい確率は，

$$\begin{aligned} P_2 &= P[\mathrm{HD}(Q_{fp}, R_{fp}) > \varepsilon_2] \\ &= 1 - \sum_{i=0}^{\varepsilon_2} \left[{}_{L_2}C_i \cdot (p_{ber})^i (1-p_{ber})^{L_2-i} \right] \end{aligned} \tag{A.3}$$

である。$L_2 = 4096, \varepsilon_2 = 1024$ の条件の下，歪み率を変えながらこの式の確率を計算したものを図 A.1 に示す。歪み率 $p_{ber} \leq 0.22$ であれば，0.99 以上の確率で正解指紋は精査をパスする。HiFP2.0 の指紋の歪み率 (ビットエラー率) は文献 [1] では最大 0.15 であり，この時 $P_2 = 1.0 \times 10^{-62}$ であるので，正解指紋が精査をパスできない心配はほとんど不要である。

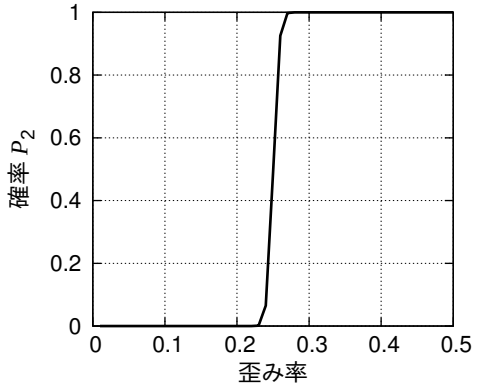


図 A.1: クエリ指紋の歪みにより正解指紋が精査をパスしない確率

なお数値計算には次の Mathematica スクリプトを実行した。

```
l2 = 4096;
e2 = 1024;
n = 10^7;
For[i = 1, i <= 50, i = i + 1,
    pber = i / 100;
    p2 = 1 - Sum[l2!/(k!*(l2-k)!) * pber^k * (1-pber)^(l2-k), {k,0,e2}];
    Print[N[pber], " ", N[p2]];
];
```

A.2.3 正解指紋がスクリーニングをパスできない確率

スクリーニングについても，正解指紋がパスできないことはあまりない。クエリ指紋 Q_{fr} と正解指紋 R_{fr} のハミング距離が ε_1 より大きい確率は，フレームが L_1 ビットであるから，

$$\begin{aligned} P_1 &= P[\mathrm{HD}(Q_{fr}, R_{fr}) > \varepsilon_1] \\ &= 1 - \sum_{i=0}^{\varepsilon_1} \left[{}_{L_1}C_i \cdot (p_{ber})^i (1-p_{ber})^{L_1-i} \right] \end{aligned} \tag{A.4}$$

である。$L_1 = 96, \varepsilon_1 = 24$ の条件の下，歪み率を変えながらこの式の確率を計算したものを図 A.2 に示す。歪み率 $p_{ber} \leq 0.16$ であればスクリーニングで検出できる確率は 0.99 以上である。

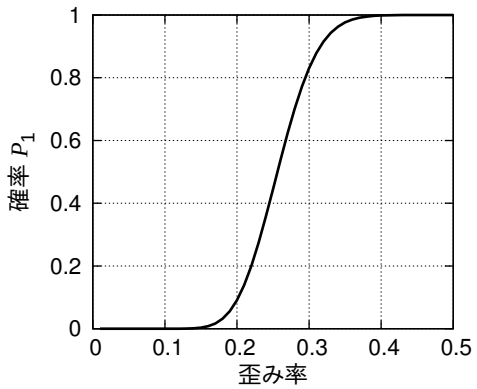


図 A.2: クエリ指紋の歪みにより正解指紋のフレームがスクリーニングをパスしない確率

なお数値計算には次の Mathematica スクリプトを実行した。

```
l1 = 96;
e1 = 24;
n = 10^7;
For[i = 1, i <= 50, i = i + 1,
    pber = i / 100;
    p1 = 1 - Sum[l1!/(k!*(l1-k)!) * pber^k * (1-pber)^(l1-k), {k,0,e1}];
    Print[N[pber], " ", N[p1]];
];
```

A.2.4 LSH のハッシュ値にビットエラーがある確率

以上から，検索全体の精度はほとんどハッシュ探索の成功確率のみに左右される。そこでクエリ指紋の 1 フレームのハッシュ値に r_n ビットより多くのビットエラーがある確率を求めると次の通りである。

$$\begin{aligned} \overline{P}_0 &= P[\mathrm{Hash}(Q_{fr}) \neq \mathrm{Hash}(R_{fr})] \\ &= 1 - \sum_{i=0}^{r_n} \left[{}_{L_0}C_i \cdot (p_{ber})^i (1-p_{ber})^{L_0-i} \right] \end{aligned} \tag{A.5}$$

