Vol.2018-MPS-121 No.8
2018/12/17

# 狭い16ビットのスケッチを用いた高速最近傍検索

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

**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 10 倍程度の高速化が実現できる．

# Fast Nearest Neighbor Search with Narrow 16-bit Sketch

Naoya Higuchi[1,a)]　Yasunobu Imamura[1]　Tetsuji Kuboyama[2]　Kouichi Hirata[1]
Takeshi Shinohara[1]

## 1. はじめに

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] 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，スケッチ間のハミング距離に応じて候補を選択する．第 2 段階では，元の空間内の距離を使用して，質問とと候補を比較することにより，最近傍を選択する．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 32 ビットや 64 ビット必要であると考えられてきた．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 4 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

[1] 九州工業大学
Kyushu Institute of Technology
[2] 学習院大学
Gakushuin University
[a)] nac24nh@gmail.com

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 4 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．

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．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．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．この手法を 16 ビットスケッチに適用することができる．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．

実際の画像データベースや音楽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 10 倍程度の高速であることを確認することができる．

## 2. 準備

以下の議論で用いる概念について簡単に説明しておく．

### 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．2 つのデータ $x_i$ と $x_j$ 間の非類似度は，距離 $D(x_i, x_j)$ で定義される．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:
すべてのスケッチ $s(x_0), \ldots, s(x_{n-1})$ を計算する．

(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，われわれは実験で 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．検索の第 1 段階において，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，スケッチは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．

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

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 $P = \{(p_0,r_0),...,(p_{w-1},r_{w-1})\}$ で以下のように定義される．

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

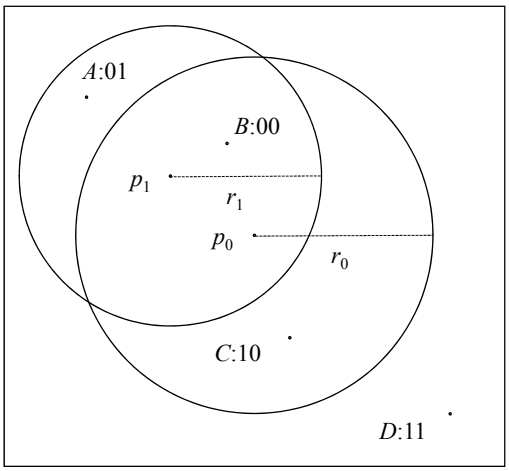


**図 1** 球 2 つによる 2 ビットスケッチ

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 $P = \{(p_0,r_0),...,(p_{w-1},r_{w-1})\}$ 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．

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

## 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 4 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えば十分であり，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 段階検索はほぼコストを無視できるように高速化が可能である．

### 3.1 16 ビットスケッチのハミング距離を用いる検索の高速化

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．

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

```
/* 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  function Search(query, s, NN, nearest, checked)
9      for i = f[s] to f[s] + n[s] − 1 do
10         if D(query, x[i]) ≤ nearest then
11             (NN, nearest, checked) ← (id[i], D(query, x[i]), checked + 1);
12             if checked ≥ K then
13                 return(NN, nearest, checked);
```

**Algorithm 1** NNSearchByHamming

### 3.2 16 ビットスケッチの距離下限値を用いる検索の高速化

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，$s_P(q)$ 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である．どのビット位置を反転させるかの系列は，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，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．

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，つぎをみたしているとする．

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

また，関数 Search は Algorithm 1 で示している．

優先順位付けには $score_\infty$ よりも $score_1$ を用いる方が精度がよくなることがわかっている．本論文での実験では，$score_1$ を用いた 16 ビットスケッチによる検索では，列挙による高速化を用いずに，$2^w = 2^{16}$ 個のすべてのスケッチに対して $score_1$ を求める素朴な方法を用いている．

## 4. 実験

本章では，以下のような画像や音楽データを用いた実験結果を示す．

- 画像: 2,900 本の動画から 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 次元データ
- 音楽: 1,400 本の音楽 CD からメル周波数軸変換によって特徴抽出した約 700 万件の 96 次元データ

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．

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，最近傍検索の実験には不適切である．したがって，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% の割合で混ぜた質問を作る．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．

