バケット距離に基づく近似最近傍探索

佐藤　智一†　　武藤　大志††　　岩村　雅一††　　黄瀬　浩一††

† 大阪府立大学工学部　〒 599-8531 大阪府堺市中区学園町 1-1
†† 大阪府立大学大学院工学研究科　〒 599-8531 大阪府堺市中区学園町 1-1
E-mail: {sato,mutoh}@m.cs.osakafu-u.ac.jp, {masa,kise}@cs.osakafu-u.ac.jp, tsuji@m.cs.osakafu-u.ac.jp

あらまし　本稿では，近似最近傍探索問題において，既存手法と比較して同精度における処理時間を削減した．近似最近傍探索とは，与えられたクエリの近傍にある確率の高い点を抽出し，その中で距離計算を行い最近傍点を探索するものである．本稿ではハッシュを用いた手法を扱う．Locality Sensitive Hashing（LSH）などの従来の手法では，クエリと同じバケットに入った点のみを距離計算の対象とする．しかし，LSH では抽出の段階で近傍点を漏らしやすい．この問題を改善する Multi-Valued Hashing（MVH）が考案された．しかし，MVH はクエリを中心とする超球に近い形で近傍点を抽出できるが，この処理自体に時間がかかる．そこで，本稿では近傍点の抽出を MVH に近い形で高速に行う手法を提案する．

キーワード　近似最近傍探索，Locality Sensitive Hashing，Multi-Valued Hashing，ビン，バケット，バケット距離

Approximate Nearest Neighbor Search Based on Bucket Distance

Tomokazu SATO†, Tomoyuki MUTOH††, Masakazu IWAMURA††, and Koichi KISE††

† Faculty of Engineering, Osaka Prefecture University
1-1 Gakuencho, Naka, Sakai, Osaka, 599-8531 Japan
†† Graduate School of Engineering, Osaka Prefecture University
1-1 Gakuencho, Naka, Sakai, Osaka, 599-8531 Japan
E-mail: {sato,mutoh}@m.cs.osakafu-u.ac.jp, {masa,kise}@cs.osakafu-u.ac.jp, tsuji@m.cs.osakafu-u.ac.jp

1. 前　書　き

近年の計算機性能の向上と画像照合技術の発展によって，大規模データベースを用いた物体認識の研究が盛んに行われている．例えば，野口らはデータベースからクエリ画像と一致する画像を高速に検索する手法を提案している [1]．この手法では，予め画像から PCA-SIFT [2] の特徴ベクトルを抽出しておき，クエリ画像から得られる特徴ベクトルに類似するもの，つまり最近傍点を探索する必要がある（以後，データベース内のベクトルを点，探索質問ベクトルをクエリと呼ぶ）．最近傍点を探索するには，データベース内の全ての点と距離計算をしてクエリから最も近いものを探索すればよいが，この方法では画像 1000 枚のデータベースを検索のに約 315 秒を要する．これに対して野口らの手法では，画像 100 万枚のデータベースから僅か約 0.06 秒で検索を行うことが可能となる．

この様な高速な検索は，近似最近傍探索によって実現される．近似最近傍探索とは，上記のような最近傍探索問題において，予めクエリの近傍にある可能性の高い点を抽出し，その中で距離計算を行い最近傍点を探索するものである．これによって，距離計算による処理時間を大幅に削減することができる．しかし，近傍点の抽出の段階で最近傍点を漏らしてしまうと，探索は失敗する．近似最近傍探索は，精度（真の最近傍点を得られる確率）を犠牲にすることで，高速な探索を実現している．最近傍探索問題において，処理速度と精度のどちらが要求されるかは扱うシステムによるが，処理速度を要求するシステムにおいては近似最近傍探索は非常に有効であると言える．

本稿では，上記のような探索問題に有効な近似最近傍探索に着目する．近似最近傍探索には様々な手法が考案されているが，ここではハッシュを用いたものを扱う．

ハッシュを用いる近似最近傍探索の代表的な手法として，Locality Sensitive Hashing (LSH) [3], [4] がある．LSH はデータ空間を線形分割し，クエリと同じ領域に入った点を抽出し，距離計算を行う．しかし，LSH は最近傍点を漏らしやすく，近傍点の抽出効率が悪い．

そこで，この問題を緩和させた手法が，Multi-Valued Hashing (MVH) [5] である．MVH は，分割された空間ごとに，ク

エリからの距離に応じて各点に重みを与え，各点のクエリからの距離の概算値を算出し，その概算値が小さいものを計算対象として抽出する．これにより，MVH はクエリを中心とする超球に近い形で，近傍点を抽出することができ，抽出点を少なくしても最近傍点を漏らすことが少ない．しかし，MVH には点の抽出精度は良いが，近傍点の抽出速度が遅いという問題があり，全体として処理を高速化することができない．その原因は，距離の概算値の算出方法にある．MVH において，距離の概算値を求めるには，非常に大きな領域に含まれる点に対して，何度も距離重みの加算処理が必要になる．これによって，加算処理による計算コストが非常に大きなものとなる．

