情報処理学会論文誌　数理モデル化と応用　Vol.13　No.1　13–22 (Mar. 2020)

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[2]　久保山 哲二[3]　平田 耕一[1]　篠原 武[1]

受付日 2018年11月19日，再受付日 2019年8月27日，
採録日 2019年10月10日

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 k 近傍検索について議論する．スケッチを用いる k 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，$K \geq k$ 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 k 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，バケット法を用いた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 ビットスケッチを用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．

キーワード：スケッチ，局所鋭敏性ハッシュ，バケット法，k 近傍検索

Fast Nearest Neighbor Search with Narrow 16-bit Sketch

Naoya Higuchi[1,a)]　Yasunobu Imamura[2]　Tetsuji Kuboyama[3]　Kouichi Hirata[1]
Takeshi Shinohara[1]

Received: November 19, 2018, Revised: August 27, 2019,
Accepted: October 10, 2019

Abstract: We discuss k nearest neighbor search using sketches, which is a kind of locality sensitive hash (LSH). Search using sketches is processed in two stages. The first stage is to select K solution candidates close in distance between the question and the sketch, where $K \geq k$. In the second stage, k nearest neighbor solutions are selected by performing real distance calculation on K candidates. Conventionally, to ensure high search accuracy, sketches of 32-bit or more have been used. In this paper, we reduce the width of sketches to 16-bit for which efficient data management by bucket is applicable. We propose a search method that enables high speed with the first stage of negligible cost. Searches using 16-bit sketches require a larger number of candidates K to maintain accuracy than using 32-bit sketches. However, by sorting the data objects by their sketch values, memory locality in the second stage search is improved and influence by increasing K is canceled. By using the proposed method, about 10 times speedup can be realized while maintaining search accuracy.

Keywords: sketch, locality sensitive hash, bucket, k nearest neighbor search

[1] 九州工業大学
Kyushu Institute of Technology, Fukuoka 820–8502, Japan
[2] 株式会社 THIRD
THIRD INC., Shinjuku, Tokyo 160–0004, Japan
[3] 学習院大学
Gakushuin University, Toshima, Tokyo 171–8588, Japan
[a)] nac24nh@gmail.com

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．スケッチは，**球面分割**（Ball Partitioning，BP）や一般化超平面分割（Generalized Hyperplane Partitioning，GHP）などの基礎分割関数を用いて空間を任意の回数分割することで作成される．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，質問と候補を比較することにより，解を選択する．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，正確な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を保ちつつ，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 43 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て非効率になるし，そもそも単純に配列を用いたバケット表は超巨大になるので現実的でない．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 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よ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う必要がなくなり，事実上コストを無視できるほど第 1 段階検索を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．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．

我々は，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分割境界との最小距離の表を用意し，それら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最近傍検索の実験により，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

本論文では最近傍検索，つまり，$k = 1$ のときの k 近傍検索について議論する．ここで，最近傍解が複数あった場合でもそのうちの 1 つだけを得られればよいとする．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を $db = \{x_0, \cdots, x_{n-1}\} \subseteq \mathcal{U}$ とする．ここで，$\mathcal{U}$ はデータ空間である．2 個のデータ 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，$K \geq 1$ はユーザが指定するパラメータとする．

(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 $s(x_{i_0}), \ldots, s(x_{i_{K-1}})$ を持つ 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，本論文の実験では，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．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 1 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上げることである．

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．

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

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 $P = \{(p_0, r_0), \ldots, (p_{w-1}, r_{w-1})\}$ を用いて，以下のように BP によるビットを連接したものとして定義される．

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

2.3　質問とスケッチ間の距離下限値

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．質問 q と $BP_{(p_i,r_i)}$ の分割境界との最小距離を $e_i(q, P)$ とすると，

$$e_i(q, P) = |D(p_i, q) - r_i|$$

これを用いて，次のように，$D(q, x)$ の距離下限値 $b_i(q, s_P(x))$ を求めることができる．

$$b_i(q, s_P(x)) = \begin{cases} 0, & \text{if} \quad BP_{(p_i,r_i)}(q) = BP_{(p_i,r_i)}(x), \\ e_i(q, P), & \text{otherwise.} \end{cases}$$

