狭幅スケッチのためのピボット選択

樋口 直哉[1,a)]　今村 安伸[2,b)]　篠原 武[1,c)]　久保山 哲二[3,d)]　平田 耕一[1,e)]

概要：スケッチは高次元データをコンパクトなビット列で表現したもので一種の LSH である．狭幅スケッチを用いた高次元データの高速最近傍検索手法を利用して，スケッチをさらに高性能化するためのピボット選択法を提案する．スケッチを用いた最近傍検索はあくまでも近似であり，その近似精度を高めることが最重要課題である．スケッチの近似精度そのものを測定することはコストが大きいので，従来のスケッチの設計には，衝突率やエントロピーなどの別の指標を用いて最適化を行ってきた．しかし，そうした間接的な指標での最適化には限界がある．比較的に短い狭幅スケッチを用いる非常に高速な検索技法が確立されたので，それを用いることにより，スケッチの近似精度に直結する指標による最適化が可能になった．類似検索自身を用いたピボット群の評価による最適化を提案する．提案手法を用いることにより約 690 万件 64 次元の画像特徴データベースに対して行った実験の結果，衝突率による最適化によるものより 90% の精度を保ちつつ 2 倍近く高速な最近傍検索を行うことが可能になる．

Pivot Selection for Narrow Sketches

NAOYA HIGUCHI[1,a)]　YASUNOBU IMAMURA[2,b)]　TAKESHI SHINOHARA[1,c)]　TETSUJI KUBOYAMA[3,d)]
KOUICHI HIRATA[1,e)]

1. はじめに

スケッチ [2], [12], [13], [14], [16] は，多次元データを表す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）の一種として，多次元空間における効率的**類似検索**を実現するために利用することができる．本論文では，検索対象のデータは距離空間内の点として与えられていると仮定し，類似検索の手法としては最近傍検索，すなわち，質問として与えられた点に最も近いデータを取得する手法を取り扱う．スケッチのデータに対するビットの割り当て方は，**球面分割**（Ball Partitioning，BP）を用いる．BP は，空間内の球を用いて，データが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，スケッチを用いたスコアによる優先順位付け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，階層的空間索引 R-tree [3] や M-tree [1] 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．

スケッチを用いた最近傍検索の検索精度・速度の性能を向上をするためには，その性能そのものを測定するための計算コストが大きいため，それを直接目的関数として最適化や学習を行うことは困難であった．そのため，衝突確率などの間接的な目的関数を用いて，それを最小化する最適化によってある程度検索性能が高くなるスケッチのピボット選択を行ってきた．ここで，**衝突**とは，異なるデータに

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

対して同じスケッチが割り当てられることをいう．われわれは，データの分布特性を利用した 2 値量子化法（QBP）を提案し，効果的に衝突確率を下げることにより検索性能を向上できることを確認した．さらに焼きなまし法の一種である手法（AIR）を提案し，さらに低い衝突確率のスケッチを求めることに成功した．しかし，衝突確率をある程度以上に下げても，必ずしも性能は向上しないことも確認した [10].

われわれは，第 1 段階で用いるスケッチの優先順位付けとして，$score_\infty$ や $score_1$ などを提案し，従来のハミング距離よりも優れていることを示した [4]．さらに 16 ビットなどの比較的にビット数が少ない狭幅スケッチを用いた高速検索技法を提案した [5], [6], [8]．それは，スケッチ自身をキーとするバケット法による効率的データ管理と優先順にスケッチを列挙する効率的アルゴリズムに基づいている．これにより，従来のスケッチを用いる検索よりも 50 倍から 100 倍以上の高速化が達成でき，空間索引やスケッチを用いない素朴な方法より 200 倍から 400 倍高速である．

さて，スケッチの高性能化のために，論文 [5], [6], [8] で報告した検索実験結果に基づいて，1 万件の質問例を対象とする検索精度そのものを目的関数として用いて最適化を行うことを考えてみよう．検索対象のデータは，画像データの 2 次元周波数強度として抽出した 64 次元の特徴データ約 690 万件である．従来の 32 ビットスケッチを用いる手法では，80%程度の精度での最近傍検索に要する時間は，1 質問当り 100 ミリ秒程度であるのに対し，狭福スケッチ（16 ビット）による高速化技法を用いると 2 ミリ秒程度である．従来の検索手法を用いて実際に検索を行って検索精度を測定すると，1 回の測定だけでも 100 ミリ秒× 1 万＝1,000 秒必要である．局所探索を用いて最適化を行うと最低でも 1 万回程度の試行を繰り返す必要があり，そうすると数か月かかる．以上のように，従来手法のような速さの検索方法では，実際の検索精度を直接目的関数とした最適化を行うことは，誰も試みることすらなかったと思われる．ところが，狭幅スケッチを用いる非常に高速な検索が可能となったので，素朴な方法でも数日で直接検索精度を用いた最適化を行うことができるようになった．

