論　文

Tiny LSHに基づいた大規模ビデオ検索

村林　昇[†]　　吉田　健一[†]

Tiny LSH for Content-Based Copied Video Detection

Noboru MURABAYASHI[†] and Kenichi YOSHIDA[†]

あらまし　近年，ブロードバンドネットワークの普及により様々なビデオサービスが実現している．このようなサービスは新しいビジネスチャンスを起こすものであるが，それに伴って違法に複製されたビデオコンテンツを検出する技術の重要性が増している．このような背景において技術的な課題は，WWW上に存在する膨大な量のビデオコンテンツから違法に複製されたビデオを検出することである．本論文では，大規模なビデオデータベースから所望のビデオ断片を実用的な精度と速度で検索可能な手法を提案する．実際のビデオコンテンツを用いた実験の結果，改変されていないビデオの場合，平均検索時間0.01秒，正解率100%で該当するビデオの検索ができること，改変されたビデオの場合もサイズの変換程度であれば著しい性能低下はないことを確認した．

キーワード　複製ビデオ，画像特徴量，LSH，Direct-mapped cache

1. まえがき

近年，ブロードバンドネットワークが普及し，ビデオコンテンツを利用する新しいサービスのためのインフラが実現した．それにより有望なビジネスチャンスが創出される一方で違法に複製されたビデオコンテンツがWWW上に増加し，このようなビジネスチャンス発展の弊害になっている．そのため，WWW上に存在する膨大な量の複製ビデオの検出技術が重要になっている．現状ではこのような違法に複製されたビデオを検出する手法として，電子透かしを適用したアプローチ[1]や，複製が疑われるビデオ（以下検索目標ビデオ）と同じビデオが，著作権が保護されるべきビデオを記憶したデータベース（以下原ビデオデータベース）中にあるか検索することに基づいたアプローチ[2]が代表的である．このような背景において技術的な課題は，原ビデオデータベースと検索目標ビデオの量が膨大なことである．

著名なオンラインビデオサービスの2009年5月の状況によれば，毎分20時間分のビデオがWWW上にアップロードされている[3]．アップロードされるビデオの平均時間長は約5分といわれており，毎日28,800時間（=24時間×60分×20時間/分），毎分にして平均240本（ビデオ1本当り5分として計算）にも及ぶ検索目標ビデオがアップロードされることになる．

仮に，10年分のビデオの著作権を保護することを考えた場合，87,600時間分（すなわち，24時間×365日×10年）の原ビデオデータベースと毎日アップロードされる28,800時間分の検索目標ビデオとを比較する必要がある．記録ビットレート1 Mbit/sのエンコードを仮定した場合，10年分の原ビデオデータベースのデータ量は39.42 TByteにも及び，毎日アップロードされる12.96 TByteの検索目標ビデオとの間で同一性を確認することになる．また検索目標ビデオには改変され原ビデオデータベースの内容と対応する画像が異なってしまったビデオも存在する．例えば，画像サイズが異なる場合や画像が部分的に切り落とされた場合，画質が良くない場合には，検索目標ビデオの検出は更に難しくなる．

本論文では，大規模な原ビデオデータベースから高速で精度良く検索目標ビデオを検索できる手法を提案する．以下，**2.**で関連研究と研究課題について概観し，**3.**で提案手法について説明し，**4.**で実験結果を説明する．**5.**では考察を行い，**6.**ではまとめと今後の課題について述べる．

[†] 筑波大学大学院ビジネス科学研究科，東京都
Graduate School of Business Sciences, University of Tsukuba, 3-29-1 Otsuka, Bunkyo-ku, Tokyo, 112-0012 Japan

2. 関連研究

ビデオの検索手法に関してはこれまで多くの研究が行われている．例えば，文献[4]はRGB色空間における画像特徴量を用いて入力された二つのビデオの一致する部分を照合する研究で，検索時間をかければ精度良く検索できることを報告している．ビデオの改変に対して耐性がある画像特徴量について，例えば，文献[5]や文献[6]などの研究も行われている．またデータを高速で検出する手法も研究が行われている．代表的な研究として，Locality-sensitive hashing (LSH) [7]，枝刈り[8]，フィルタリング[9]，及び木に基づいたデータ構造[2],[10]などがある．

本研究ではハッシュ法に基づく既存手法を大規模な複製ビデオの検出に応用することを考え，予備検討としてハッシュ値の分布特性を計測した．図1に，後述するビデオデータのRGB色空間における画像特徴量に基づいたハッシュ値の出現頻度の順位と出現頻度の関係を示す．図1に示すように，ハッシュ値の分布はべき乗則（図1における「従来のハッシュ」）に従っている．すなわちビデオデータは類似した画像フレームを多く含むので，画像特徴量に基づいたハッシュ値も同じものが多く存在することになる．

この特性から出現頻度の多いハッシュ値は衝突し，ハッシュ関数に基づいた検索手法では，この衝突が検索速度低下の原因となる．類似画像の高速検索手法であるLSHは類似データに対しては同じハッシュ値となるよう設計されており，ビデオデータに適用した場合，衝突の可能性は更に高くなり検索速度は低下する（図1における「Tiny LSH」(注1)）．木に基づいたデータ構造を適用した場合も，類似データの頻出により似たような性能上の課題がある．

上記のような既存手法の課題を背景に，我々は文献[11]において，実環境下において適用可能な複製ビデオの検出法として以下の手法を提案した．

（1） 検索目標ビデオを検索キーとして，著作権のあるビデオを全て一つのビデオとしてまとめた原ビデオデータベースを検索．

（2） 検索結果，すなわち原ビデオデータベース中一番似ていた部分が検索目標ビデオと一致する場合，複製ビデオを検出できたと判定．

（3） 大規模な原ビデオデータベースに対して精度良く高速で検索可能．具体的には，長さ5分の検索目標ビデオを136年分の原ビデオデータベースから0.1秒以下で検索可能．

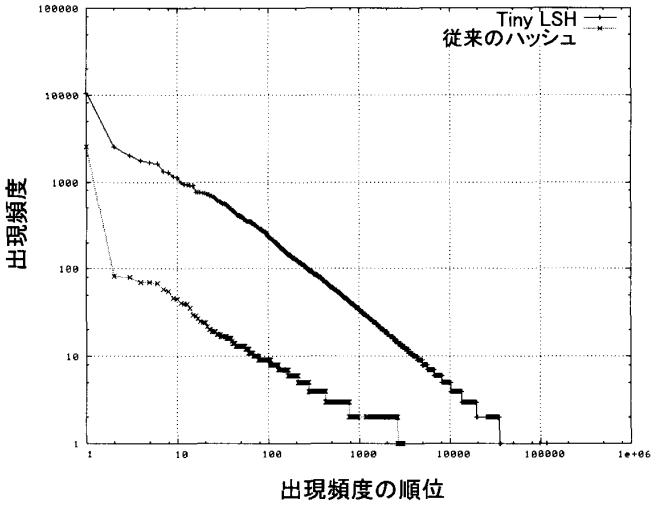


図1 ハッシュ値の出現順位と出現頻度の関係
Fig. 1 Distribution of hash values.