本稿では，MVH の近傍点の抽出処理の遅延を招く「多数のデータに対するアクセス」を排除し,「分割された領域間の距離」を定義することによって，効率よく距離の概算値の小さい点を探索する方法を示す．領域間に距離を与えることで，データの増加に伴う抽出処理の遅延がなくなり，高速に近傍点を探索することができる．実験の結果，提案手法は精度 95%〜96%で，処理時間を約 63%に，精度 30%〜31%では，約 6.6%に抑えることができた．

2. 近似最近傍探索

本稿では，検索問題の処理速度向上に有効な近似最近傍探索の性能向上を目的としている．ここでの性能は，処理時間（クエリ（点）を与えてから近似最近傍点が得られるまでの時間），精度 (真の最近傍点が得られる割合)，メモリ使用量の 3 点で評価する．近似最近傍探索には，2 段階の処理がある．まず，クエリの近傍にある可能性の高い点を抽出し (図 1(a))，距離計算を行いその中から最近傍点を探索する (図 1(b))．抽出の処理で真の最近傍点を漏らせば，探索は失敗する．これは，図 1(a) の矩形領域の中に最近傍点が含まれなかった場合を示す．つまり，精度と処理時間には，抽出する点を少なくするほど，探索の失敗は起こりやすいが，処理時間は短くなるというトレード・オフの関係がある．ここで求められるのは，抽出の処理で真の最近傍点を漏らさずに，抽出する点の数を少なくすることである．これを実現するには，クエリを中心とする超球状に点を抽出することが理想的である．

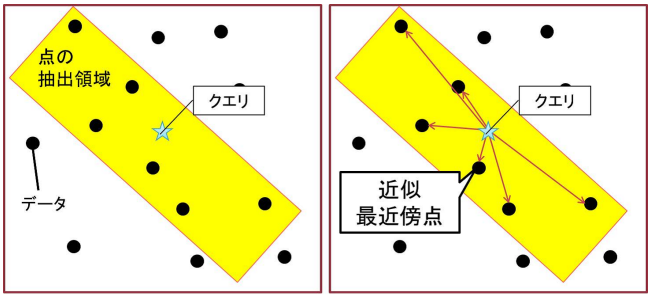
本研究では，近似最近傍探索手法のうち，ハッシュを用いる方法に焦点を当て，探索精度を維持しつつ，計算時間を削減する方法を検討する．

3. ハッシュを用いた近似最近傍探索

ハッシュを用いた近似最近傍探索は，基本的にデータ空間を複数の軸に沿って線形分割し，その領域ごとにデータをハッシュテーブルに登録する．そして，クエリの入った領域または，その近傍の領域に入った点をハッシュテーブルから抽出し，その中から最近傍点を求めることで，高速な探索を実現している．本節では，共にハッシュを用いるが，点の抽出の仕方が異なる 2 つの既存手法について，それぞれ説明する．

3.1 Locality Sensitive Hashing

Locality Sensitive Hashing(LSH) [3], [4] はハッシュを利用した近似最近傍探索手法の 1 つである．ここでは LSH の中でも本研究に関連する，ベクトル空間で用いることが出来る，文献 [4] の LSH の概要について述べる．



(a)点の抽出　(b)距離計算

図 1　近似最近傍探索

LSH が近似最近傍点を求めることができるのは局所性に鋭敏な (Locality Sensitive) ハッシュ関数を用いるためである．局所性に鋭敏なハッシュ関数とは，距離が近い点同士は近いハッシュ値を取る確率が高く，距離が遠い点同士は近いハッシュ値を取る確率が小さいハッシュ関数である．