これがハッシュ探索 1 回の失敗確率である。Staged LSH ではハッシュ探索を最大で n_m 回 $(n_m = (L_{fp} - L_{fr} + 32) \div 32 = 126)$ 繰り返す。n_m 回全て失敗する確率は

$$\overline{P}_{\mathrm{all}} = \overline{P}_0{}^{n_m} \tag{A.6}$$

である。最終的に，Staged LSH の検索全体としての精度は $1 - \overline{P}_{\mathrm{all}}$ にほぼ等しい。

$\overline{P}_{\mathrm{all}}$ は図 A.3 のようになる。歪み率 $p_{ber} \leq 0.25$ までを想定し，隣接 LSH の探索対象のハミング半径 $r_n = 0, 1, 2$ とした。

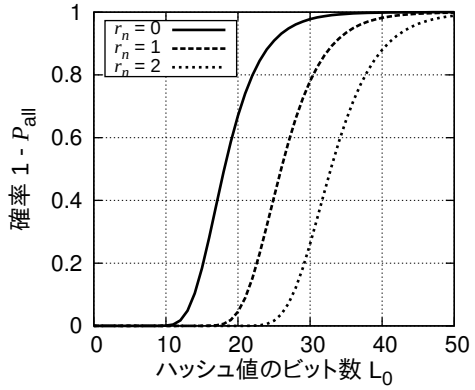


図 A.3: ハッシュ探索が 126 回全て失敗する確率

数値計算には次の Mathematica スクリプトを実行した。

```
nm   = 126;
pber = 0.25;
Print["0 0 0 0"];
For[l0 = 1, l0 <= 50, l0 = l0 + 1,
    pall0 = N[(1 - (1-pber)^l0)^nm];
    pall1 = N[(1 - Sum[l0!/(k!*(l0-k)!) * pber^k * (1-pber)^(l0-k), {k,0,1}])^nm];
    pall2 = N[(1 - Sum[l0!/(k!*(l0-k)!) * pber^k * (1-pber)^(l0-k), {k,0,2}])^nm];
    Print[l0, " ", pall0, " ", pall1, " ", pall2];
];
```

A.2.5 ハッシュ値のビット数の最適値

ハッシュ値のビット数 L_0 の最適解を求める。L_0 が大きいと，衝突の発生確率が下がるので検索速度は向上するが，クエリ指紋の歪みがハッシュ値に波及しやすくなるので精度が落ちる。またハッシュテーブルのメタデータが大きくなるという問題もある。

そこで一つの最適解は，クエリ指紋の歪み率 p_{ber} の許容上限を設定し，その範囲内で例えば $P_{\mathrm{all}} < 0.05$ になるような最大の L_0 を選ぶことである。図 A.3 において $P_{\mathrm{all}} < 0.05$ に抑えるには，$r_n = 0$ ならば $L_0 = 13$，$r_n = 1$ ならば $L_0 = 20$，$r_n = 2$ ならば $L_0 = 26$ がそれぞれ最適である。なお $r_n = 0$ については，Yang の研究 [4] でも $L_0 = 13$ が最適とされている。

ハッシュテーブルのメタデータのサイズについて考えると，図 2.14 のデータ構造ならば $2^{L_0} \cdot 4$ バイトになる。表 A.3 に示す通り，千万曲の指紋 DB とハッシュテーブル (本体データ) の合計サイズが約 10GB であることを考えると，いずれもメタデータのサイズは比較的小さい。ただし組込み用途ではコストの観点から 10GB の DRAM を用意するより，小容量の高速メモリと大容量の低速メモリを組み合わせて使うことが考えられる。その場合，256MiB もあると現時点では小容量の高速メモリを圧迫しかねないため，もう少しハッシュのビット数を減らすか，1 隣接を採用するべきである。

表 A.3: ハッシュ値のビット数とハッシュテーブルのメタデータのサイズ

ハッシュ値のビット数 L_0	メタデータのサイズ ($2^{L_0} \cdot 4$ バイト)
13	32KiB
20	4MiB
26	256MiB

A.3 検索時間の理論値

本節では検索時間の理論値を導出する。

A.3.1 1回のハッシュ探索とスクリーニングの平均実行時間

Staged LSH の検索時間の期待値を求めるには，1 回のハッシュ探索とそれに伴うスクリーニングと精査の平均実行時間 (図 2.10 のアルゴリズムの外側のループ 1 回) を求め，ハッシュ探索の実行回数の期待値を乗じれば良い。まずは 1 回のハッシュ探索とそれに伴うスクリーニングと精査の平均実行時間 $T_{\mathrm{all},1}$ を求める。メモリの 32 ビットのデータの平均転送時間を T クロックとおくと，