（4） 上記目的のため，LSHと同じく類似したフレームに同じハッシュ値を計算結果とするハッシュ関数構成のアイデア（以下Tiny LSHと記す）と，SPAMフィルタ[12]で適用された高速データ検索手法とを組み合わせて使用．

本論文では，文献[11]の研究に基づいて，テストデータとして109.5時間分のビデオコンテンツを用いた実験結果を，提案手法の特性に関する考察も含めて報告する．

3. Tiny LSHに基づいた大規模複製ビデオの検索手法

この章では，大規模な原ビデオデータベースに対応できる検索手法を提案する．手法の特徴は次の三つである．

- RGB色空間に基づいた画像特徴量の適用．
- サイズ変更など簡単なビデオの改変を扱うためのTiny LSHの適用．
- Tiny LSHにより記憶・索引されたデータを高速で検索するためDirect-mapped cacheの利用．

ここで，サイズ変更など簡単なビデオの改変を扱うのは，検索目標ビデオには，画面サイズやエンコードレートの変更など，単純な改変を伴う場合が多く，そのような単純な改変に対応するためである．

(注1)：厳密にいえば，図1に示す「Tiny LSH」は我々の使用したハッシュ関数（詳細は次の章を参照）により計算された値の分布を示している．しかしながら，本質的にはここで議論している特徴は同じである．

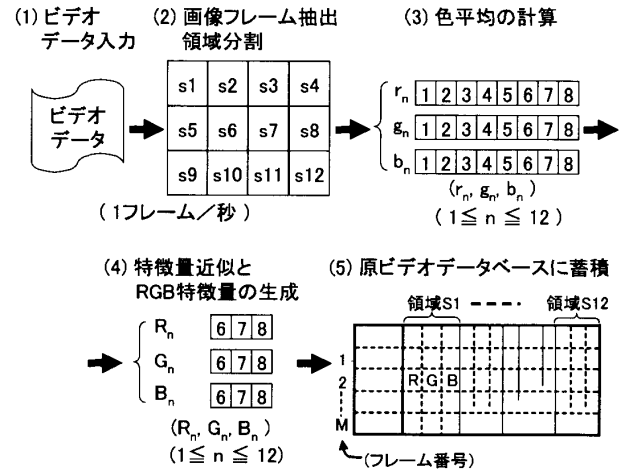


図 2　画像特徴量（RGB 特徴量）の抽出方法
Fig. 2　Image feature extraction process.

3.1　RGB 色空間に基づいた画像特徴量の抽出

ここでは，提案手法で用いた RGB 色空間に基づいた画像特徴量の抽出方法について説明する．文献 [4] は RGB 色空間に基づいた特徴量を用いたビデオ検索手法の研究で，検索時間を気にしなければ検出精度が良いことが報告されており，本研究では基本的にその手法と同様のアプローチを適用した．図 2 に適用した手法のアイデアを示す．

はじめに，入力した著作権保護の対象ビデオから 1 秒に 1 回画像フレームを抽出して，その画像フレームを 4×3 の領域 S_n ($1 \leq n \leq 12$) に分割する（図 2 (1)–(2)）．次に領域ごとに各 8 ビットの RGB 色データ (r_n, g_n, b_n) $(0 \leq r_n, g_n, b_n \leq 255)$ のそれぞれの平均値を計算する（図 2 (3)）．計算した各平均値データから上位 3 ビットのデータだけを取り出し 36 次元の特徴ベクトル (R_n, G_n, B_n) $(0 \leq R_n, G_n, B_n \leq 7)$ とする（図 2 (4)）．この 36 次元特徴ベクトル（以下 RGB 特徴量）を，検索に用いるため，取り出したフレームの順番で原ビデオデータベースに蓄積する（図 2 (5)）．

3.2　Tiny LSH によるハッシュ関数

画面サイズの変更等により元のビデオデータが改変されたビデオを検出するには類似した画像フレーム（すなわち類似した RGB 特徴量をもつ画像フレーム）を検索する必要がある．本研究では，このために LSH [7] と類似したアプローチを適用した．

LSH は類似した特徴量に対して同じ値を返すハッシュ関数を用いるものである．$\boldsymbol{a}$ を p-Stable 分布から取り出したランダムなベクトル，b をデータ範囲 $[0, w]$ におけるランダムな値とするとき，特徴ベクトル $\boldsymbol{q}$ に

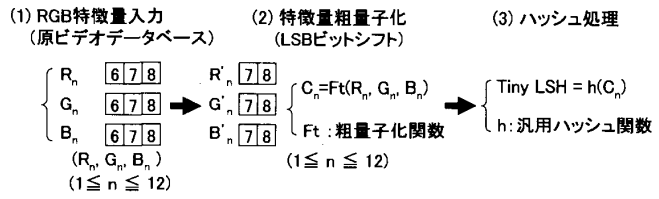


図 3　Tiny LSH の処理方法
Fig. 3　Process of the Tiny LSH.

対する LSH 関数 $h_{ab}(\boldsymbol{q})$ は以下の式となる [7].

$$h_{ab}(\boldsymbol{q}) = \frac{\boldsymbol{a} \cdot \boldsymbol{q} + b}{w} \tag{1}$$

文献 [7] では複数の LSH 関数 h_{ab} を用いることで多次元データ空間上の類似したデータを検索する手法が述べられている．

問題を RGB 特徴量に基づいた画像に限定すれば LSH の関数はより簡便に構成できる．提案手法は文献 [7] で提案されている従来の LSH とハッシュ関数の生成方法が異なるので，区別するために「Tiny LSH」と呼ぶことにする．

提案する Tiny LSH の処理方法を図 3 に示す．はじめに，図 2 で説明した RGB 特徴量 (R_n, G_n, B_n) $(1 \leq n \leq 12)$ $(0 \leq R_n, G_n, B_n \leq 7)$ を入力する（図 3 (1)）．次に，類似した画像に対しては同一の特徴量となるように，また暗い画像でもできるだけ顕著な近似的な画像特徴量を取り出して汎用ハッシュ関数の入力データ C_n となるように粗量子化関数 Ft により特徴量の粗量子化処理を行う（図 3 (2)）．具体的には特徴量のデータ値の大きさに応じて Least Significant Bit (LSB) を無視する処理を行う．

粗量子化関数 Ft のコードを図 4 に示す．画面が暗い画像フレームの場合には，特徴量の上位ビットは 0 になる傾向にあるため同じ特徴量の画像フレームが多く生じて検索正解率が低下する場合がある．粗量子化関数 Ft は，計算結果 C_n が 0 になる場合は，順次，ビットシフト量を変化させてできるだけ 0 にならない処理を行い汎用ハッシュ関数 h の入力データ C_n を生成する．

粗量子化関数 Ft は，各 3 bit，36 次元の RGB 特徴量を，各 3 bit，12 次元の特徴量 $C_n (1 \leq n \leq 12,\ \ 0 \leq C_n \leq 4)$ に変換する．これは元の RGB 特徴量を粗く量子化し，画面が類似したフレームの特徴量を同じにすることを意図しており，結果として画面が類似したフレームの Tiny LSH 値が同じになる．

以上のように，文献 [7] の LSH が p-Stable 分布を

```
粗量子化関数 : Ft(Rn, Gn, Bn)
  (1 ≤ n ≤ 12)
1:   Cn=(Rn >>1 + Gn >>1 + Bn >>1) >>1
2:   if (Cn == 0)
3:       Cn =(Rn >>1 + Gn >>1 + Bn >>1)
4:   if (Cn == 0)
5:       Cn = (Rn + Gn + Bn)
6:  Return Cn
```

図 4 粗量子化関数
Fig. 4 Correction function.

