ビット幅の狭いスケッチを用いた高速類似検索に関する研究

樋口 直哉[1,a)]　今村 安伸[1,b)]　久保山 哲二[2,c)]　平田 耕一[1,d)]　篠原 武[1,e)]

概要：局所性鋭敏ハッシュ (LSH) の一種であるスケッチを用いた最近傍検索について議論する．スケッチを用いる最近傍検索は 2 段階で行う．第 1 段階では，質問とのスケッチ間の距離が近い K 個の解候補を選択する．ただし，$K \geq 1$ である．第 2 段階では，K 個の解候補に対して実距離計算を行うことで最近傍解を選択する．従来，高い検索精度を保証するためには 32 ビット以上のスケッチを用いる必要があると思われていた．本研究では，スケッチのビット数を 16 に減らすことにより，バケット法にデータ管理による高速化を可能とし，第 1 段階の検索コストをほとんど無視できる手法を提案する．16 ビットスケッチを用いる検索は，精度を維持するためには，32 ビットスケッチをもちいる場合より大きな候補数 K を必要とする．データオブジェクトをスケッチの値によってソートしておくことで，第 2 段階の検索におけるメモリ局所性を向上することで候補数 K の増加による速度低下を低減できる．提案手法を用いると，検索精度を維持しつつ 20 倍程度の高速化が実現できる．

Fast Similarity Search using Narrow Sketches

NAOYA HIGUCHI[1,a)]　YASUNOBU IMAMURA[1,b)]　TETSUJI KUBOYAMA[2,c)]　KOUICHI HIRATA[1,d)]
TAKESHI SHINOHARA[1,e)]

1. はじめに

1.1 スケッチと類似検索

多次元空間における効率的**類似検索**を実現するために，**スケッチ** [1], [2], [3], [4], [5] を用いる検索手法が開発されてきた．スケッチは，多次元データを表したコンパクトなビット列であり，データ間の類似性をある程度保持できる局所性鋭敏ハッシュ (LSH) の一種である．**球面分割**（BP）は，データに対して，球内であればビット 0，そうでなければビット 1 を割り当ててスケッチを作成する方法である．BP は，vantage point tree [6] でも用いられている．スケッチを用いた類似検索は 2 段階からなる．第 1 段階では，スケッチ間の距離を用いて候補を選択する．第 2 段階では，元の空間内の距離を使用して，質問と候補を比較することにより，最近傍を選択する．通常，スケッチ間の距離にはハミング距離が用いられている．スケッチを用いた検索では，スケッチ間の距離がオブジェクト間の距離を完全に反映することができないため，階層的空間索引 R-tree [7] や M-tree [8] を用いた検索と異なり，正確に最近傍解を求めることができない．階層的空間索引法と遜色ない速度である程度の精度を保証するためには，スケッチのビット数は，少なくとも 32 ビットや 64 ビット必要であると考えられてきた．

1.2 ビット幅の狭いスケッチの提案

本論文では，より幅の狭い 16 ビットのスケッチを用いる手法を提案する．データベースの大きさは，数百万であると仮定する．32 ビットのパターンの個数は 2^{32}=約 40 億個あり，データベースサイズがそれを超えるほど巨大でない限り，空のバケットが多過ぎるので，バケット法は適さない．32 ビットより幅が広いスケッチを用いる場合は，第 1 段階検索は，データのすべてのスケッチをそのまま用意

[1] 九州工業大学
Kyushu Institute of Technology
[2] 学習院大学
Gakushuin University
[a)] nac24nh@gmail.com
[b)] imamura.kit@gmail.com
[c)] kuboyama@tk.cc.gakushuin.ac.jp
[d)] hirata@ai.kyutech.ac.jp
[e)] shino@ai.kyutech.ac.jp

して保存しておき，質問のスケッチとの距離を計算して，全探索により解候補を選択する．

それに対し，16 ビットのパターンの個数は 2^{16} 個しかないので，スケッチをキーとするバケット法でデータを管理することができる．そうすると，第 1 段階検索は，2^{16}=約 6 万 6 千個のスケッチとの照合を行えばよく，データベースサイズに依らない一定コストで高速に実行することが可能となる．また，16 ビットスケッチの第 1 段階検索の候補選択に用いられる質問のスケッチと近いスケッチは，約 6 万 6 千個のうちのごくわずかでしかない．したがって，近い順にスケッチを列挙するアルゴリズムを利用すれば，スケッチ間の照合を行わずに，事実上コストを無視できるほど高速化することが可能である．また，同じスケッチをもつデータベースのオブジェクトが 10 個以上存在することが多くなるので，オブジェクトをスケッチ順にソートしておくことで，第 2 段階の検索におけるメモリ局所性を向上できる．