ここで，$b_i(q, s_P(x))$ を求めるために，x については，そのスケッチのみがあればよく，x 自身は不要であることに注意されたい．第 1 段階における解候補選択の基準として距離下限値 $b_i(q, s_P(x))$ 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．

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

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

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

なお，$score_\infty$ や $score_1$ がハミング距離よりも高精度の優先順位であることは，本論文でも示すように，実験的に確認することはできるが，残念ながら，その理由は判明していない．

2.4　スケッチの最適化と量子化球面分割（QBP）

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，量子化球面分割（Quantization Ball Partitioning，QBP）を用いてスケッチの衝突が少なくなるように最適化を行う．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．

スケッチを用いた検索では使用する基礎分割関数によって，検索性能が大きく変化する．BP は中央値付近のデータの多くが球の内側となってしまうため，衝突が多くなってしまうことが考えられる．衝突を少なくするには，ピボットの中心点を空間の隅にとることが有効であるが，空間の隅は次元数に対して指数的に多く存在するので，素朴な方法で効率的に選択するのは困難である．本研究では，こうした問題を解決できる QBP を用いる．QBP は，データベースから任意に点を選び，データ中央値を閾値として空間の最小値もしくは最大値の 2 値に量子化して，ピボットの中心点とする．これにより，データ分布を考慮した効果的なピボット選択が可能となる．

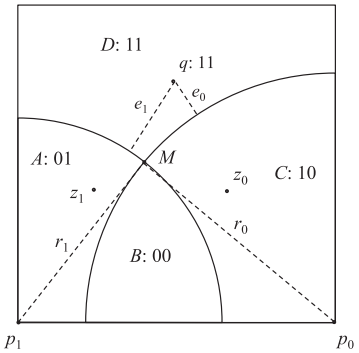
図 1 に QBP を用いて，衝突確率が小さなピボット集合を求める発見的手法 SelectPivotQBP を示す．SelectPivotQBP は，1 ビットスケッチから始めて，幅を順次増やしながら追加するピボットをパラメータ *Trials* で指定された回数乱択により衝突確率が小さくなるように選んでいる．

2.5　QBP によるスケッチと優先順位の例

図 2 に，2 次元ユークリッド平面上における QBP による 2 ビットスケッチと優先順位の例を示す．データの中央値

```
/* dim: 特徴データの次元数, n: データベースのデータ数 */
/* x[0],x[1],...,x[n]: データベースの特徴データ */
/* z[i]（j = 0,...,dim − 1): データ z の第 j 次元の特徴値 */
/* med[j]（j = 0,...,dim − 1): データベースの特徴データの第 j 次元の中央値 */
/* Trials: 各ピボットの探索の試行回数 */
/* MIN,MIN: 特徴値の最大値と最小値 */
1  procedure SelectPivotQBP(p[w][dim], r[w], w)
       /* p[w][dim], r[w]: ピボットの中心と半径のための配列 */
       /* w: スケッチの幅（ビット数） */
       /* eval: ピボット群の評価値 (衝突確率）を求める関数 */
2      for i = 0 to w do
3          best ← ∞;
4          for t = 1 to Trials do
5              c ← random(); z ← x[c];
               /* 2 値量子化（binary quantization） */
6              for j = 0 to dim − 1 do
7                  if z[j] ≤ med[j] then
8                      p[i][j] ← MIN;
9                  else
10                     p[i][j] ← MAX;
11             r[i] ← D(p[i], med);
12             current ← eval(p[0], r[0], ..., p[i], r[i]);
13             if current < best then
14                 best ← current; temp ← p[i]; rtemp ← r[i];
15         p[i] ← temp; r[i] ← rtemp;
```

図 1　2 値量子化を用いたピボット選択アルゴリズム SelectPivotQBP
Fig. 1　Pivot selection algorithm SelectPivotQBP using binary quantization.



部分空間	優先順位		
	Hamming	$score_\infty$	$score_1$
A	1	e_1	e_1
B	2	e_1	e_0+e_1
C	1	e_0	e_0
D	0	0	0

質問 q からの優先順位
$D < A = C < B$（Hamming）
$D < C < A = B$（$score_\infty$）
$D < C < A < B$（$score_1$）

図 2　QBP による 2 ビットスケッチと優先順位
Fig. 2　2-bit sketch by QBP and priorities.