利用して確率的に類似データに同じハッシュ値を割り当てようとするのに対し，Tiny LSH は RGB 特徴量の性質を使って類似した画像フレームに対して同じハッシュ値を割り当てる．すなわち Tiny LSH は応用範囲を任意の多次元データベースから画像検索に絞ることで画像特徴量の近似処理と汎用ハッシュ関数を用いるだけで，類似検索機能を実現している．

3.3 Direct-mapped cache に基づいた高速検索

提案手法では，Tiny LSH の概念と文献 [12] で用いられた高速データ検出の手法を組み合わせ，処理を高速化した．図 5 はアイデアの中核となる Direct-mapped cache と原ビデオデータベースからなるデータ構造を示している．

Direct-mapped cache には原ビデオデータベースに記憶された個々のフレームと類似したフレームが検索対象ビデオに含まれていたときに，高速に検索するためのポインタを記憶する．このとき Tiny LSH で計算したハッシュ値を一つだけ用いるのでは図 1 に示したデータの特性からハッシュ値の衝突に起因して性能が低下するので複数種類の Tiny LSH 関数を用いて 1 フレーム当り複数のポインタを記憶する．複数のポインタを用いても極端に類似フレームの多いフレームへのポインタは記憶しきれない（後述するように上書き処理で消される）が，一つのビデオを構成する複数のフレームの中の幾つかは Direct-mapped cache 中にそのフレームへのポインタが残る．一つでもポインタが残れば後述の検索処理が対応できる．

図 5 の原ビデオデータベースは，図 2 (4) で生成したフレームごとに RGB 特徴量を蓄積する．蓄積された RGB 特徴量から Tiny LSH によって k 個の異なるハッシュ値を計算し，計算したハッシュ値を使って Direct-mapped cache 中に原ビデオデータベースにおける各フレームに対するポインタを蓄積する．

図 6 に，原ビデオデータにおけるポインタの蓄積処

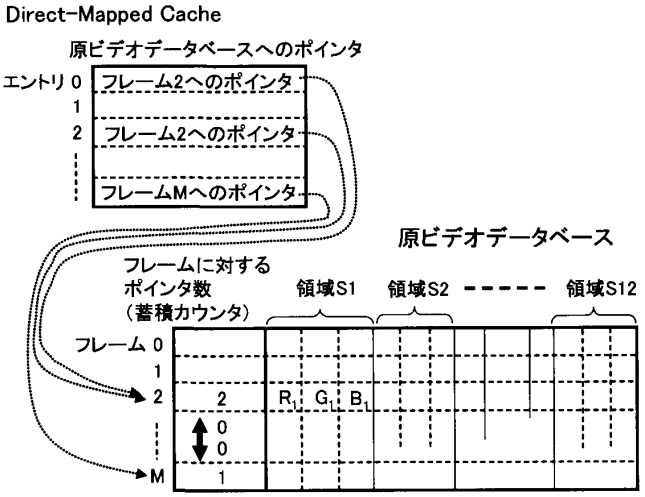


図 5 Tiny LSH に基づく Direct-mapped cache
Fig. 5 Direct-mapped cache with Tiny LSH.

理の方法を示す．はじめに，図 6 の 1 行目の for 処理でフレーム f に対して k 個の異なる関数 Tiny LSH_i によって k 個の異なるハッシュ値を生成する．次章の実験では図 3 (3) 汎用ハッシュ関数 h として SDBM ハッシュ関数 [13] を用いた．具体的には粗量子化関数 Ft の計算結果 $C_n(1 \leq n \leq 12)$ を長さ 12 の文字列としてまとめた後，その文字列を SDBM ハッシュ関数の入力データとしてハッシュ処理して，Tiny LSH_0 の値とした．また Tiny LSH_(i-1) $(1 \leq i)$ の値を SDBM ハッシュ関数の入力データとしてハッシュ処理した計算結果を Tiny LSH_i $(1 \leq i \leq k)$ の値とした．

次に（図 6 の 2 行目の for 処理から）生成した k 個のハッシュ値に対するポインタを Direct-mapped cache に蓄積する．9 行目の上書き処理により一つのビデオを構成する複数のフレームへのポインタ全てが上書きされてしまうことをなるべく防ぐために，ポインタの重要性の判定基準として以下の二つを考える．

（1） 原ビデオデータベース中，蓄積カウント値（=そのフレームへの Direct-mapped cache からのポインタ数）が連続して 0 になっている区間の前後のフレームへのポインタは重要

（2） Direct-mapped cache から参照するポインタが沢山ある原ビデオデータベース中のフレームへのポインタの重要性は低い．

(1) の判定基準は，ポインタがたくさんあるフレーム系列とそのフレーム系列へのポインタは，ポインタが少ないフレーム系列に比べ，元のビデオに関係するポインタが全て上書きで消される可能性が少ないと考えるためである．(2) は，あるフレームの被参照数が

```
蓄積処理 store(f)   f:フレーム
 1: for(i=0; i<k; i++)p[i]=Tiny LSH_i(f);
 2: for(i=0; i<k; i++){
 3:  if(p[i] ハッシュ値の衝突がない){
 4:   ポインタを蓄積;
 5:   (蓄積した f に対する蓄積カウンタ)++;
 6:  }
 7:  if(p[i] ハッシュ値が衝突){
 8:   if(重要性が低いポインタがある){
 9:     重要性が低いポインタを上書きして蓄積;
10:     (ポインタを上書きした f に対する蓄積カウンタ)--;
11:     (蓄積した f に対する蓄積カウンタ)++;
12:   }
13:  }
14: }
```

図 6　画像フレームにおける LSH データ蓄積処理
Fig. 6　Storing LSH data for the video frame.

```
検索処理 retrieve(v) (v: 検索目標ビデオ)
 1: foreach (q:フレーム位置 in {q1,q2, ... qL}){
 2:   // 第 1 から最終フレームまで均等間隔で L 箇所;
 3:   for ( t'= (q-m) ; t'<= (q+m) ; t'++) {
 4:     t=t' に対する原ビデオデータベース中のフレーム位置;
 5:     位置 t に対応したポインタが Direct-mapped
 6:             cache に記憶されているかチェック;
 7:     if(位置 t に対応するポインタが記憶されている){
 8:       t を基準に v と原ビデオデータベース中の
 9:             ビデオとのビデオ間距離を式 (4) で計算;
10:       最小距離 dv, 開始位置 t" を更新しメモリに記憶;
11:        (ここで t"=t-q)
12:     }
13:   }
14: }
15: return(メモリに記憶された最小距離 dv, 開始位置 t");
```

図 7　LSH データの検索処理
Fig. 7　Retrieval process of the LSH data.