1.3 スケッチの列挙による高速化

ここで，スケッチの列挙による第 1 段階検索の高速化について，概略を説明しておこう．検索前にすべての 16 ビットパターンを ON ビットの個数の昇順にソートしたものを準備しておく．質問とこのビットパターンとのビットごとの排他論理和（XOR）を求めると質問とのハミング距離の順にスケッチを列挙することができる．この列の先頭部分のみを用いて第 2 段階検索を実行する．この手法により，第 1 段階検索では，スケッチ間のハミング距離計算が不要となり，検索コストをほとんど必要としなくなる．

例を用いて，第 1 段目の高速化のためのスケッチの列挙法を説明しよう．スケッチの幅を 3 ビットとする．スケッチ間の距離はハミング距離を用いるとする．スケッチ間のハミング距離は，0, 1, 2, 3 の 3 種類である．質問のスケッチ $s = 011$ とすると，それぞれの距離となるスケッチは，以下の通りである．

距離 0 のスケッチ

s と一致するスケッチは 1 個，s 自身，つまり，011.

距離 1 のスケッチ

s と 1 ビットだけ異なるスケッチは 3 個あり，1 箇所だけ 1 の 3 ビット列（001, 010, 100）と s との XOR で求められる．それぞれ，$s \oplus 001 = 010$, $s \oplus 010 = 001$, $s \oplus 100 = 111$ である．

距離 2 のスケッチ

s と 2 ビットだけ異なるスケッチ 3 個は，2 箇所だけ 1 の 3 ビット列（011, 101, 110）と s との XOR で求められる．それぞれ，$s \oplus 011 = 000$, $s \oplus 101 = 110$, $s \oplus 110 = 101$ である．

距離 3 のスケッチ

s と 3 ビットすべて異なるスケッチは 1 個．3 箇所すべて 1 の 3 ビット列 111 と s の XOR で求められる．$s \oplus 111 = 100$

このように，3 ビットスケッチの場合には，ON ビット数の昇順のビットパターンの列 000, 001, 010, 100, 011, 101, 110, 111 と質問のスケッチとの XOR を取ることで質問とのハミング距離の昇順にスケッチを列挙することができる．XOR に用いるビットパターン列は質問に依らないので，ON ビットの個数の昇順に並べたものを事前準備しておくことができる．データベースのオブジェクトはスケッチをキーとしたバケット法で管理することができるので，第 1 段階の検索においては，オブジェクトと質問のスケッチ間の距離を計算する必要がない．これにより，事実上，第 1 段階の検索コストは無視することができる．

1.4 スケッチによる選択の高精度化

われわれは，第 1 段階における解候補選択基準として，ハミング距離の代わりに，距離下限値を用いたスコアを用いて，検索精度・速度を向上することができる手法を提案している [9]．これは，球面分割によるスケッチは次元縮小射影 Simple-Map [10] の量子化像とみなすことができるという事実に基づいた手法である．スケッチの各ビットに対して，質問 q とオブジェクト x のスケッチが不一致であるときの q と x 間の距離の下限値を求めることができる．距離下限値を用いたスコアには，最大値を取る $score_\infty$ と総和を取る $score_1$ がある．この手法を 16 ビットスケッチに適用することができる．列挙による高速化も可能である．

$score_\infty$ の場合，それぞれの質問に対して，スケッチの各ビットに対応する質問との距離下限値の表を用意し，距離下限値の順位を求めておけば，質問とのスコアの小さい順にスケッチを列挙することができる．それを利用して，スコアの小さい順にオブジェクトが K 個になるまで第 2 段階検索を実行することができる．各質問のための準備のためのコストは無視できるほど小さいので，実際には，第 1 段階目の検索コストは，ハミング距離のときと同様に，事実上無視できる．

$score_1$ の場合，$score_\infty$ の場合と同様に距離下限値の順位を求めておき，さらに優先度付き待ち行列を用いることでスコアの小さい順にスケッチを列挙することができる．ただし，$score_\infty$ の順でもスケッチの列挙のコストは，非常に小さいが，キュー操作と余分にキューに放り込むためコストがかかるため，無視できるほど小さくはない．

実際の画像データベースを用いた実験により，16 ビットスケッチを用いた提案手法は従来の 32 ビットスケッチを用いた手法より 20 倍程度高速であることを確認することができる．

2. 準備

2.1 スケッチを用いる最近傍検索

与えられたデータベースのデータは，0〜$n-1$ の自然数で索引付けされているとする．n 個のデータからなるデータベースは $db=\{x_0,\cdots,x_{n-1}\}\subseteq\mathcal{U}$ とする．ここで，$\mathcal{U}$ はデータ空間である．データ x_i と x_j の間の非類似度は，距離 $D(x_i,x_j)$ で測るものとする．質問 $q\in\mathcal{U}$ に関する**最近傍検索**とは，すべての $y\in db$ に関して $D(q,x)\leq D(q,y)$ となるような $x\in db$ を見つけることである．スケッチを用いた最近傍検索は，以下のように行う．ここで，s はデータをスケッチに射影する関数とし，$K\geq 1$ とする．