本研究では，狭幅スケッチの高速検索技法を活用して，検索精度計算の高速化を行う．まず，質問に対する優先順のスケッチの列挙と，事前計算した質問集合に対する正解情報を利用すれば，実距離計算を行わずに検索精度を求めることを可能とし，10 倍程度の高速化を達成する．さらに，検索精度評価に用いるデータベースを 16 分の 1 に縮小し 2〜3 倍の高速化をする手法を提案する．このことにより，1 時間程度で 16 ビットのスケッチのためのピボット群を選択することができるようになった．これにより，従来の QBP などでは 85%程度止まりであった検索精度を 90%に高めることができる．また，従来は，90%の精度を達成するためにはデータベースの 2%強の解候補数が必要であったが，これをほぼ半分にすることが可能となり，2 倍程度高速に検索できるようになった．

2. 準備

2.1 スケッチを用いる最近傍検索

本論文では，最近傍検索について議論する．最近傍解が複数あった場合でもそのうちの一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（データ ID）で索引付けされているとする．つまり，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，*sketch* 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) 第 1 段階 （スケッチ間のスコアによるフィルタリング）: 質問 q のスケッチ $sketch(q)$ に近いスコアのスケッチ $sketch(x_{i_0}), \ldots, sketch(x_{i_{K-1}})$ をもつ K 個の解候補 $x_{i_0}, \ldots, x_{i_{K-1}}$ を選ぶ．

(2) 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 q に最も近いデータ x_{ans} を選ぶ．

第 1 段階検索において用いるスケッチのスコアは，従来用いられてきたハミング距離よりも高い検索精度が得られる $score_1$ [4] を用いる．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．最近傍が複数存在する場合には，そのうちの一つが得られれば正解と考えるので，検索の**再現率**は，精度より若干低くなる．あきらかに，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，できるだけ小さい K で目的精度を達成することが，最も重要なスケッチを用いる検索の課題の一つである．

2.2 球面分割に基づくスケッチと距離下限を用いた優先順位付け

中心点と半径のペア (p,r) を**ピボット**と呼ぶ．ピボット (p,r) による球面分割 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 $sketch_P$ 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．

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

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 [4] を用いる．これは，球面分割によるスケッチは次元縮小射影 Simple-Map [15] 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．ピボット集合を $P=\{(p_0,r_0),...,(p_{w-1},r_{w-1})\}$ 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，つぎのように，$D(q,x)$ の距離下限値 $b_i(q,s_P(x))$ を求めることができる．

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

距離下限値の総和である以下の $score_1$ を用いると従来のハミング距離より高精度の検索が行える．

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

2.3 衝突率を用いたピボット選択

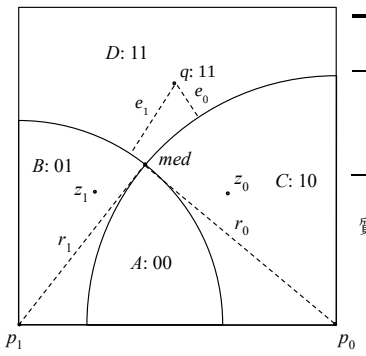
スケッチを用いた検索の精度を高めるために，衝突確率が小さくなるようにピボット集合 P を選ぶ．異なるデータ x と y に対して，それらのスケッチが一致するとき，すなわち，

$$sketch_P(x)=sketch_P(y)$$

となるとき，**衝突**が発生するという．量子化球面分割（Quantization Ball Partitioning, QBP）を用いてスケッチの衝突がある程度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．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．Algorithm 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．

```
/* 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[j]（j = 0,...,dim−1): データ z の第 j 次元の特徴値
   med[j]（j = 0,...,dim−1): 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;
```