多いときは，そのフレームを参照するポインタを一つ上書きしてもそのフレームに対するポインタは残ると考えるためである．

図 6 の 3 行目 if 文～6 行目はハッシュ値の衝突がない場合の処理で，単にポインタを蓄積し，蓄積したフレーム（すなわちポインタが参照しているフレーム）に対する蓄積カウンタを増やす．各フレームが何個のポインタで参照されているかは，この蓄積カウンタによって判定することができる．図 6 の 7 行目 if 文～13 行目はハッシュ値が衝突する場合の処理である．更に，8 行目の if 文で重要性が低いポインタがある場合にはそのポインタを上書きして，上書きしたポインタが参照する蓄積カウンタを減らし，書き込んだポインタが指すフレームの蓄積カウンタを増やしている．

検索処理では検索目標ビデオ v の各フレーム位置 q に対して，位置 q のフレームの RGB 特徴量を検索キーとして，Direct-mapped cache に記憶したポインタを用いて，原ビデオデータベースに蓄されている類似フレームの位置 t を検索した後，原ビデオデータベースに蓄積されている前後のフレームシーケンスで構成されるビデオと検索目標ビデオのビデオ間距離（後述の RGB 特徴量に基くビデオ間距離 d_v）を計算し，最も距離の近いビデオを検索結果のビデオとする（図 7）．ここで，類似フレームの検索結果（すなわち位置 t）は q 及び後述の m に対応して複数あるので，最小距離を与えるビデオの開始位置 t" を一つ検索結果とする．

具体的には図 7 の 1 行目の foreach と 3 行目の for 処理により q と m の値を変えることで原ビデオデータベース内の位置 t を変えながら，検索目標ビデオと原ビデオデータベースに記憶されたビデオの距離 d_v を計算し（図 7 の 9 行目），最小距離を与える位置 t（正確には q, m によるずれを補正した開始位置 t"）と，そのときの距離 d_v を検索出力とする．

検索目標ビデオ v の全フレーム中 L 箇所の各位置 q ごとに処理を行う理由は，全てのフレームにおいて距離計算を行わなくても検索精度が確保できるので，検索時間を短縮するためである．後述の実験では，検索目標ビデオ長 $d \geq 60$ の場合は，L=60 とし，$d < 60$（秒）の場合は，L=d とした．

3 行目の for 処理は，q を中心に前後 m の範囲（すなわち $(q-m)$～$(q+m)$ の間）で位置をずらして，検索目標ビデオと原ビデオデータベース中のビデオとが最も一致する位置を検索するための距離計算を行うことを意図する．ここで位置 t' は最小のビデオ間距離を検索するための基準となる検索目標ビデオ v 内のフレーム位置を意味し，フレームのハッシュ値に基づいて図 5 の Direct-mapped cache を検索することで，対応する原ビデオデータベース内のフレーム（すなわち同じハッシュ値をもつフレーム）位置 t を検索する．検索処理の手続き全体では，検索目標ビデオに含まれる画像フレームの第 1 から最終フレームまで均等間隔で L 箇所の各位置 q ごとに検索処理を行う．

位置 q を中心に前後 m の範囲で検索する理由は，前述の上書き処理への対応である．すなわち，連続し

た画像フレームは，類似した特徴量をもつ傾向があるので，Direct-mapped cache 上の消されたポインタ付近に隣接フレームへのポインタが残っている可能性が高くなる．そのため，近傍を探索することで，上書きされたポインタの指すフレームに近いフレームを見つけられる場合がある．図 7 の 5〜6 行目で，検索位置にポインタが記憶されているかチェックし，7 行目の if 文でポインタが記憶されている場合に，8〜10 行目で画像フレームのシーケンスが一致するか RGB 特徴量の距離計算を行い，最小距離 d_v と開始位置 t”(=t−q) を逐次記憶していく．$(q-m)$〜$(q+m)$ に対応する範囲で検索が終わったら，最終的に 15 行目でメモリに記憶されている最小距離 d_v と原ビデオデータベース内の開始位置 t” を検索出力とする．

検索目標ビデオと原ビデオデータベース内に記憶されたビデオとの距離は次式のように計算する．

(領域間距離)

$$\begin{aligned} d_{rSn} = \; & (\; (R_{\text{target}} - R_{\text{stored}})^2 \\ & + (G_{\text{target}} - G_{\text{stored}})^2 \\ & + (B_{\text{target}} - B_{\text{stored}})^2 \;)^{1/2} \end{aligned} \tag{2}$$

$(1 \leq n \leq 12)$

target：検索目標ビデオ

stored：原データベース内のビデオ

(画像フレーム間距離)
$$d_f = \sum_{k=1}^{12} d_{rSk} \tag{3}$$

(ビデオ間距離)
$$d_v = \sum_{f=p}^{p+d} d_f \tag{4}$$

すなわち，RGB 特徴量を用いたユークリッド距離を各領域 S_n $(1 \leq n \leq 12)$ 間の距離 d_{rSn} とし，各領域間の距離 d_{rSn} の和を画像フレーム間距離 d_f とする．この画像フレーム間距離を検索位置 p からビデオの長さ d 分だけ加算することでビデオ間距離 d_v とする．

直感的には上記処理は抜けの多い処理にも思える．例えば，提案手法の RGB 特徴量は画像フレームの特徴量を少ないビット数で保持している（1 フレーム当りたかだか 12×3×3 ビット）．Tiny LSH 関数も更に上位ビットの情報しか処理していない．すなわち提案手法はビデオの粗い特徴を扱っているだけである．また，量子化の影響で Tiny LSH 関数値が類似した画像フレームに対して同じ値とならない場合もある．蓄積処理もまた完全ではなく，ハッシュ値の衝突が多い画像フレームに対して Direct-mapped cache の中にポインタを蓄積していないときがある．Direct-mapped cache におけるポインタの欠落は対応する画像フレームの検索を失敗させ，その結果，図 7 の検索処理全体が失敗しビデオ検出精度が低下することになる．

このように，提案手法には欠点が存在するものの，次の章で説明する実験結果では，実際の動作にこのような欠点は現れていない．検索時間が，ハッシュ値の衝突に基づく影響を受けないことと，原ビデオデータベースにおけるデータ蓄積量に依存しない点が，提案手法の特徴である．

4. 実　　験

提案手法の検索性能を評価するために，テストデータとして実際のビデオコンテンツを用いて実験を行った．この章では実験結果について説明する．

4.1 テストデータセット

表 1 にテストデータの概要を示す．実験のデータベース用テストデータとして，The Open Video Project の Web サイト [14] から 109.5 時間分のビデオコンテンツをダウンロードして用いた．具体的には，The Open Video Project の Web サイトにおいて，検索条件として，(All field, Any Genre, More than 10 minutes, sound, color, MPEG2) で検索したビデオでダウンロード可能な総時間 109.5 時間分，380 本のビデオコンテンツを用いた．ダウンロードしたビデオから，表 1 のような 3 種類のテストデータ（Original ビデオ，Re-scaled ビデオ，Cropped ビデオ）を用意した．Re-scaled ビデオは，軽微な変更のあるビデオとして，Cropped ビデオは変更内容を更に大きくし条件を厳しくした実験をするために用意した．

これら 3 種類のテストデータは，表 1 に示す処理

表 1　テストデータ
Table 1　Test data.