(1) 準備段階:
あらかじめ，すべてのスケッチ $s(x_0),\ldots,s(x_{n-1})$ を求めて保存しておく．

(2) 第 1 段階 (スケッチ間のハミング距離によるフィルタリング):
質問 q のスケッチ $s(q)$ に近いスケッチ $s(x_{i_0}),\ldots,s(x_{i_{K-1}})$ をもつ K 個の解候補 $x_{i_0},\ldots,x_{i_{K-1}}$ を選ぶ．

(3) 第 2 段階 (実距離計算による最近傍検索):
解候補 $x_{i_0},\ldots,x_{i_{K-1}}$ から最近傍データを選ぶ．

スケッチは，オリジナルの特徴データに比べて比較的小さな構造である．たとえば，本論文で報告する実験では，64 バイトの画像特徴データに対して 32 ビットや 16 ビットのスケッチを使用する．従来のスケッチを用いる検索の第 1 段階においては，特徴間の実距離よりもビット演算によって容易に計算できるハミング距離を用いる．しかしながら，スケッチ間のハミング距離は元の距離関係を完全に保存できないので，それらをフィルタとして用いる．検索の**精度**は，正しく最近傍が求められる確率である．スケッチを用いる検索では，第 1 段階で求める K 個の候補に最近傍が含まれている確率である．第 1 段階における解候補数 K が大きいほど精度は上がるが，検索は遅くなる．したがって，スケッチにおける最も重要な課題の一つは，より小さい K で高精度を達成する，あるいは，許容可能な誤差で検索速度をあげることである．

2.2 球面分割に基づくスケッチ

本論文で，我々は**球面分割**に基づくスケッチを用いる．中心点と半径のペア (p,r) を**ピボット**と呼ぶ．球面分割 BP は，以下のように定義される．

$$BP_{(p,r)}(x)=\begin{cases}0 & \text{if } D(p,x)\leq r\\ 1 & \text{otherwise}\end{cases}$$

BP に基づく**幅** w のスケッチ関数 s_P は w 個のピボット集合 $P=\{(p_0,r_0),\ldots,(p_{w-1},r_{w-1})\}$ で以下のように定義される．

$$s_P(x)=BP_{(p_{w-1},r_{w-1})}(x)\ldots BP_{(p_0,r_0)}(x)$$

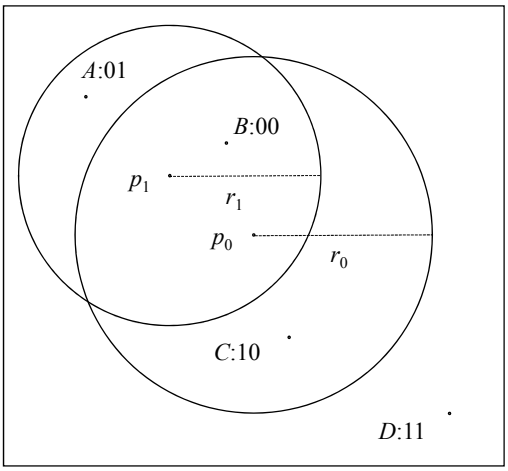

図 1 2 個の球による 2 ビットスケッチ

図 1 においてユークリッド平面上の 4 点 A,B,C,D を考える．2 個のピボット集合 $P=\{(p_0,r_0),(p_1,r_1)\}$ を用いると，スケッチは $s_P(A)=01, s_P(B)=00, s_P(C)=10, s_P(D)=11$ となる．q は両方の球の外側の任意の質問であるとする．q と A,B,C,D のスケッチとのハミング距離はそれぞれ 1, 2, 1, 0 である．第 1 段階における従来の優先順位は $D<A=C<B$ である．A と C は q からのハミング距離によって区別できないことに注意しよう．

2.3 質問とスケッチ間の距離下限

本研究では，第 1 段階の検索において，距離下限値を用いた検索優先順序付け [9] を用いる．ピボット集合を $P=\{(p_0,r_0),\ldots,(p_{w-1},r_{w-1})\}$ とする．以下によって $D(q,x)$ の距離下限値 $e_i(q,s_P(x))$ を求めることができる．

$$e_i(q,s_P(x))=\begin{cases}0 & \text{if } BP_{(p_i,r_i)}(q)=BP_{(p_i,r_i)}(x)\\ |D(p_i,q)-r_i| & \text{otherwise}\end{cases}$$