を med とする．データからランダムに選んだ 2 点を z_0, z_1 とすると，med を閾値として 2 値量子化すると，それぞれ p_0, p_1 になる．量子化した点と中央値の距離を r_0, r_1 として，2 個のピボット集合 $P = \{(p_0, r_0), (p_1, r_1)\}$ を用いると，2 ビットスケッチが得られる．中央値との距離を半径とするのは，球面分割によってデータがほぼ半々に分けられるからである．2 個の球面分割によって，平面は，4 個の部分空間 A, B, C, D に分けられる．A または B, C, D の任意の点 x に対して，$s_P(x)$ は，それぞれ 01 または 00, 10, 11 である．図のように，q を両方の球の外側の質問とすると，q からの部分空間の任意の点に対する優先順位は，図の右の表のようになる．ただし，e_0, e_1 は，q と分割境界との最小距離である．ここで，従来の優先順位であるハミング距離では部分空間 A と C が区別できないが，$score_1$ を用いるとすべてが分離できることに注意しよう．また，質問 q を部分空間 D にとる場合でも，e_0, e_1 の大小によって，優先順位が変化することがあることにも注意されたい．

3. 16 ビットスケッチを用いる高速検索

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

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．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，事実上検索コストを無視できるようになる．

図 3 に 16 ビットスケッチのハミング距離を用いた最近傍検索アルゴリズムを示す．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．
- $num[s]$：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，

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

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　16 ビットスケッチの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，距離下限値が質問ごとに異なるため，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．ここでは，以下を仮定する．

- 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．
- $e_i = |D(q, p_i) - r_i|$：質問と位置 i に対応する球面分割境界の最小距離．簡単のため，これらの距離下限値は，次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 のたかだか 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 $score_\infty$ を持つスケッチは，以下のようになる．

表 1　グレイコード生成とスケッチの列挙
Table 1　Gray code generation and sketch enumeration.

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

$score_\infty = 0$ 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次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分かる．

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

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，**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値の順序が決まれば，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

```
/* 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 */
/* num[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 SEARCH(query,s,NN,nearest,checked)
2      for i = f[s] to f[s]+num[s]−1 do
3          checked ← checked+1;
4          if D(query,x[i]) ≤ nearest then
5              (NN,nearest) ← (id[i],D(query,x[i]));
6          if checked ≥ K then
7              return(NN,nearest,checked);
8      return(NN,nearest,checked);
9  function NNSEARCHBYHAMMING(query)
10     (NN,nearest,checked) ← (“none”,∞,0);
11     for i = 0 to 2^w − 1 do
12         s ← sketch(query) ⊕ m[i];
13         (NN,nearest,checked) ←SEARCH(query,s,NN,nearest,checked);
14         if checked ≥ K then
15             return NN;
16     return NN;
```

図 3　16 ビットスケッチのハミング距離による最近傍検索アルゴリズム
Fig. 3　Nearest neighbor search algorithm by Hamming distance of 16 bit sketch.

```
/* 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 と分割境界との最小距離の順位表 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;
```

図 4　距離下限を利用した優先順位 $score_\infty$ による最近傍検索アルゴリズム
Fig. 4　Nearest neighbor search algorithm by priority $score_\infty$.

リズムは非常に単純になる．なぜならば，その並びの次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．このことにより，定数遅延の列挙が可能となっている点に注意しよう．

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，次が成り立つ．

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

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．

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

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，$\ll$ は左論理シフト演算子である．

質問と分割境界との最小距離は，$e_2 \geq e_1 \geq e_0$ 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．図 4 に示したアルゴリズムでは，$bidx[0], \cdots, bidx[w-1]$ は，$0, 1, \cdots, w-1$ の並べ替えで，次を満たしているとする．

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

列挙アルゴリズムは，この順位だけで最小距離そのものは用いていない．また，関数 Search はハミング距離を用いるアルゴリズム（図 3）で示している．

3.3 16 ビットスケッチの距離下限値を用いる検索の高速化（$score_1$）

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分かっている．しかし，$score_\infty$ は距離パターンがたかだか $w+1=17$ 個しかないことを利用して高速化しているが，$score_1$ の値は最大 $2^w = 2^{16}$ 個あるため同様の高速化はできない．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