Original ビデオ	720×480 画素，6 Mbit/s, 30 fps, MPEG2 ビデオ（ダウンロードしたビデオを，画像サイズ，記録ビットレートを正規化処理）
Re-scaled ビデオ	360×240 画素，1 Mbit/s, 30 fps, MPEG2 ビデオ（画像サイズを縮小，記録ビットレートを下げて再エンコード）
Cropped ビデオ	360×240 画素，320 kbit/s, 24 fps, MPEG2 ビデオ（Original ビデオの画面中心 576×384 画素を抜き出し，更に画像サイズを縮小し記録．ビットレートとフレームレートを下げて再エンコード）

方法により生成した．はじめにダウンロードしたビデオは画像サイズ，記録ビットレートが統一されていないので，画像サイズ 720×480 画素，記録ビットレート 6 Mbit/s，フレームレート 30 fps の MPEG2 ビデオに統一するように正規化処理を行い，Original ビデオとした．Re-scaled ビデオは，Original ビデオから画像サイズを 360×240 画素に変更した後，記録ビットレートを 1 Mbit/s，フレームレート 30 fps の MPEG2 ビデオに変更したものである．Cropped ビデオは，Original ビデオの上下 10%（48 画素）ずつ，左右 10%（72 画素）ずつを切り取って画面中心の画像サイズ 576×384 画素を取り出した後，画像サイズ 360×240 画素にスケーリングし，記録ビットレート 320 kbit/s，フレームレート 24 fps の MPEG2 で再エンコードしたものである．Cropped ビデオと Re-scaled ビデオでは，Cropped ビデオの方が，Original ビデオからの画像の欠落の他に，記録ビットレートとフレームレートが低く，画質が低下している．

4.2 検索基本性能

最初の実験では，Original ビデオを原ビデオデータベースとして用いた．また，Original ビデオ，Re-scaled ビデオ，Cropped ビデオ，それぞれから長さを変えて（30 秒～600 秒）ビデオを取り出し検索目標ビデオとした．実験では原ビデオデータベース（Original ビデオ）中，検索目標ビデオに最も類似したビデオの位置を提案手法によって検出できるか調べ，その正解率 P_{det} を求めた．検索目標ビデオの総数 N_{all} は検索目標ビデオの長さ d に応じて，Original ビデオ長 (109.5(時間)×60(分)×60(秒))/検索目標ビデオ長 d(秒) となる．

検索目標ビデオが原ビデオデータベースから複製されたかの判定は，検索目標ビデオを順番に検索して行き，図 7 に基づいて検索された位置 t が対応する原ビデオデータベースの位置内にあるとき，正しく検索できたと判定した．正しく検索できた検索目標ビデオの数を N_{ok} とすると，検索の正解率 P_{det} (%) を下式のように定義した．

$$P_{det} = 100 \times \frac{N_{ok}}{N_{all}} \qquad (5)$$

3.3 で説明した設定すべきパラメータ，k（設定するハッシュ関数の数），m（ポインタ探索範囲）は予備実験の検討に基づいてそれぞれ 8，4 とした．また，図 5 の Direct-mapped cache エントリ数とビデオデータベースエントリ数は両方ともに原ビデオデータベース

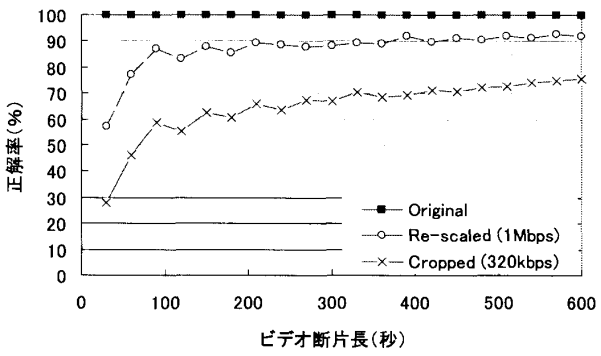


図 8 検索正解率
Fig. 8 Result of the experiment.

中の画像フレーム数 (109.5(時間)×60(分)×60(秒)) に設定した

図 8 に Original ビデオ，Re-scaled ビデオ及び Cropped ビデオに対しての検索結果を示す．データから分かるように Original ビデオから検索目標ビデオを取り出した場合全てのビデオ長で 100%の正解率であった．Re-scaled ビデオと Cropped ビデオでは検索目標ビデオが長いほど検出正解率は良くなっている．

Re-scaled ビデオでは，300 秒以上のビデオに対して，提案手法は 88%以上の正解率で検索ができている．ビデオが短くなると長い場合よりも検索正解率が低下するが，30 秒のビデオに対して 57%，60 秒の長さでは 77%の正しさで検出できている．

提案手法は 1 フレームから抽出する情報は小さいが，式 (4) の計算に基づいて一続きとなった複数フレームの情報を集めることで検索正解率が上がっている．この結果は文献 [4] の結果を追検証した結果となっている．

Cropped ビデオの場合は Re-scaled ビデオに比べて検出は難しく，同じような検出性能には達していないが，300 秒のビデオに対して 67%の正しさで一致する位置を検出できた．

図 9 は，各長さの検索目標ビデオを処理する場合の平均検索時間を示している(注2)．図に示したように提案手法は，Original ビデオでは全てのビデオ長に対してほぼ 0.01 秒以下（毎分 6000 本の検索が可能）で検索ができている．また，Original ビデオに比べて Re-scaled/Cropped ビデオは長い平均検索時間がかかっている．実験で使用したビデオ間距離計算用のプログラムコードは，それまで見つかった最小のビデオ間距離を記憶しておき，結果がそれより長くなった段

(注2)：CPU: Pentium4 3 GHz Memory:2 GB OS:Linux を使用．

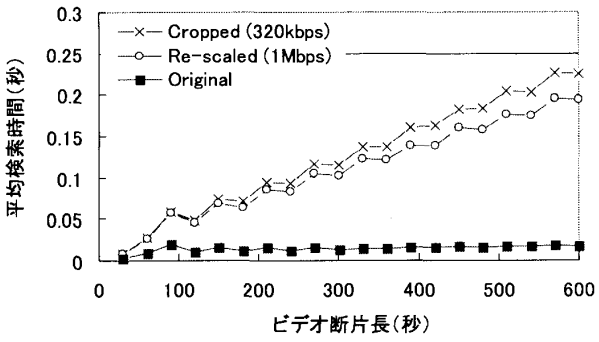


図 9 平均検索時間特性
Fig. 9 Average retrieval time.

階で以降のフレーム間距離の計算をスキップ（すなわち式 (4) $\sum$ の計算中に計算を中断）し，時間を短縮するように実装してある．Original ビデオから検索目標ビデオを取り出した実験では検出性能が良いことから最小のビデオ距離が早目に見つかり，スキップされた処理が増え検索時間が短くなったと考えられる．

一方，Re-scaled ビデオと Cropped ビデオでは，最小のビデオ距離が見つかった後の距離計算のスキップ回数が Original ビデオに比べて少ないので検索時間は長くかかっている．Re-scaled ビデオでは，**1.** で述べたビデオサービスにおけるビデオの平均の長さである 300 秒では平均検索時間が 0.1 秒（毎分 600 本の検索が可能）であった．

4.3 適合率と再現率

前節で説明した実験では，検索目標ビデオは常に，原ビデオデータベースに記憶されている．実際の状態では，そのようなことは少なく，原ビデオデータベースに記憶された最も類似したビデオから検索目標ビデオが複製されたかを判断しなければならない．ビデオの改変やビデオの長さ d によってビデオ間距離は変化するので，次の実験ではビデオが複製か否かの判定しきい値 d_{th} を変化させて提案手法の性能を検証した．

実験では検索目標ビデオと対応する原ビデオデータベース内のビデオの平均距離を長さ 300 秒の検索目標ビデオ 100 個を使って計測し，その平均 d_{av} を判定しきい値 d_{th} の基準とした．残った検索目標ビデオで判定しきい値を変化させ，適合率と再現率を検出した．すなわち，提案手法が元のビデオの位置を正しく検索できたビデオ（総数 N_{all}）のうちビデオ間距離 d_v が d_{th} 以下のビデオ数を N_{ok}，また，ビデオ間距離 d_v が d_{th} 以下のビデオ総数を N_p とし，

$$\text{適合率} = 100 \times \frac{N_{ok}}{N_p} \tag{6}$$

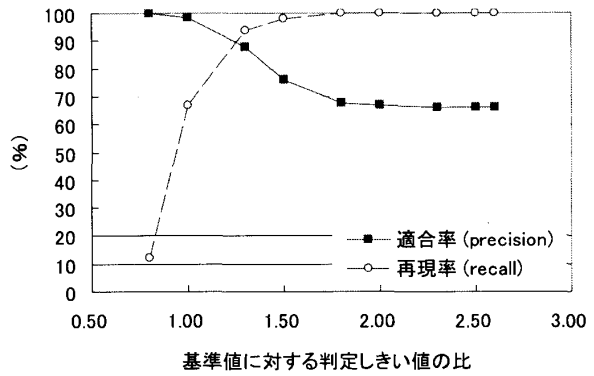


図 10 Cropped ビデオの適合率と再現率の特性
Fig. 10 Precision and recall on Cropped videos.