第 1 段階における解候補選択の基準として距離下限値 $e_i(q,s_P(x))$ を用いて優先付けする．下限値の最大である以下の $socre_\infty$ を優先順位とするとき，真の距離下限値であるので安全な枝刈りが可能である．

$$score_\infty(q,s_P(x))=\max_{i=0}^{w-1}e_i(q,s_P(x))$$

また，距離下限の総和である以下の $score_1$ は，もはや距離下限ではないが，これを用いるとより高精度の検索が行える．

$$score_1(q,s_P(x))=\sum_{i=0}^{w-1}e_i(q,s_P(x))$$

2.4 スケッチの最適化

われわれは，スケッチを用いた検索の精度を高めるために，衝突確率が小さくなるようにピボット集合 P を選ぶ．異なるデータ x と y に対して，それらのスケッチが一致するとき，すなわち，

$$s_P(x) = s_P(y)$$

となるとき，**衝突**が発生するという．本論文では，QBP [9] を用いてスケッチの最適化を行う．数百万件のデータベースに対して，最適化された 16 ビットスケッチを用いる場合は，各スケッチに射影されるデータ数はある程度均等になることが期待できる．

3. 16 ビットスケッチを用いる高速検索

16 ビットスケッチの総数は 2^{16}=約 6 万 6 千であるので，数百万個の個々のデータのスケッチとの照合を行うよりも，すべての 16 ビットスケッチとの照合を行う方が効率的であり，そのコストはデータベースサイズに依らない．各スケッチに射影されるデータ数は，ある程度均等であると考えられるので，実際の検索時には，質問のスケッチと近似するごく一部のスケッチしか必要ない．したがって，質問のスケッチに近い順にスケッチを列挙するアルゴリズムを利用することにより，第 1 段階検索のコストを非常に小さくすることが可能である．

3.1 ハミング距離を用いる検索の高速化

16 ビットのスケッチを用いる検索は，ハミング距離を用いる場合は比較的容易に高速化することができる．質問 q のスケッチ $s(q)$ をとし，m を ON ビットの個数が h のビットパターンとすると，排他的論理和 $\oplus$ の性質により，

$$s(q) \oplus (s(q) \oplus m) = m$$

であるので，$s(q)$ と $s(q) \oplus m$ のハミング距離は h である．事前に，すべての 16 ビットパターンを ON ビットの個数の昇順にソートしたものを準備しておく．質問のスケッチとこのビットパターンとの XOR を求めるとハミング距離の順にスケッチを求めることができるので，これを用いて順次第 2 段階検索を実行する．これにより，事実上第 1 段階の検索コストをほぼ無視できるようになる．

Algorithm1 に 16 ビットスケッチを用いた最近傍検索手法を示す．ここに，データベースは，以下のように，スケッチをキーとするバケットを用いて管理すると仮定している．

$x[0], x[1], \cdots, x[n-1]$: 特徴データの配列，ただし，データがメモリ上でスケッチ順になるようにソートしておく．（ポインタを介した間接的なソートではない）

$id[i]$: 特徴データ $x[i]$ のデータ ID.

$f[s]$: スケッチが s であるデータの配列 x における先頭位置.

$n[s]$: スケッチが s であるデータ数.

このようにすると，スケッチが s であるデータは，

$$x[f[s]], x[f[s]+1], \cdots, x[f[s]+n[s]-1]$$

になる．また，ハミング距離による検索のための準備として，すべての 16 ビットパターンを ON ビット数の昇順に並べた配列 m を用意しておく．

$$m[0], m[1], \cdots, m[2^{16}-1]$$

3.2 距離下限値を用いる検索の高速化 ($score_\infty$)

距離下限値を利用した $score_\infty$ を用いる場合は，距離下限値が質問毎に異なるため，ハミング距離を利用する場合と異なり，事前に準備したビットパターン列との XOR でスケッチを列挙することはできない．しかし，以下に説明するように，質問のスケッチとの $score_\infty$ が小さい順にスケッチを列挙することができる．

まず，3 ビットスケッチのときの具体例を示す．ビット列を 2 進数と見たときの 2^i の位を位置 i とする（$i = 0, 1, 2$）．

(p_i, r_i)：位置 i に対応する球面分割のピボット．

$P = \{(p_0, r_0), (p_1, r_1), (p_2, r_2)\}$: ピボット集合．

q: 質問点．

$e_i = |D(q, p_i) - r_i|$: 質問点と位置 i のビットが異なるスケッチを持つデータとの距離下限．任意のデータ x に対して，

$$BP_{(p_i,r_i)}(q) \neq BP_{(p_i,r_i)}(x) \rightarrow D(q,x) \geq e_i$$

が成立つ．簡単のため，これらの距離下限は，つぎを満たしているとする．

$$e_2 \geq e_1 \geq e_0$$

そうすると，スケッチ間の $score_\infty$ は，小さい順に 0, e_0, e_1, または e_2 の高々 4 種類しかない．質問のスケッチを $s_P(q) = 011$ とすると，それぞれの $score_\infty$ をもつスケッチは，以下のようになる．

$score_\infty$ = 0 のスケッチ

011 自身.