4.1 実験環境

本論文では，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

いずれの特徴データも周波数強度の対数を 8 ビット符号なし整数で表しており，総和が一定になるように正規化している．文献 [9] での実験においては，我々は，SISAP の Colors データセットも用いていた．しかし，データが約 10 万件しかなく，本研究には適さないので用いていない．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，SelectPivotQBP 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．表 2 に，それぞれのデータに対して，16 ビットスケッチをキーとするバケットの様子（平均要素数（average），空バケット数（empty），要素数 10 以上のバケット数の割合（≥ 10））を示した．多くのバケットは表 2 に示されるように 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．ソートの有無による比較実験では，約 3 倍の高速化が確認できている．

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．画像データベース中のデータ間の距離は，平均 1,650 である．ランダムデータと最近傍の距離は，平均 3,300 である．これに対して，データベース中の動画を低画質化したものから作成した質問（非常によく似たものがある**近質問**，ある程度似たものがある**準近質問**）の最近傍との距離は，300 前後であり，データベース中に似たものがない動画から作成した**遠質問**の最近傍との距離は，平均 600 程度である．そこで，本論文での実験では，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ことにした．たとえば，ノイズレベル 5%の質問 q は，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．つまり，

$$q = 0.95x + 0.05y$$

である．それぞれのノイズレベルについて，1,000 質問作る．質問のレイズレベルごとの最近傍距離の平均を**図 5** に示す．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．これらを近質問，準近質問，遠質問の代わりに用いる．

実験に用いた PC の諸元を**表 3** に示す．

4.2 検索精度と検索時間

画像データおよび音データに対する実験結果をそれぞれ**表 4**，**表 5** に示す．検索精度とは，得られた解が実際の最近傍解の実距離と一致している確率である．ここで，score は検索優先順序，width はスケッチの幅（ビット数），K

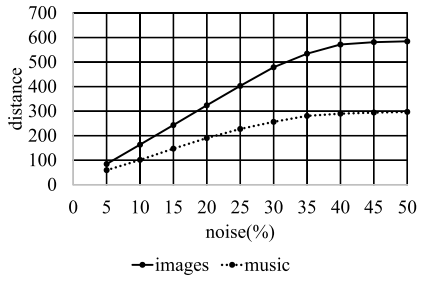


図 5　平均最近傍距離
Fig. 5　Average distance to nearest neighbors.

表 2　16 ビットスケッチのバケット
Table 2　Buckets for 16-bit sketches.

Data	Image	Music
average	105	108
empty	908 (1.5%)	2104 (3.1%)
≥ 10	87%	74%

表 3　実験環境
Table 3　Experiment environment.

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

表 4　画像データに対する検索時間と検索精度
Table 4　Search time and accuracy for images.

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
1st st. time	28.7	4.36	–	35.6	3.23	–	32.0	4.90
total time	29.8	7.16	2.85	36.9	6.06	2.68	33.2	7.76
All	70.2	73.4	73.0	74.3	79.7	79.7	80.2	85.1
very-near	99.8	99.7	99.6	100	100	100	100	100
near	96.9	94.4	94.8	99.1	99.3	99.3	100	100
middle	80.4	80.3	79.3	85.7	88.8	88.8	92.7	94.4
far	46.0	53.9	53.3	52.4	63.5	63.5	63.7	73.2
very-far	28.0	38.5	37.9	34.4	47.2	47.2	44.7	58.0

表 5　音データに対する検索精度
Table 5　Precisions for musics.

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
1st st. time	30.3	4.33	–	36.3	3.23	–	34.4	4.63
total time	31.7	7.89	3.58	38.0	6.82	3.38	36.0	8.30
All	65.5	66.8	66.0	63.6	75.1	73.4	72.3	77.8
very-near	99.7	98.7	98.2	99.5	99.8	99.7	100	100
near	93.3	89.1	89.3	92.2	95.3	94.8	97.2	97.2
middle	70.6	68.2	67.6	67.4	80.3	78.5	79.8	83.0
far	40.2	44.2	43.7	36.9	55.6	53.1	51.2	61.5
very-far	23.9	33.7	31.4	21.9	44.5	41.2	33.2	47.4