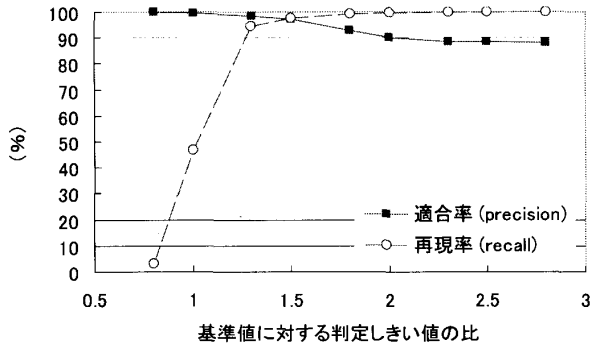


図 11 Re-scaled ビデオの適合率と再現率の特性
Fig. 11 Precision and recall on Re-scaled videos.

$$\text{再現率} = 100 \times \frac{N_{ok}}{N_{all}} \tag{7}$$

として，適合率 (%) と再現率 (%) を計測した．

Original ビデオの検索の場合は，適合率と再現率はすべて判定しきい値の範囲で 100%であった．図 10 と図 11 に，Re-scaled ビデオと Cropped ビデオの結果を示す．図において，横軸は上記平均距離 d_{av} と使用した判定しきい値 d_{th} の比を示す．図に示すように，設定したしきい値の割合が増える（すなわち，許容するビデオ間距離が長くなる）に従って再現率は増加し適合率は低下する．適合率と再現率が等しいときは Cropped ビデオに対しては 90%，Re-scaled ビデオに対しては 98%である．

Re-scaled ビデオだけを検索する場合には，判定しきい値を平均の約 1.5 倍に設定することが実用的と思われ，その場合，適合率と再現率は等しくなる．Cropped ビデオに対しては，判定しきい値を平均の約 1.3 倍に設定することが実用的と思われ，その場合，適合率と再現率は等しくなる．Re-scaled ビデオと Cropped ビデオの検索の場合には，判定しきい値の設定には幾つかのトレードオフが存在し，再現率と適合率に対する

最適なしきい値の設定は今後の課題である．

5. 考　　察

5.1 提案手法のメモリ必要量

4. の実験結果から，ビデオ画像の改変が大きくなければ提案手法は複製ビデオの検出を十分な精度で行えることが分かった．ここでは，提案手法を実際の環境下で使う場合の課題となる必要なメモリ容量について考察する．

表 2 は提案手法に必要なメモリ容量の概要である．メモリ容量の計算では，RGB 特徴量を蓄積する図 5 の原ビデオデータベースはビットアライメントを考慮し 4 bit 単位で実装するものとして計算した．32 bit ポインタを適用すると仮定すれば，Direct-mapped cache 構造は 4 G フレーム（ビデオ 136 年分）を保持することができる．各画像フレームのデータサイズは 24 Byte なので，全てのデータを蓄積するには 96 GB が必要になるが，最近の市販されているサーバ用 PC が高性能なことを考えると実用的に問題のない値である．仮に 8 GB メモリを用いる場合には，提案手法は 358 M フレーム（すなわち 11.2 年=99.6 K 時間長のビデオ）を扱うことができる．したがって，汎用 PC を用いた場合でも 11 年分のデータを扱うことができる．

表 2　メモリ必要量の見積り
Table 2　Memory requirements for proposed method.

64 bit システム (32 bit ポインタ)		
4 G フレーム	=	60(秒)×60(分)×24(時間) ×365(日)×136(年)
	→	1.19 M 時間
96 GByte	=	(18 Byte(=(4×3)(領域) ×4 bit×3 RGB/8 bit) +4 Byte(フレーム ID) +2 Byte(蓄積カウンタ)) × 4 G(フレーム)
8 G PC システム		
8 GByte	=	(18 Byte(=(4×3)(領域) ×4 bit×3 RGB/8 bit) +4 Byte(フレーム ID) +2 Byte(蓄積カウンタ)) × 358 M(フレーム)
	→	11.2 年，99.6 K 時間

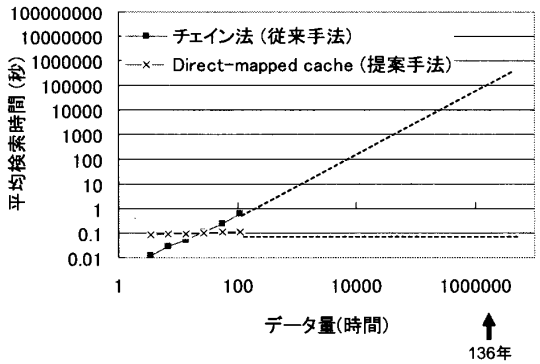


図 12　提案手法と従来手法の平均検索時間の比較（Re-scaled ビデオ断片長 300 秒）
Fig. 12　Average retrieval time of proposed method and conventional method.

5.2 提案手法と従来手法のビデオ検索性能比較

本論文で提案した Tiny LSH は従来の LSH ハッシュ関数と生成方法は異なるが類似検索を実現するハッシュ関数としての挙動は同じである．したがって，ハッシュ関数を用いた従来手法と提案手法である Direct-mapped cache の比較で主たるポイントは検索時間である．

そこでハッシュ処理を使う代表的な従来手法であるチェイン法 [15] に Tiny LSH 関数を適用して作成したシステムと提案手法による（Tiny LSH 関数と図 5 に示した高速検索手法を併用した）システムとの性能比較を行った．チェイン法はハッシュ値の衝突が起きた場合に，同一フレームに対するポインタを鎖（チェイン）状にリンクして接続する手法である．そのため，ハッシュ値の衝突が多い場合にはリンクが長くなり，その部分では線形探索になるため検索時間が大きくなる問題がある．

ビデオのデータ量が変わると類似フレームの量が変わりハッシュ値の衝突頻度が変わる．そのようなことから，検索に用いるビデオのデータ量を 3.4 時間～109.5 時間の間で変化させて実験を行い，提案手法と従来手法の性能比較を行った(注3)．

図 12 は，ビデオ断片長 d=300 秒の場合の平均検索時間を示すもので，実験結果に基づいてビデオデータ量を 136 年まで増加させた場合の推定データを点線で示す．図 12 から提案手法は，データ量が増加しても処理 1 回当りの平均検索時間はほぼ一定であるが，従来手法のチェイン法ではデータの増加とともに平均検索時間が増加し 136 年分のデータ量では大よそ 10^5 秒（≃27.7 時間）になると推定できる．これは，ハッシュ値の衝突する確率が高くなりハッシュテーブルのリンクが長くなって線形探索の時間が大幅に増加するためと推定できる．