$score_\infty$ = e_0 のスケッチ

011 と位置 0 のビットだけ異なるもの，つまり，010. これは $s_P(q)$ と 001 の XOR で求められる．

$score_\infty$ = e_1 のスケッチ

011 と位置 2 のビットは同じで，位置 1 のビットが異なり，位置 0 のビットは任意．つまり，000 と 001．これらは，$s_P(q)$ とそれぞれ 010, 011 の XOR である．

$score_\infty$ = e_2 のスケッチ

011 と位置 2 のビットが異なり，残りは任意．つまり，100, 101, 110, 111．これらは，$s_P(q)$ とそれぞれ 100, 101, 110, 111 の XOR.

このように，$s_P(q)$ との $score_\infty$ が小さい順のスケッチは，

```
/* x[0],x[1],⋯,x[n−1]: スケッチ順にソートした特徴データの配列．*/
/* id[i]: 特徴データ x[i] のデータ ID */
/* f[s]: スケッチが s であるデータの配列 x における先頭位置 */
/* n[s]: スケッチが s であるデータ数 */
/* K: 第 1 段階で得る候補数＝実距離計算回数 */
/* m[0],m[1],⋯,m[2^w−1]: ビットパターンを ON ビット数の昇順に並べた配列 */
/* w: スケッチの幅（ビット数） */
1  function NNSearchByHamming(query)
2      (NN, nearest, checked) ← ("none", ∞, 0);
3      for i = 0 to 2^w − 1 do
4          s ← sketch(query) ⊕ m[i];
5          (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
6          if checked ≥ K then
7              return NN;
8      return NN;
9  function Search(query, s, NN, nearest, checked)
10     for i = f[s] to f[s] + min(n[s], K − checked) − 1 do
11         if D(query, x[i]) ≤ nearest then
12             (NN, nearest) ← (id[i], D(query, x[i]));
13     return(NN, nearest, checked + min(n[s], K − checked));
```

Algorithm 1　NNSearchByHamming

```
/* w : スケッチの幅（ビット数） */
/* K : 第 1 段階における候補数 = 実距離計算回数 */
1  function SearchByScoreInf(query)
2      query に対する距離下限の順位表 bidx[0],…,bidx[w−1] を準備する;
3      (NN, nearest, checked) ← ("none", ∞, 0);
4      s ← sketch(query);
5      (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
6      if checked ≥ K then
7          return NN;
8      for i = 0 to 2^w − 1 do
9          s ← s ⊕ (1 ⟨⟨ bidx[bitcount(i ⊕ (i + 1)) − 1]);
10         (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
11         if checked ≥ K then
12             return NN;
13     return NN;
```

Algorithm 2　SearchByScoreInf

$s_P(q)$ と 2 進数で小さい順となるつぎのビットパターン列との XOR で求められる．

$$000, 001, 010, 011, 100, 101, 110, 111$$

$score_\infty$ はこのビットパターンの先頭の ON ビットの位置で決まることに注意すれば，これをグレイコード順に並べてもよいことがわかる．

$$000, 001, 011, 010, 110, 111, 101, 100$$

これは，2 進数の値の順序とグレイコードの順序は，どちらも先頭の ON ビットが立っている位置の昇順であるから

表 1　グレイコード生成とスケッチの列挙

グレイコード	スケッチの列挙	$score_\infty$
000	011	0
$000 \oplus 001 = 001$	$011 \oplus 001 = 010$	e_0
$001 \oplus 010 = 011$	$010 \oplus 010 = 000$	e_1
$011 \oplus 001 = 010$	$000 \oplus 001 = 001$	e_1
$010 \oplus 100 = 110$	$001 \oplus 100 = 101$	e_2
$110 \oplus 001 = 111$	$101 \oplus 001 = 100$	e_2
$111 \oplus 010 = 101$	$100 \oplus 010 = 110$	e_2
$101 \oplus 001 = 100$	$110 \oplus 001 = 111$	e_2