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 6 千個に対する割合（%），100%は列挙を用いない場合），time は 1 質問あたりの検索時間（ミリ秒）を表す（1st st. は第 1 段階の検索時間）．列挙を用いる場合は，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．

また，比較のために，画像データに対する全探索を実行したところ，1 質問あたり 550 ミリ秒の検索時間を要することが分かった．最近傍探索時には，質問とデータ間の距離計算をそれまでに見つかった暫定解との距離を超える段階で打ち切ることで高速化することができ，探索時間を 280 ミリ秒まで短縮できる．したがって，精度 70%程度であれば，従来法の 32 ビットスケッチでハミング距離を用いた検索（29.8 ミリ秒）でも，全探索より 10 倍あるいは 20 倍程度高速であることが分かる．さらに，提案手法の 16 ビットスケッチを用いると，優先順位付けに $score_\infty$ を用いた検索（2.68 ミリ秒）は，全探索より 100 倍以上あるいは 200 倍以上高速であり，しかも高精度（79.7%）であることが分かる．

表 6　画像データに対する検索精度 90%以上の実験結果
Table 6　Experimental results with precisions of 90% or more for images.

score	Hamming			$score_\infty$			$score_1$	
width	32	16		32	16		32	16
K(%)	2.0	6.5		1.5	5.0		1.0	2.5
sketches(%)	–	100	5.8	–	100	4.1	–	100
time(ms)	139	22.0	17.5	106	16.8	12.8	107	12.0
All	91.5	90.2	90.2	90.1	92.0	90.6	93.8	91.4
very-near	100	100	100	100	100	100	100	100
near	100	99.7	99.7	99.9	100	99.9	100	100
middle	97.4	95.4	95.4	97.3	97.3	96.6	99.1	97.3
far	84.4	82.4	82.4	81.8	85.8	83.4	88.7	85.3
very-far	76.0	73.8	73.8	71.6	77.0	73.4	81.5	74.7

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分かる．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 ビットスケッチの列挙による高速化の有効性も確認できる．列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分かる．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 倍程度の高速化にとどまっているが，最高精度（画像：85.1%，音：77.8%）を達成している．

4.3　高精度検索における実験結果

従来の手法では，R-Tree など他の手法の検索速度より高速にするためには，K を大きくして精度を高くすることができなかったが，提案手法は，より高い精度が要求される場合であっても，高速な検索が期待できる．精度が 90%を超えるより大きな K で比較した結果を**表 6**，**表 7** に示す．精度を 90%以上に保つためには，画像データにおいては，

表 7　音データに対する検索精度 90%以上の実験結果
Table 7　Experimental results with precisions of 90% or more for musics.

score	Hamming			$score_\infty$			$score_1$	
width	32	16		32	16		32	16
K(%)	3.5	10		3.5	8.0		1.5	4.0
sketches(%)	–	100	7.4	–	100	5.5	–	100
time(ms)	227	37.4	32.7	188	30.1	25.1	160	18.7
All	90.8	90.4	90.1	90.3	92.2	90.6	92.0	90.9
very-near	100	100	100	100	100	99.8	100	100
near	99.2	98.8	98.6	98.6	99.1	98.9	99.4	99.2
middle	93.9	92.9	92.5	92.9	94.7	93.4	95.2	94.1
far	85.2	84.2	83.5	84.3	86.8	85.7	86.9	84.6
very-far	76.0	76.3	76.2	75.8	80.5	75.9	78.6	76.6

従来の 32 ビットスケッチを用いた検索速度（1 質問あたり 139 ミリ秒，$score_\infty$ を用いた 106 ミリ秒，$score_1$ を用いて 107 ミリ秒）は，素朴な全探索（550 ミリ秒）より 4 倍程度，あるいは距離計算の打ち切りを行う全探索（280 ミリ秒）より 2 倍程度の高速化にしかならない．16 ビットスケッチでは 32 ビットのときより，K は大きくする必要があるが，8 倍から 10 倍高速化でき，画像データに対しては，全探索より 40 倍または 20 倍程度の高速化が達成できる．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，列挙による高速化を用いないにもかかわらず最速である．したがって，$score_1$ のための効率的な列挙が可能となれば，さらなる高速化が期待できる．