ハッシュテーブルのリンクが長くなり線形探索の時間が大幅に増加することは図 1 に示すビデオデータの

(注3)：**4.2** の実験と同じく，図 5 の Direct-mapped cache エントリ数とビデオデータベースフレーム数は両方ともに原ビデオ (Original) (3.4～109.5 時間) の画像フレーム数に合わせて設定した

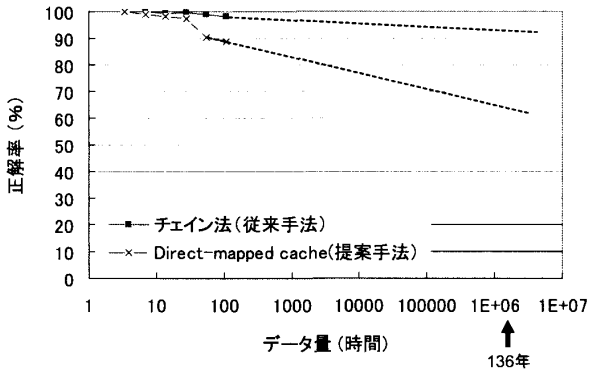


図 13　提案手法と従来手法の正解率の比較（Re-scaled ビデオ断片長 300 秒）

Fig. 13　Retrieval accuracy of proposed method and conventional method.

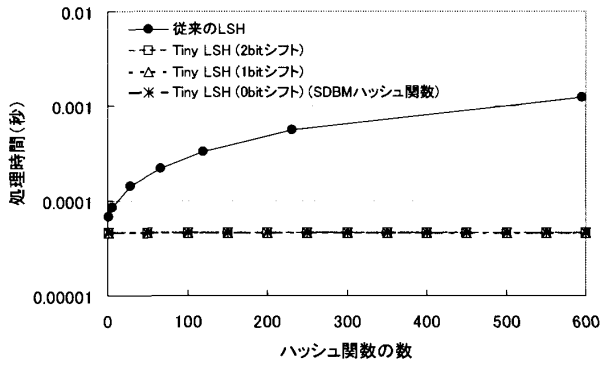


図 14　提案手法と従来手法のハッシュ値一致検索の処理時間

Fig. 14　Processing time of matching hash value.

特性から，文献 [7] の LSH 関数等を使っても避けられないことには注意を要する．提案手法はビデオ検索において類似フレームが多いことから生じる本質的な問題への一つの対応案となっている．

図 13 は，ビデオ断片長 d=300 秒の場合の Re-scaled ビデオに対する検索正解率を示すもので，この実験結果に基づいてビデオデータ量を 136 年まで増加させた場合の回帰分析による推定値を点線で示す．このデータから，136 年分のデータ量では，チェイン法の場合は約 94%，提案手法の場合は約 64%になると推定できる．チェイン法では，ハッシュ値の衝突が起きてもハッシュテーブルのリンク長が増加することでデータは残るので検索できるが，提案手法では **3.3** で説明したポインタの上書きによって残るポインタの確率が低下するために正解率が低下すると考えられる．チェイン法は正解率の大きな低下はないが図 12 の結果から平均検索時間は実用レベルの性能ではないことが分かる．

5.3　提案手法と従来手法の類似フレーム検索性能比較

5.2 ではビデオ全体を対象とした提案手法と従来手法の検索性能の比較を行った．本節では，類似フレームの検索性能について，Tiny LSH と従来の LSH を比較する．

Tiny LSH としては，**3.** 記載の実装（粗量子化関数 Ft のビットシフト量が 1）と，RGB 値を合計後のビットシフト量を 2 と大きくしたもの，RGB 値の各ビットシフト量と合計後のビットシフト量をすべて 0 にしたもの（この場合，図 3 (2) の粗量子化関数 Ft の処理をしない場合と等価で，通常の SDBM ハッシュ関数と同じになる），3 種類を用意した(注4)．

従来の LSH としては，文献 [7] に基づいて実装したコードを用いた(注5)．ここで，従来の LSH は，利用するハッシュ関数の数により検索性能が変わるので 1〜600 の範囲で利用するハッシュ関数の数を変えて実験した．

図 14 に処理時間（一つのハッシュ値一致検索の時間＋データ蓄積時間）を示す(注6)．従来の LSH は Tiny LSH よりも処理時間が長く，ハッシュ関数の数が増加すると処理時間も増加する（ビットシフト量を変えた Tiny LSH は，いずれも測定誤差の範囲で同じ処理時間であった）．

5.4　正解データが少ない場合の検索特性

4.2 において，最小のビデオ間距離が早めに見つかる場合に計算をスキップして検索時間の短縮化を行っていることを述べた．検索目標のデータに対してビデオデータベースに正解となるデータ数が少ない（不正解データ数が多い）場合は，このスキップ処理が行われる回数が少ないために検索時間が増加する．

ここでは，ビデオデータベースに蓄積する Original ビデオの全 RGB 特徴量の中で，検索目標ビデオの RGB 特徴量とは対応しないデータ数の割合を変化させることで，不正解データ数を変化させるものとし，改変しないビデオの検索（Original ビデオと Original ビデオの検索）において，一つのビデオ断片を検索処理する場合の検索時間の変化を計測する実験を行った．

(注4)：これらは，いずれも無視する Least Significant Bit の数が違うだけで，問題を RGB 特徴量に基づいた画像に限定し量子化の程度を変えた Tiny LSH の異なる実装と考えている．また，正確には Tiny LSH_0 の値のハッシュ値の一致を考える．

(注5)：LSH の実装コードを提供して頂いた Princeton 大学 Alexandr Andoni 博士に感謝します．

(注6)：Tiny LSH と SDBM ではハッシュ関数の数は一つであるが，データを見やすくするために 50 ごとに各プロット点を表示した．

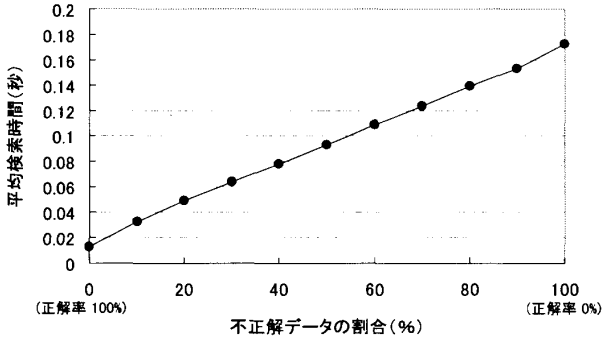


図 15　正解データが少ない場合の平均検索時間
Fig. 15　Average retrieval time in case of existing no-matching data.

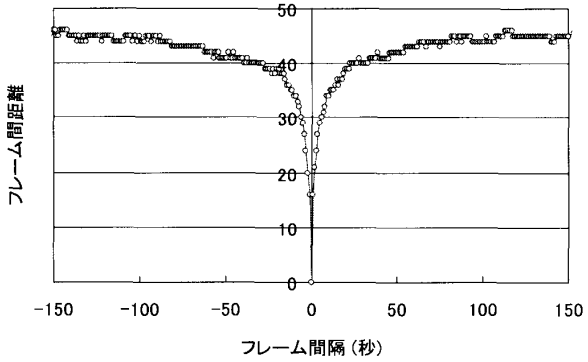


図 16　RGB 特徴量に基づくフレーム間距離特性
Fig. 16　Frame distance based of the RGB features.