文献 [4] の LSH では次式のハッシュ関数を用いる．

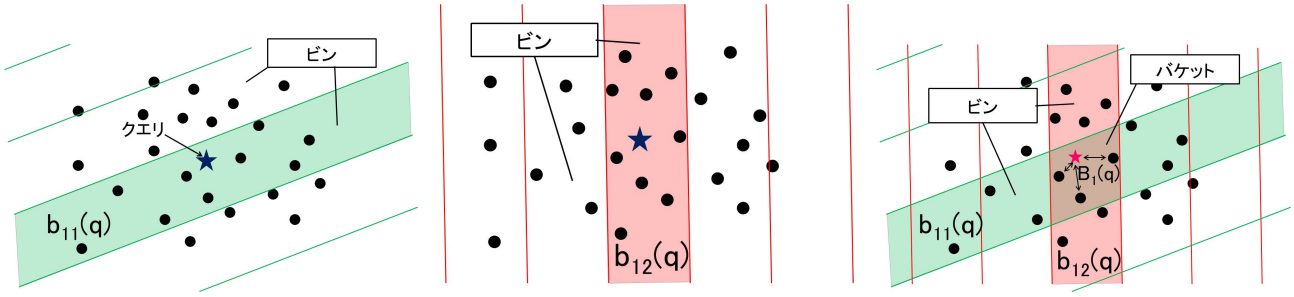
$$h_{ji}(\boldsymbol{p}) = \left\lfloor \frac{\boldsymbol{a}_{ji} \cdot \boldsymbol{p} + c_{ji}}{w} \right\rfloor \qquad (1)$$

ただし，$\boldsymbol{a}_{ji}$ は各次元の要素の値がガウス分布から独立に選ばれたベクトル，w はハッシュ幅であり，c_{ji} は区間 $[0, w]$ から一様に選ばれた実数である．

$h_{ji}(q)$ について，点 p^* がクエリ q と同じハッシュ値をとる（すなわち $h_{ji}(q) = h_{ji}(p^*)$ を満たす）領域を $\mathrm{b}^h_{ji}(q)$ とする．この 1 つ 1 つの線形分割された領域をビンという．図 2(a) の着色部分は $\mathrm{b}^h_{11}(q)$ を図示したものである．LSH は，このように局所性に鋭敏なハッシュ関数を用いることで，高確率でクエリに近い点が存在する空間（クエリが入ったビン）内の点を最近傍点の候補とし，計算時間を削減する．

ところが特徴空間が高次元の場合，領域 $\mathrm{b}^h_{ji}(q)$ が大きくなり，これによって明らかに最近傍点になり得ない点も抽出してしまい，探索効率が悪くなる．そこで，LSH はハッシュ関数を複数用いてハッシュ関数群を構成し，この問題を緩和する．図 2(b) はハッシュ関数群 g_1 を $g_1 = \{h_{11}, h_{12}\}$ とした時の例で，g_1 を構成する h_{11}，h_{12} において $\mathrm{b}^h_{11}(q)$ と $\mathrm{b}^h_{12}(q)$ の積領域に入った点のみを最近傍点の候補にする．また，このような積領域つまり $\mathrm{B}^g_j(q) = \bigcap_{j=1}^{k} \mathrm{b}^h_{ji}(q)$ をバケットと呼ぶ．これを一般化する．k 個のハッシュ関数 $h_{j1}, h_{j2}, \ldots, h_{jk}$ を組み合わせて，関数群 $g_j = \{h_{j1}, \ldots, h_{jk}\}$ を作る．このとき $g_j(q) = g_j(p^*)$，すなわち p^* が $^\forall i, h_{ji}(q) = h_{ji}(p^*)$ を満たす領域がクエリが入ったバケット $\mathrm{B}^g_j(q)$ であり，ここに入った点のみを最近傍点の候補とする．

さらに LSH は，複数のハッシュ関数群を用いて最近傍点の候補を増やし，精度向上を図っている．ハッシュ関数群 g_1 と g_2 があったとき，バケット $\mathrm{B}^g_1(q)$ またはバケット $\mathrm{B}^g_2(q)$ のいずれかに入った点を最近傍点の候補すると，候補となる点数を



(a) h_{11}　(b) h_{12}　(c) h_1 と h_2 によるバケット内で距離計算

図 2　Multi-Valued Hashing

増やすことができる．これを一般化すると，LSH は関数群 g_j を L 個用いて，$g_j(1 \leqq j \leqq L)$ において，一度でもクエリと同じバケットに入った点を最近傍点の候補とし，精度を向上させている．

3.2　Multi-Valued Hashing (MVH)

MVH は直交する複数の基底に沿って空間を等分し，これを基に各点にクエリからの距離の概算値を与えることで，クエリを中心とする超球に近い形で近傍点を抽出することができる．距離の概算値は求める処理は，それぞれの基底で空間を等分してビンを作成し，各点が入ってるビンがクエリの入ったビンからどれだけ離れているかを距離重みとして与え，全ての基底での距離重みの和を取っている．本節では Multi-Valued Hashing の詳細と，その問題点を示す．ここで説明するのは文献 [5] の MVH1 である．

3.2.1　手法の概要

ここでは，L2 距離に対応した MVH を例に示す．MVH では，

$$h_j(\boldsymbol{p}) = \left\lfloor \frac{\boldsymbol{\Psi}_j \cdot \boldsymbol{p}}{w} \right\rfloor \tag{2}$$