実験に用いた PC は，CPU Intel(R) Xeon(R) CPU E5-2640 2.5 GHz，メモリ 64GBytes である．

画像データおよび音データに対する検索精度をそれぞれ表 1，表 2 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（1st st. は第 1 段階の検索時間）．枚挙を用いる場合は，第 1 段階検索のコストは分離することは困難なので省略している．実距離計算回数 $K$ は検索精度が同程度（画像では 70%，音データでは 65%）以上になるように 32 ビットでは 0.1%，16 ビットでは 1.0% とする．All は全質問に対する平均精度を表す．32 ビットスケッチに対する $K = 0.1\%$ の設定は，筆者らの研究室における R-Tree による検索速度（100 ミリ秒／質問）より高速である程度の精度（70%程度）の検索を達成するものである．

**表 1** 画像データに対する検索精度

| score | | Hamming | | | $score_\infty$ | | | $score_1$ | |
|---|---|---|---|---|---|---|---|---|---|
| width | | 32 | 16 | | 32 | 16 | | 32 | 16 |
| $K$ | | 0.1% | 1.0% | | 0.1% | 1.0% | | 0.1% | 1.0% |
| sketches | | – | 100% | 0.76% | – | 100% | 0.73% | – | 100% |
| time | 1st st. | 28.7 | 4.36 | – | 35.6 | 3.23 | – | 32.0 | 4.90 |
| (ms) | total | 29.8 | 7.16 | 2.85 | 36.9 | 6.06 | 2.68 | 33.2 | 7.76 |
| 検索精度 | | 70.2 | 73.4 | 73.0 | 74.3 | 79.7 | 79.7 | 80.2 | 85.1 |

**表 2** 音データに対する検索精度

| score | | Hamming | | | $score_\infty$ | | | $score_1$ | |
|---|---|---|---|---|---|---|---|---|---|
| width | | 32 | 16 | | 32 | 16 | | 32 | 16 |
| $K$ | | 0.1% | 1.0% | | 0.1% | 1.0% | | 0.1% | 1.0% |
| sketches | | – | 100% | 0.55% | – | 100% | 0.48% | – | 100% |
| time | 1st st. | 30.3 | 4.33 | – | 36.3 | 3.23 | – | 34.4 | 4.63 |
| (ms) | total | 31.7 | 7.89 | 3.58 | 38.0 | 6.82 | 3.38 | 36.0 | 8.30 |
| 検索精度 | | 65.5 | 66.8 | 66.0 | 63.6 | 75.1 | 73.4 | 72.3 | 77.8 |

優先順位付けにハミング距離を用いる場合は，いずれのデータベースに対しても，16 ビットスケッチ（$K = 1.0\%$）で 32 ビットスケッチ（$K = 0.1\%$）と同等の検索精度を達成できる．優先順位付けに $score_\infty$ や $score_1$ を用いると，検索精度が向上している．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 10 倍程度高速化できている．$score_1$ に対しては，列挙による手法を使わないので，4 倍程度の高速化に留まっているが，最高精度（画像：85.1%，音：77.8%）を達成している．

90%以上の検索精度を達成するためには，従来法の 32 ビットスケッチを用いる場合には，ハミング距離では，画像で $K = 2\%$，音で $K = 3.5\%$ とする必要があり，1 質問当たりの検索時間は，それぞれ，139 ミリ秒，227 ミリ秒と大きくなる．これに対し，提案手法を用いると，$score_\infty$ では，画像で $K = 5\%$，音で $K = 5.5\%$ とすればよく，検索時間は，それぞれ，12.8 ミリ秒，25.1 ミリ秒である．また，$score_1$ を用いると $K$ をあまり大きくしなくても高精度を達成でき（画像：2.5%，音：4%），列挙による高速化を用い

```
/* 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 */
function SearchByScoreInf(query)
    query に対する距離下限の順位表 bidx[0],...,bidx[w−1] を準備する;
    (NN, nearest, checked) ← ("none", ∞, 0);
    s ← sketch(query);
    (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
    if checked ≥ K then
        return NN;
    for i = 0 to 2^w − 1 do
        s ← s ⊕ (1⟨⟨bidx[bitcount(i ⊕ (i+1)) − 1]);
        (NN, nearest, checked) ←Search(query, s, NN, nearest, checked);
        if checked ≥ K then
            return NN;
```

**Algorithm 2** SearchByScoreInf

ないにもかかわらず最速である（画像：12.0 ミリ秒，音：18.7 ミリ秒）．

## 5. 結論と今後の課題

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約 10 倍の高速化を実現した．今回，16 ビットスケッチに対して，優先順位付けにハミング距離や $score_\infty$ を用いる場合には，質問のスケッチに近い順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．同様の $score_1$ に対する高速化アルゴリズムは今後の課題である．

データベースの大きさ $n$ と最適なスケッチの幅 $w$ の関係についても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.