Algorithm 1: SELECTPIVOTQBP



部分空間	スコア	
	ハミング	$score_1$
A: 00	2	e_0+e_1
B: 01	1	e_1
C: 10	1	e_0
D: 11	0	0

質問 q からの優先順位
$D<B=C<A$ （ハミング）
$D<C<B<A$ （$score_1$）

図 1 QBP による 2 ビットスケッチと優先順位
Fig. 1 2-bit sketch by QBP and priorities

2.4 狭幅スケッチを用いる高速検索

本節では，より幅の狭い 16 ビットのスケッチを用いる手法を紹介する．データベースの大きさは，数百万であると仮定する．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．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のスコアが小さいスケッチは，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，スコアが小さ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第 1 段階検索を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平均 100 個以上存在するので，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．

Algorithm 2 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．列挙の詳細については，論文 [5], [7] を参照されたい．データ

ベースを $x[0],\ldots,x[n-1]$ とする（配列 x に格納されていると考える）．$sketch(x)$ は，データ x に対するスケッチである．このとき，データベースをスケッチをキーとするバケットを用意する．ここでは，つぎの条件を満たす 3 個の配列 num, id, f で表されているとする．

- $num[s]$ = スケッチが s であるデータベース中のデータの個数（$s = 0,\ldots,2^{w-1}$）．
- $id[j]$ = データベースのデータをスケッチ順に並べたときの j 番目のデータ ID （配列 x における index）$(j = 0,...,n-1)$．
- $f[s]$ = スケッチが s であるデータのスケッチ順での先頭位置 （$s = 0,\ldots,2^{w-1}$）．

このとき，スケッチが s であるデータは，

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

である．また，スケッチが s であるデータ ID は，

$$id[f[s]],\ldots,id[f[s]+num[s]-1]$$

である．

```
/* K : 第 1 段階での取得候補数 = 第 2 段階での実距離計算回数 */
1  function SEARCH(q, s, NN, nearest, checked)
2      for i = f[s] to f[s]+num[s]-1 do
3          checked ← checked+1;
4          if D(q, x[id[i]]) ≤ nearest then
5              (NN, nearest) ← (id[i], D(q, x[id[i]]));
6          if checked ≥ K then return(NN, nearest, checked);
7      return(NN, nearest, checked);
8  function SEARCHBYSCORE1(q)
9      q に対する score_1 順のスケッチの列挙を s[0], s[1], ... とする ;
10     (j, NN, nearest, checked) ← (0, "none", ∞, 0);
11     while checked ≤ K do
12         (NN, nearest, checked) ←SEARCH(q, s[j], NN, nearest, checked);
13         j ← j+1;
14     return NN;
```

Algorithm 2: SEARCHBYSCORE1

3. 検索精度そのものを目的関数とする最適化

3.1 狭幅スケッチによる高速検索を利用した検索精度の評価アルゴリズム

ここで，事前に求めておいた正解情報と狭幅スケッチの列挙を利用することにより，実距離計算を行わずに検索精度を評価することができることを示す．スケッチの列挙を用いたアルゴリズム（SEARCHBYSCORE1）では，質問のスケッチから始めて，その $score_1$ のスコアが小さい順にスケッチを列挙することに基いている．そこでは，列挙されたスケッチ s に対して，そのスケッチをもつデータと質問

```
/* Q = {Q[0], ..., Q[|Q|-1]}: 質問集合.
   Ans[0], ..., Ans[|Q|-1]: 正解（最近傍データの ID）情報.
   K : 第 1 段階での取得候補数 */
1  function CHECK(ans, s, checked)
2      for i = f[s] to f[s]+num[s]-1 do
3          checked ← checked+1;
4          if ans = id[i] then return(true, checked);
5          if checked ≥ K then return(false, checked);
6      return(false, checked);
7  function PRECISIONBYSCORE1(P, Q, db)
8      ピボット集合 P によるスケッチをキーとするデータベース
       db のためのバケットを用意する ;
9      success ← 0;
10     for i = 0 to |Q|-1 do
11         Q[i] に対する score_1 順のスケッチの列挙を
12         s[0], s[1], ... とする ;
13         (j, found, checked) ← (0, false, 0);
14         while checked < K and not found do
15             (found, checked) ←CHECK(Ans[i], s[j], checked);
16             if found then success ← success+1;
17             j ← j+1;
18     return success/|Q|;
```