である．どのビット位置を反転させるかの系列は，表 1 のように，0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0 のようになっている．このグレイコードの性質を利用すると，効率的な列挙が可能となる．質問に対する各ビット位置の距離下限の順序が決まれば，それを用いた相対的な位置のビット反転操作を行うのである．反転させるビット位置は

$$bitcount(i \oplus (i+1)) - 1$$

で求めることが可能である．オール 0 のビットパターンから始めず，表 1 のように，質問のスケッチ $s_P(q)$ から始めることで質問からのスコアが小さい順にスケッチを列挙することが可能である．グレイコードの性質を利用することによりスケッチを列挙するアルゴリズムは非常に単純になる．なぜならば，その並びのつぎのスケッチを得るための操作は，ちょうど *1* ビットの反転操作で行えるからである．

ここで，上に述べた手法で正しくスケッチを列挙できる理由を説明しておく．整数 i に対応するグレイコードを $g(i)$ とする（$i = 0, 1, \ldots$）．つまり，$g(0) = 000$, $g(1) = 001, \ldots, g(7) = 100$ である．そうすると，生成すべき列挙の第 i 番目のスケッチは，$s_P(q) \oplus g(i)$ である．排他的論理和 $\oplus$ の性質により，つぎが成り立つ．

$$(s_P(q) \oplus g(i)) \oplus (s_P(q) \oplus g(i+1)) = g(i) \oplus g(i+1)$$

ここで，グレイコードの性質より，$g(i) \oplus g(i+1)$ は ON ビットが 1 個だけのビットパターンであることに注意しよう．

$$g(i) \oplus g(i+1) = (1 \langle\langle (bitcount(i \oplus (i+1)) - 1))$$

であるので，$s_P(q)$ から始めて，グレイコードを生成するときと同じビット反転操作を適用すると，求めるスケッチが列挙できる．ここで，$\langle\langle$ は左論理シフト演算子である．

距離下限は，$e_2 \geq e_1 \geq e_0$ を満たしているとは限らないので，実際には，これらの順位を求めておく．アルゴリズム (Algorithm2) では，$bidx[0], \cdots, bidx[w-1]$ は，$0, 1, \cdots, w-1$ の並べ替えで，つぎを満たしているとする．

$$e_{bidx[w-1]} \geq \cdots \geq e_{bidx[1]} \geq e_{bidx[0]}$$

また，関数 Search は Algorithm 1 で示している．

3.3 距離下限値を用いる検索の高速化 ($score_1$)

アルゴリズム (Algorithm3) は，質問との $score_1$ が小さい順にスケッチを列挙できることを利用した最近傍検索手法である．アルゴリズム (Algorithm2) と同様に，各質問に対して，まず，距離下限をとその順位表を求めておく．このアルゴリズムで質問との $score_1$ の昇順にスケッチが正しく列挙できていることを以下で説明する．簡単のために，距離下限は，

$$e[w-1] \geq \cdots \geq e[1] \geq e[0]$$

を満たしていると仮定する．実際には，アルゴリズム (Algorithm2) と同様に，距離下限の順位表を用いて，相対ビット位置で操作を行っている．

スケッチの列挙には優先度付き待ち行列 PQ を用いている．PQ の要素は，$(score, s, i)$ の組で，s はスケッチ，$score = score_1(query, s)$，$MSB(qs \oplus s) = i1$ である．ここで，MSB は最左ビット位置，qs は query のスケッチとする．

まず，ループの外側では，質問のスケッチ qs と一致するもの，つまり $score_1 = 0$ であるものを検索している．続いて，$(e[0], qs \oplus 1, 1)$ を PQ に挿入している．このとき，$MSB(qs \oplus (qs \oplus 1)) = MSB(1) = 0$ であり，$score_1(query, qs \oplus 1) = e[0]$ であることに注意しよう．

ループ内では，PQ からスコアが最小の $(score, s, i)$ ものを取り出して，スケッチが s であるものを検索し，その後，2 個の要素 $(score - e[i-1] + e[i], (s \oplus (1 \langle\langle i)) \oplus (1 \langle\langle i1), i+1)$ と $(score + e[i], s \oplus (1 \langle\langle i), i+1)$ を PQ に挿入している．ここで，PQ から取り出されたスケッチ s と質問のスケッチ qs に対して，

$$s \oplus qs = 1X，\text{ただし，} X \text{ は } i-1 \text{ 桁の 2 進数}$$

となっていて，PQ に挿入されるスケッチは，$10X$ と $11X$ となっていることに注意しよう．このことから，アルゴリズムでは，幅 w ビットのすべてのスケッチを重複なしで列挙する手順を用いていることがわかる．また，列挙されるスケッチのスコアが $score_1$ の昇順になっていることは，PQ から取り出されたスケッチのスコアはその後で挿入されるスケッチのスコアより大きくないことから明らかである．