1. LSH のハッシュ値を求めるのに 1 クロック
2. ハッシュ値に対応するフレームリストのアドレス取得に T クロック
3. リスト内のフレームのアドレス取得に $\sum_{i=0}^{r_n} [{}_{L_0}C_i] \times (n_m \times N_{\mathrm{DB}} \div 2^{L_0}) \times T$ クロック
4. リスト内のフレーム取得に $\sum_{i=0}^{r_n} [{}_{L_0}C_i] \times (L_1 \div 32) \times (N_{\mathrm{DB}} \div 2^{L_0}) \times T$ クロック
5. リスト内のフレームのハミング距離の計算に $\sum_{i=0}^{r_n} [{}_{L_0}C_i] (N_{\mathrm{DB}} \div 2^{L_0}) \times 9$ クロック (ただし 1 つ上の項目と同時実行可能)
6. 精査を実行する場合，L_2 ビットの指紋の取得に $(L_2 \div 32) \times T$ クロック
7. 精査を実行する場合，L_2 ビットのハミング距離の計算に 134 クロック (ただし 1 つ上の項目と同時実行可能)

である。L_0 が十分大きくない限りにおいて，支配的なのは項目 3 と項目 4 であり，合計で

$$\begin{aligned} T_{\mathrm{all},1} &\simeq \sum_{i=0}^{r_n} [{}_{L_0}C_i] (n_m + L_1 \div 32) \times \frac{N_{\mathrm{DB}}}{2^{L_0}} \times T \\ &= 129 \sum_{i=0}^{r_n} [{}_{L_0}C_i] \times \frac{N_{\mathrm{DB}}}{2^{L_0}} \times T \end{aligned} \tag{A.7}$$

となる。

A.3.2 ハッシュ探索の実行回数の期待値

i 回目までのハッシュ探索で指紋を見つけられる確率を $P_{0,i}$ で現す。これは前述の LSH のハッシュ値にビットエラーがある確率 $\overline{P}_0$ を用いて次式で表される。$1 \leq i \leq n_m$ である。

$$P_{0,i} = 1 - \overline{P}_0{}^i \tag{A.8}$$

この $P_{0,i}$ を用いてハッシュ探索の実行回数を n_0 の期待値を求める。

$$\begin{aligned}
E[n_0] &= 1 \cdot P_{0,1} + 2 \cdot (P_{0,2} - P_{0,1}) + 3 \cdot (P_{0,3} - P_{0,2}) \\
&\quad + \cdots + (n_m - 1) \cdot (P_{0,n_m-1} - P_{0,n_m-2}) + n_m \cdot (1 - P_{0,n_m-1}) \\
&= n_m - P_{0,1} - P_{0,2} - \cdots - P_{0,n_m-1} \\
&= 1 + \sum_{i=1}^{n_m-1} (1 - P_{0,i}) \\
&= 1 + \sum_{i=1}^{n_m-1} \overline{P}_0{}^i \\
&= 1 + \overline{P}_0 \cdot \frac{1 - \overline{P}_0{}^{n_m-1}}{1 - \overline{P}_0} \\
&= \frac{1 - \overline{P}_0{}^{n_m}}{1 - \overline{P}_0} \\
&= \frac{P_{0,n_m}}{P_{0,1}}
\end{aligned} \tag{A.9}$$

以上を用いて，全体の検索時間 T_{all} の期待値は次で求められる。

$$E\left[T_{\mathrm{all}}\right] = E[n_0] \times T_{\mathrm{all},1} \tag{A.10}$$

図 A.4 に，次の 3 通りの条件で歪み率を変えながら $E[T_{\mathrm{all}}]$ を計算したグラフを示す。ここでは $T = 1$ クロックとした。いずれも歪み率 0.25 までのハッシュ探索失敗率が 0.05 未満になるように調整したパラメータであるが，検索時間は隣接 LSH 探索を有効にする方が短い。

- $r_n = 0,\ L_0 = 13$
- $r_n = 1,\ L_0 = 20$
- $r_n = 2,\ L_0 = 26$

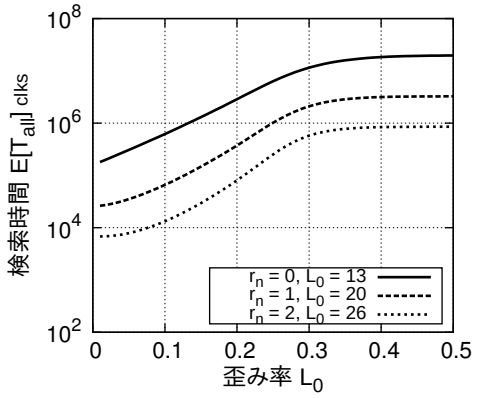


図 A.4: 全体の検索時間の期待値