Algorithm 3: PRECISIONBYSCORE1

の実距離計算を行っている．しかし，質問に対する正解情報，最近傍のデータ ID を事前に求めておけば，検索精度を求めるためには，K 個の候補中にそのデータ ID が出現するかしないかを確認すれば十分であり，実距離計算を省くことができる．Algorithm 3 に高速化した精度評価アルゴリズム（PRECISIONBYSCORE1）を示す．Algotirhm 2 では，実距離計算を行う際のメモリ局所性を向上するために，特徴データをスケッチ順にソートしたものを用いている．しかし，精度評価においては，実距離計算は不要であるため，特徴データのソートは不要である．実際，ピボット探索においては，ピボットを変更するたびにバケット表を作成する必要があり，データのソートを省いた方がバケットデータベース再構築のコスト増大を防ぐ効果もある．このため，Algorithm 3 と Algotirhm 2 では，実際には使用するバケットデータベースが若干異なっている点に注意されたい．

3.2 縮小データベースを用いた検索精度評価の高速化

さらに検索精度の計算コストを小さくするために，最終的に目的とするデータベースをそのまま評価に用いないで，データを一定間隔で間引いた縮小データベースを使用することを提案する．

3.3 局所探索による最適化を用いたピボット選択

スケッチの幅は 16 ビットとし，QBP に基づいたものを

仮定する．ピボットの個々の中心点は 64 個の MIX（0）または MAX(255) からなる．ピボットの半径は中央値との距離で求めるのでパラメータとする必要はない．したがって，16 ビットスケッチのためのピボット集合は，16 × 64=1,024 個のパラメータで定められる．局所探索により検索精度が高くなるピボット集合を探索する．探索のときのあるピボット集合の近傍は，16 個のピボットの 1 個だけの 64 個の MIN または MAX のうち一定の個数以下のものを反転（MIN と MAX を交換）したものとする．最適化の初期の段階は反転数を 4 個とし，徐々に減らして，最終段階では 1 個となるようにする．

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 690 万件の 64 次元の画像データを用いた実験結果を示す．

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 合計 1 万質問作る（質問集合 1）．最終的に得られたピボット集合を用いて検索精度を評価するためには，最適化時に検索精度を評価するために用いる質問集合とは別の 1 万質問集合を用意した（質問集合 2）．

ピボット選択時の検索精度を評価する際には，スケッチを用いた第 1 段階検索における候補数選択数 K をデータベースサイズ（データ件数）の 1%とし，最終的に得られたピボット集合を用いた検索精度も同じ候補数比率を用いる．

実験に用いた PC の諸元を表 1 に示す．

表 1 実験環境

CPU	Intel(R) Core(TM) i7-7700 3.6 GHz
memory	16GBytes

4.1 衝突率の最小化によるピボット選択

まず，本論文での提案手法との比較のために，SELECTPIVOTQBP によるピボット選択の結果を表 2 に示す．ここでは，ピボット選択には質問集合 1 は用いていないので，検索精度は質問集合 1 に対するもののみを示した．試行回数は，アルゴリズム中のパラメータ *Trials* で指定する，各ピボットを選択するときの反復回数である．衝突確率は，データベースから乱択した 10,000 個のデータ間の衝突確率を用いた．この表により，試行回数を増やせば，衝突確率を下げることができ，検索精度を向上することができる様子がうかがえる．しかし，これ以上反復回数を増やしても，ほとんど衝突確率を下げることができなくなる．また，焼きなまし法の一種である AIR（Annealing by Increasing Resampling） [9], [11] を用いれば，さらに低い衝突確率を持ったピボット集合を得ることが可能であるが，検索精度は改善できないことが報告されている [10].

表 2 最小衝突法によるピボット選択

試行回数	衝突確率	検索精度
1	7.0×10^{-4}	76.2%
10	1.0×10^{-4}	84.0%
100	6.5×10^{-5}	84.7%
1000	4.9×10^{-5}	85.2%

4.2 実距離計算の省略による精度評価コストの低減

局所探索において実距離計算による照合を省略しないとき（試行回数 100 回）と省略したとき（試行回数 1,000 回）の実験結果を表 3 に示す．試行回数は局所探索における反復回数，time は計算時間（秒），training はピボット探索時の評価に用いた質問集合 1 に対する検索精度，test は質問集合 2 に対する検索精度である．この結果より，実距離計算による照合を省略することにより，計算時間が 10 分の 1 以下に短縮できることがわかる．