4. 実験

約 2,900 本の動画のコマ画像から 2 次元周波数スペクトラムとして特徴抽出した約 700 万件の 64 次元の画像データを用いた実験結果を示す．

スケッチは，32 ビットと 16 ビットのものを用いる．いずれも，QBP [9] を用いて衝突が少なくなるように選んだピボット集合を用いる．画像データに対して，16 ビットスケッチをキーとするバケットの様子を調べたところ，平均要素数は 105，空バケット数は 908 で全体の 1.5%，要素数 10 以上のバケット数の割合は 87%であった．多くのバケットは 10 個以上の要素を有するので，スケッチの順番でデータをソートすることによる第 2 段階の高速化が期待できる．

ランダム生成されたデータは高次元空間において近くにデータが存在することがほとんどないので，最近傍検索の質問には不適切である．そこで，データベースからランダムに選んだ 2 個のデータ x と y を用いて，x に対して，ノイズとして y を 5%, 10%, ..., 50% の割合で混ぜて質問を作る．たとえば，ノイズレベル 5% の質問 q はデータベースからランダムに選んだデータ x と y をそれぞれ 95% と 5% の重

```
/* w : スケッチの幅（ビット数） */
/* K : 第 1 段階における候補数 = 実距離計算回数 */
/* PQ : 優先度付き待ち行列（要素は（score, sketch, i）で score が優先度） */
1  function SEARCHBYSCORE1(query)
2      qurey に対する距離下限 e[0],...,e[w−1] を求める;
3      query に対する距離下限の順位表 bidx[0],...,bidx[w−1] を準備する;
4      (NN, nearest, checked) ← ("none", ∞, 0);
5      s ← sketch(query);
6      (NN, nearest, checked) ←SEARCH(query, s, NN, nearest, checked);
7      if checked ≥ K then
8          return NN;
9      PQ ← (e[bidx[0]], s ⊕ 1⟨⟨bidx[0], 1);
10     while PQ is not empty do
11         (score, s, i) ← PQ;
12         (NN, nearest, checked) ←SEARCH(query, s, NN, nearest, checked);
13         if checked ≥ K then
14             return NN;
15         if i < w then
16             s ← s ⊕ (1⟨⟨bidx[i]);
17             PQ ← (score − e[bidx[i] − 1] + e[bidx[i]], s ⊕ (1⟨⟨bidx[i] − 1), i + 1);
18             PQ ← (score + e[bidx[i]], s, i + 1);
```

Algorithm 3 SEARCHBYSCORE1

みで加重和したものである．それぞれのノイズレベルについて，1,000 質問作る．質問をノイズレベルによって 5 個のグループ 5 – 10%, 15 – 20%,25 – 30%,35 – 40%,45 – 50%, に分け，それぞれ，very-near, near, middle, far, very-far とする．

実験に用いた PC の諸元を表 2 に示す．

表 2 実験環境

CPU	Intel(R) Xeon(R) CPU E5-2640 2.5 GHz
memory	64GBytes

検索精度と検索速度を表 3 に示す．ここで，score は検索優先順序，width はスケッチの幅（ビット数），K は第 1 段階での候補数（データベースサイズ約 700 万件に対する割合（%））, Sketches は検索時に列挙されたスケッチ数（16 ビットのみ，6 万 4 千に対する割合（%），100%は列挙を用いない場合），time は 1 質問当たりの検索時間（ミリ秒）を表す．実距離計算回数 K は検索精度が同程度（90%）以上になるようにする．All は全質問に対する平均精度を表す．

これらの表から，ノイズレベルが高くなるにつれて，つまり，最近傍解が遠くなるにつれて，検索精度が低くなっていることがわかる．これは，高次元データにおける「次元の呪い」の影響と考えられる．優先順位付けに $score_\infty$ や $score_1$ を用いると，より小さい K で Hamming と同じ検索精度が達成できるため，検索速度が向上している．16 ビットスケッチの枚挙による高速化の有効性も確認できる．枚挙を用いるかどうかで精度に若干の差が出ているが，これは同一の優先順位のものがある場合に手法によって選択されるものが異なるからである．また，検索時に列挙される 16 ビットスケッチは，ごくわずかであることがわかる．列挙による高速化を用いると，検索速度は，従来の 32 ビットスケッチを用いる場合よりハミングで 8 倍，$score_\infty$ で 10 倍，$score_1$ で 19 倍程度高速化できている．

つぎに，データベースのオブジェクトをスケッチ順にソートしておくことによる第 2 段階の検索におけるメモリ局所性の向上に関する実験を表 4 に示す．あらかじめデータベースのオブジェクトをソートしておくことで，それぞれ 3 倍程度高速化できている．

表 4 ソートによる高速化

score	Hamming		$score_\infty$		$score_1$	
sort	あり	なし	あり	なし	あり	なし
time(ms)	16.2	53.9	10.7	34.3	5.56	17.5