不正解データは RGB 特徴量のデータ範囲で人工的に乱数を発生させ擬似的な RGB 特徴量を生成し，元のデータと入れ換えることで，擬似的な RGB 特徴量（すなわち不正解データ）の割合を 0%（検索目標ビデオに対応するデータが全て存在する場合（正解率 100%））～100%（検索目標ビデオに対応するデータが存在しない場合（正解率 0%））まで変化させ図 9 に結果を示した実験のうちビデオ断片長 300 秒の Original ビデオデータを対象とした実験を再度実施した．

図 15 は，ビデオ断片長 300 秒の Original ビデオデータ（全データ 109.5 時間分）に対して，不正解データの割合を変えて，ビデオ断片一つを検索する場合の平均検索時間を計測した実験結果を示す．この実験結果から，平均検索時間は不正解データの割合にほぼ比例して増加するが，全てが不正解データという極端な場合でも実験に用いた PC 環境では 0.18 秒以下で検索処理が終了している．

5.5　画像フレーム間の類似特性に関する考察

Tiny LSH は類似した RGB 特徴量に対して同じハッシュ値となるので，類似する近接の画像フレームに対しても同じ Tiny LSH 値となる可能性がある．そのため，Direct-mapped cache に記憶したポインタを使い対応するフレーム位置を検索するときに，ポインタで指示されたフレームの前後のフレームも対応する可能性を考慮する（図 7 の 3 行目）ことで，結果として正しいフレームの対応関係を検出でき，検索正解率が改善できる．

上記考察を確認するため，Original ビデオにおける画像フレーム位置を基準にして，その基準フレームと前後の画像フレームについて RGB 特徴量に基づいた画像間距離を計測した．図 16 はその計測結果であり，300 秒の区間ごとにその区間中心を基準に前後 150 秒の範囲で画像間距離を計測し，その平均値をプロットしている．この計測結果からフレーム間隔が離れるほど画像距離は大きくなり約 10 フレーム以上離れると距離は大きく変化しなくなることが分かる（すなわち，前後 10 秒は類似した画像が継続している場合が多いことを示している）．

6.　む　す　び

本論文では，所望のビデオ断片を大容量のビデオデータベースの中から実用的な精度と速度で検索可能な Tiny LSH に基づいた複製ビデオの検索手法を提案した．予備実験において従来手法では WWW 上にある膨大なビデオコンテンツを対象とした検索ではハッシュ値の衝突に基づいた問題が存在すること明らかにした上で，提案手法では似た入力値には同じハッシュ値を返すという LSH と同様の考え方で用いたハッシュ関数と，Direct-mapped cache を利用する高速データ検索手法を組み合わせることにより，そのような問題を改善して精度の良い複製ビデオの検出を試みた．実際のビデオコンテンツを用いた検索実験で以下のことを明らかにした．

- 提案手法は 96 GByte のメモリを用いて 136 年分の長さのビデオを扱うことができる．
- 平均検索時間は，改変のないビデオの場合は，実験した全てのビデオ断片長に対して平均 0.01 秒以下（毎分 6000 本の検索が可能）であった．画像サイズと記録ビットレートを改変したビデオの場合は，ビデオ断片長が 300 秒のデータに対して平均 0.1 秒以下（毎分 600 本の検索が可能）であった．
- 検索正解率はビデオの改変に依存し，改変のない場合は 100%，画像サイズと記録ビットレートを変え画質を低下させた場合は 88%，画像の一部削除など

更に画質を低下させた場合では67%であった．

ビデオの検索性能はビデオの内容によっても影響されると考えられる．今回の実験よりも大規模なビデオデータや，スクリーン画面をビデオカメラで撮影したビデオなど改変の種類を変えたビデオデータに対して実験することと，提案手法に幾つかあるパラメータの最適な設定法の検討が今後の課題である．

文　献

[1] A.Z. Tirkel, G.A. Rankin, R.M. van Schyndel, W.J. Ho, N.R.A. Mee, and C.F. Osborne, "Electronic watermark," Digital Image Computing, Technology and Applications (DICTA'93), pp.666–673, Macquarie University, Sidney, 1993.

[2] A. Joly, O. Buisson, and C. Frelicot, "Statistical similarity search applied to content-based video copy detection," 21st International Conference on Data Engineering Workshops (ICDEW'05), p.1285, 2005.

[3] "YouTube 日本版公式ブログ：「YouTube に動画が投稿されるペースは？」," 入手先 <http://youtubejpblog.blogspot.com/2009/05/youtube.html>（参照 2009-05-23）.

[4] 長坂晃朗，宮武孝文，"時系列フレーム特徴の圧縮符号化に基づく映像シーンの高速分類手法," 信学論（D-II），vol.J81-D-II, no.8, pp.1831–1837, Aug. 1998.

[5] J.Z. Wang, G. Wiederhold, O. Firschein, and S.X. Wei, "Content-based image indexing and searching using daubechies' wavelet," Int. J. Digital Libraries, vol.1, no.4, pp.311–328, 1997.

[6] D.G. Lowe, "Distinctive image features from scale-invariant keypoints," Int. J. Comput. Vis., vol.60, no.2, pp.91–110, 2004.

[7] M. Datar, P. Indyk, N. Immorlica, and V. Mirrokni, Locality-sensitive hashing using stable distributions, MIT Press, 2006.

[8] 木村昭悟，柏野邦夫，黒住隆行，村瀬　洋，"グローバルな枝刈りを導入した音や映像の高速探索," 信学論（D-II），vol.J85-D-II, no.10, pp.1552–1562, Oct. 2002.

[9] K.M. Pua, J.M. Gauch, S.E. Gauch, and J.Z. Miadowicz, "Real time repeated video sequence identification," Comput. Vis. Image Underst., vol.93, no.3, pp.310–327, 2004.

[10] T. Liu, A. Moore, A. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," Proc. Neural Information Processing Systems (NIPS), Vancouver, BC, Canada, 2004.

[11] K. Yoshida and N. Murabayashi, "Tiny LSH for content-based copied video detection," Proc. International Symposium on Applications and the Internet 2008 (SAINT 2008), pp.89–95, 2008.

[12] K. Yoshida, F. Adachi, T. Washio, H. Motoda, T. Homma, A. Nakashima, H. Fujikawa and K. Yamazaki, "Density-based spam detector," KDD2004, pp.486–493, 2004.

[13] "Sdbm," 入手先 <http://search.cpan.org/src/NWCLARK/perl-5.8.8/ext/SDBM_File/sdb%m/README>（参照 2010-01-25）.

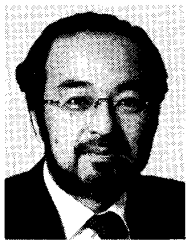
[14] "The Open Video Project," 入手先 <http://www.open-video.org>（参照 2009-10-01）.

[15] D.E. Knuth, The Art of Computer Programming: Sorting and Searching, vol.3, Addison-Wesley, 1972.

（平成 23 年 3 月 7 日受付）

村林　昇

1982 東京都立大・工・電気卒，1984 筑波大学大学院理工学研究科修士課程了．同年ソニー（株）入社．2010 年 3 月筑波大学大学院ビジネス科学研究科博士後期課程了，博士（システムズ・マネジメント）．



吉田　健一　（正員）

1980 東工大・理・情報科学卒，同年日立製作所入社．1992 年 9 月博士（工学，大阪大学）．2002 より筑波大学大学院ビジネス科学研究科教授．インターネット上の各種データを，機械学習の手法を使って解析する研究に従事．