表 3 実距離計算による照合の省略効果

試行回数	照合	time(sec)	training	test
100	有	629	86.7%	85.6%
1,000	無	597	88.8%	87.0%

4.3 縮小データベースの使用

表 4 に縮小データベースを使用したときの結果を示す．試行回数 1,000 回の局所探索を行った．局所探索で最終的に得られたピボット集合による評価用データベースに対する最終検索精度を final に示した．縮小データベースを用いる場合は，縮小しないときの最近傍が間引かれることにより，最近傍解との距離が大きくなるため，最終検索精度は低くなる傾向にある．しかし，その縮小データベースに対する検索精度を高くすると，元のデータベースに対する検索精度も高くなることが期待できる．実験結果からは，縮小データベースを用いても，訓練例（training）とテスト例（test）の両方で検索精度が向上していることがわかる．縮小データベースを用いない場合には，test の方が若干精度が低くなっている．縮小データベースを用いる方が test に対する精度が下がらないため，最適化の結果の汎化性能

は高いと思われる．16 分の 1 以下に縮小してもあまり評価コストはさがらないので，以下の実験では 16 分の 1 のデータベースを使用することにする．

表 4 縮小データベースの利用

評価用データベース	time(sec)	final	training	test
全体	613	88.5%	88.5%	87.2%
4 分の 1	297	84.1%	87.8%	87.1%
16 分の 1	236	76.7%	87.8%	87.5%
64 分の 1	229	67.7%	87.5%	87.6%

4.4 局所探索により最適化

今回のピボット探索ではパラメータ数が 16 × 64 = 1,024 であるので，局所探索では試行回数がある程度以上になると局所解に収束する．実際に実験で確認したところ 1 万 5 千回までに収束するようである．表 5 に収束するまで試行を繰り返したときの 5 回の実験結果を示す．これにより，最小衝突法では K = 1%での検索精度は 85%程度であったものをほぼ 90%に高めることが可能となった．精度 90% を達成する K は 2%強であったので，提案手法で求めたピボットでは K=1%で精度 90%となるので，ほぼ 2 倍の高速化が達成できる．

表 5 局所探索の収束

試行	converge	time(sec)	final	training	test
1	12,000	2,777	79.4%	89.0%	88.9%
2	12,000	2,861	79.1%	89.1%	88.8%
3	12,742	3,474	79.7%	88.6%	88.9%
4	13,390	3,069	80.4%	89.4%	88.5%
5	19,837	4,523	80.6%	89.6%	88.8%
平均	13,994	3,341	79.8%	89.1%	88.8%

5. 結論と今後の課題

本論文で提案した手法により効果的に検索性能が高いスケッチを作成することが可能となった．最適化には局所探索を用いたが，さらに高性能のスケッチを作成するために，焼きなまし法（AIR) を適用することを計画している．また，検索時の評価指標については，検索精度そのものを用いたが，ある質問について第 1 段階の検索で選択する候補の中に正解が含まれるかどうかで，検索精度が大きく変動するという問題があるので，それを緩和する手法についても検討を開始している．こうした点を解決してより高性能のスケッチの設計を行うことが最も重要な今後の課題である．

参考文献

[1] 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.

[2] 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.

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

[4] 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'18*, pp. 356–363, 2018.

[5] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Fast filtering for nearest neighbor search by sketch enumeration without using matching. In *Proc. AI'19, LNAI 11919, Springer*, pp. 240–252, 2019.

[6] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Fast nearest neighbor search with narrow 16-bit sketch. In *Proc. ICPRAM'19*, pp. 540–547, 2019.

[7] 樋口直哉, 今村安伸, 久保山哲二, 平田耕一, 篠原武. 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. 火の国情報シンポジウム 2019.

[8] 樋口直哉, 今村安伸, 久保山哲二, 平田耕一, 篠原武. 狭い 16 ビットのスケッチを用いた高速最近傍検索. 情報処理学会論文誌「数理モデル化と応用」（採録決定）.

[9] N. Higuchi, Y. Imamura, T. Kuboyama, K. Hirata, and T. Shinohara. Annealing by increasing resampling. In *In Revised Selected Papers, ICPRAM'19, LNCS 11996, Springer*, pp. 71–92, 2020.

[10] Y. Imamura, N. Higuchi, T. Kuboyama, K. Hirata, and T. Shinohara. Pivot selection for dimension reduction using annealing by increasing resampling. In *Proc. LWDA'17*, pp. 15–24, 2017.

[11] Y. Imamura, N. Higuchi, T. Shinohara, T. Kuboyama, and K. Hirata. Annealing by increasing resampling in the unified view of simulated annealing. In *Proc. ICPRAM'19*, pp. 173–180, 2019.

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

[13] V. Mic, D. Novak, and P. Zezula. Speeding up similarity search by sketches. In *Proc. SISAP'16*, pp. 250–258, 2016.

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

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

[16] 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.