最後に 1 質問あたりのスケッチの枚挙にかかる時間を表 5 に示す．ここでは，それぞれ $K = 5\%$ で実験している．ハミングと $score_\infty$ は同程度の非常に低いコストで枚挙できているが，$score_1$ は約 10 倍コストがかかっている．これは $score_1$ は優先度付き待ち行列を用いており，DEQ1 回に対して ENQ2 回行っているためである．

5. 結論と今後の課題

16 ビットスケッチに対して，質問のスケッチに近い順

表 5　1 質問当たりのスケッチ枚挙時間（$K = 5\%$）

score	Hamming	$score_\infty$	$score_1$
time(ms)	0.134	0.124	1.34

にスケッチを列挙することによる第 1 段階検索の高速化を行った．スケッチを 32 ビットからより狭い 16 ビットにし，効率的な第 1 段階検索とバケット法によるデータ管理によってハミングで約 8 倍，$score_\infty$ で約 10 倍，$score_1$ で約 19 倍の高速化を実現した．$score_1$ は他の手法に比べてスケッチの列挙にかかるコストは大きいが，同程度の検索精度を達成するために必要な K が小さくて済むため，検索時間は最も短くなった．また，データベースのオブジェクトをスケッチ順にソートしておくことで，第 2 段階の検索におけるメモリ局所性を向上できることが確認できた．

データベースの大きさ n と最適なスケッチの幅 w の関係についてさらに調査する必要がある．本論文では，n は数百万と仮定したが，より大きなデータベースに対しては，w を 16 より大きくした方がよいかもしれない．

本研究での実験では，スケッチの最適化は衝突確率を評価指標とし，それを最小化する発見的手法 QBP [9] を用いた．衝突確率は，一種の焼きなまし法である AIR を用いれば，QBP よりも小さい衝突確率をもつスケッチのピボット集合を得ることができるが，検索精度は向上しないことが報告されている [11]．しかし，今回の提案手法ではバケット法によるデータ管理を行っているので，衝突確率が小さいスケッチを用いることの利点として，メモリアクセスの局所化による速度向上の可能性も考えられる．いずれにしても，スケッチの最適化について，さらに調査する必要があると思われる．

参考文献

[1] A. Müller and T. Shinohara. Efficient similarity search by reducing i/o with compressed sketches. In *Proc. SISAP'09*, pp. 30–38, 2009.

[2] V. Mic, D. Novak, and P. Zezula. Speeding up similarity search by sketches. In *Proc. SISAP 2016*, pp. 250–258, 2016.

[3] W. Dong, M. Charikar, and K. Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *Proc. ACM SIGIR'08*, pp. 123–130, 2008.

[4] V. Mic, D. Novak, and P. Zezula. Improving sketches for similarity search. In *Proc. MEMICS'15*, pp. 45–57, 2015.

[5] Z. Wang, W. Dong, W. Josephson, M. Charikar Q. Lv, and K. Li. Sizing sketches: A rank-based analysis for similarity search. In *Proc. ACM SIGMETRICS'07*, pp. 157–168, 2007.

[6] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. SODA 1993*, pp. 311–321. ACM Press, 1993.

[7] A. Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *Proc. SIGMOD'84*, pp. 47–57. ACM Press, 1984.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. VLBD'97*, pp. 426–435, 1997.

[9] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Nearest neighbor search using sketches as quantized images of dimension reduction. In *Proc. ICPRAM 2018*, 2018.

[10] T. Shinohara and H. Ishizaka. On dimension reduction mappings for approximate retrieval of multi-dimensional data. In *Progress of Discovery Science, LNCS 2281*, pp. 89–94. Springer Berlin / Heidelberg, 2002.

[11] Y. Imamura, N. Higuchi, T. Kuboyama, K. Hirata, and T. Shinohara. Pivot selection for dimension reduction using annealing by increasing resampling. In *Proc. LWDA 2017*, 2017.

表 3 検索精度と検索速度

score	Hamming			$score_\infty$			$score_1$		
width	32	16		32	16		32	16	
K	2.0%	6.1%		1.5%	4.2%		1.0%	1.9%	
sketches	–	100%	5.8%	–	100%	4.1%	–	100%	1.26%
time (ms)	139	22.0	16.2	106	16.8	10.7	107	12.0	5.56
All	91.5	90.2	90.1	90.1	92.0	90.0	93.8	91.4	90.2
very-near	100	100	100	100	100	100	100	100	100
near	100	99.7	99.6	99.9	100	99.8	100	100	100
middle	97.4	95.4	95.5	97.3	97.3	96.2	99.1	97.3	97.2
far	84.4	82.4	82.04	81.8	85.8	81.9	88.7	85.3	82.8
very-far	76.0	73.8	73.2	71.6	77.0	72.0	81.5	74.7	71.3