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．

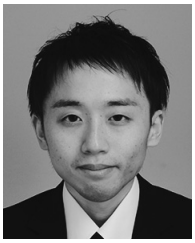
データベースの大きさ n と最適なスケッチの幅 w の関係についてもさらに調査する必要がある．本論文では，n は数百万と仮定したが，より大きなデータベースに対しては，w を 16 より大きくした方がよいかもしれない．特徴データの次元数の影響についても調査する必要がある．

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 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持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．

本研究に関連する最重要課題は，距離下限値を用いる優先順位 $score_\infty$ や $score_1$ がハミング距離よりも高精度である理由について解明することである．また，$score_2$ は，文献 [9] で $score_1$ とほぼ同精度であることが報告されているので，本論文では用いなかったが，たとえば，$score_{1.5}$ などとともに検討する必要があるかもしれない．

参考文献

[1] Müller, A. and Shinohara, T.: Efficient similarity search by reducing I/O with compressed sketches, *Proc. SISAP'09*, pp.30–38 (2009).
[2] Mic, V., Novak, D. and Zezula, P.: Speeding up similarity search by sketches, *Proc. SISAP 2016*, pp.250–258 (2016).
[3] Dong, W., Charikar, M. and Li, K.: Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces, *Proc. ACM SIGIR'08*, pp.123–130 (2008).
[4] Mic, V., Novak, D. and Zezula, P.: Improving sketches for similarity search, *Proc. MEMICS'15*, pp.45–57 (2015).
[5] Wang, Z., Dong, W., Josephson, W., Lv, Q., Charikar, M. and Li, K.: Sizing sketches: A rank-based analysis for similarity search, *Proc. ACM SIGMETRICS'07*, pp.157–168 (2007).
[6] Yianilos, P.: Data structures and algorithms for nearest neighbor search in general metric spaces, *Proc. SODA 1993*, pp.311–321, ACM Press (1993).
[7] Guttman, A.: R-Trees: A dynamic index structure for spatial searching, *Proc. SIGMOD'84*, Yormark, B. (Ed.), pp.47–57, ACM Press (1984).
[8] Ciaccia, P., Patella, M. and Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces, *Proc. VLBD'97*, pp.426–435 (1997).
[9] Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K. and Shinohara, T.: Nearest Neighbor Search using Sketches as Quantized Images of Dimension Reduction, *Proc. ICPRAM 2018* (2018).
[10] Shinohara, T. and Ishizaka, H.: On dimension reduction mappings for approximate retrieval of multi-dimensional data, *Progress of Discovery Science, LNCS 2281*, pp.89–94, Springer Berlin/Heidelberg (2002).
[11] Imamura, Y., Higuchi, N., Kuboyama, T., Hirata, K. and Shinohara, T.: Pivot selection for dimension reduction using annealing by increasing resampling, *Proc. LWDA 2017* (2017).



樋口 直哉　（学生会員）

1992 年生．2011 年九州工業大学情報工学部卒業．2017 年同大学大学院博士前期課程修了．

今村 安伸

1982 年生．2003 年久留米工業高等専門学校卒業．2006 年同専攻科修了．2008 年九州工業大学大学院博士前期課程修了．2018 年同博士後期課程単位取得退学．博士（情報工学）．2018 年（株）THIRD 入社．チーフ・サイエンティフィック・オフィサー．

久保山 哲二　（正会員）

1992 年九州大学工学部卒業．1994 年同大学大学院修士課程修了．博士（工学）．1997 年東京大学助手．2008 年学習院大学准教授．2013 年同大学教授．離散データ構造の近似パターンマッチング・機械学習・データマイニング等の研究に従事．

平田 耕一　（正会員）

1967 年生．1990 年九州大学理学部数学科卒業．1992 年同大学大学院修士課程修了．1995 年同博士後期課程修了．博士（理学）．1995 年九州工業大学助手．2000 年同大学助教授．2012 年同大学教授．データマイニング，帰納論理プログラミング，計算学習理論，計算論理学，アブダクションの研究に従事．

篠原 武　（正会員）

1955 年生．1980 年京都大学理学部卒業．1982 年九州大学大学院修士課程修了．理学博士．1982 年九州大学助手．1987 年九州工業大学助教授．1994 年同大学教授．計算論的学習理論，帰納推論，情報検索の研究に従事．