で示されるハッシュ関数を用いる．ただし，$\boldsymbol{\Psi}_j$ は正規直交基底，$\boldsymbol{p}$ はデータ点，w はハッシュ幅である．近傍点の抽出には $h_1 \ldots h_L$ の L 個のハッシュ関数を用いる．図 3(a) は h_1 の処理に関するものである．クエリの入ったビンの両隣のビン ($h_1(\boldsymbol{x}) = h_1(\boldsymbol{q}) \pm 1$ を満たすビン) に入った点には 1^2 の距離を反映した重み (距離重み) を加算する．また，クエリの s 個隣のビン ($h_1(\boldsymbol{x}) = h_1(\boldsymbol{q}) \pm s$ を満たすビン) には s^2 の距離重みを加算する．これを，$t-1$ 個隣のビン ($h_1(\boldsymbol{x}) = h_1(q) \pm t-1$ を満たすビン) まで行い，t 個以上隣のビンには全て距離重み t^2 を与える．図 3(b) は h_2 の処理に関するものであり，こちらも h_1 と同様に処理を行う．このような処理を全ての h_j に対して行う．次に h_j で付加された距離重みを全て加算する．するとこれが各点の距離の概算値となる．図 3(c) は，h_1, h_2 で付加された距離重みを加算した結果を示したものである．例えば，h_1 で u_1，h_2 で u_2 の距離重みを付加された点はそれぞれの値を足して，$u_1 + u_2$ という距離の概算値を保持する．これを全ての点に対して行う．その結果を基にして，距離の概算値のヒストグラムを作る．このヒストグラムはクエリの近傍では概ね L^2 距離を反映する．故に，このヒストグラムの中で，任意の閾値 v よりも小さな点のみを距離計算の対象とすることで，距離計算のコストの削減が期待できる．

3.2.2　問　題　点

MVH は，距離を反映した距離重みをに与えてヒストグラムを作成し，効率のよい点の抽出を可能にするが，このヒストグラムの作成に大きな処理時間がかかる．ビンの領域は 1 つの軸にのみ着目して領域を分割しているので領域が非常に大きく，近傍点になり得ない点が多く含まれる．MVH では図 3(a), 3(b) のようクエリの入ったビンだけでなく，さらにその近傍のビンに対しても各点に加算処理をしている．距離の概算値を求めるに膨大な加算処理が必要となり，たとえ球状に近い近傍点の抽出が実現しても，高速な処理は望めない．

4.　提 案 手 法

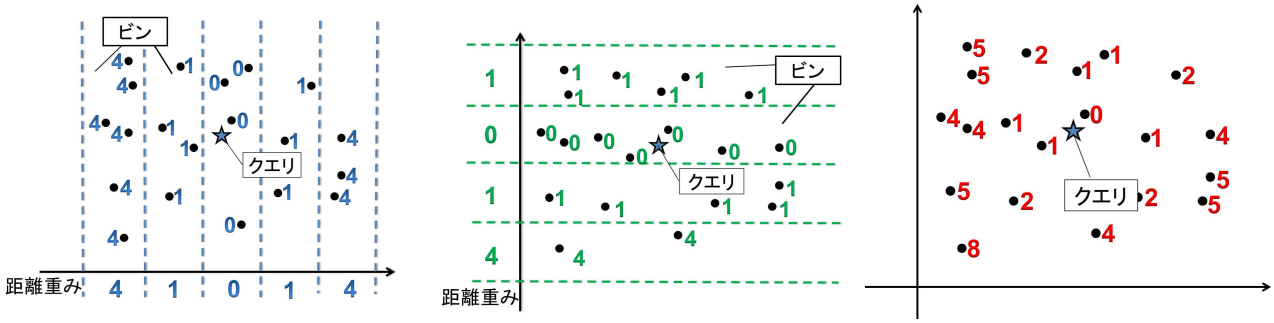
本節では，MVH の近傍点の抽出処理を遅くしている「多数のデータに対するアクセス」を一切せずに,「バケットに距離の概算値を与える」ことによって，効率よく距離の概算値の小さい点を探し出す方法を示す．MVH ではビンごとにハッシュテーブルを作るために，データの衝突が多くなる．そこで，提案手法ではこの衝突が処理が遅くなる原因であると考え，それぞれのビンの積領域の集合であるバケットを基にハッシュテーブルを作成する．これにより，ハッシュサイズがそれぞれのビンのハッシュサイズの積となり，データの衝突が減少する．そして，クエリの入ったバケットに近いバケットのみを参照することにより，無駄なデータへのアクセスを排除することができる．

4.1　バケット距離に基づく近傍点の抽出

提案手法でも，MVH と同じハッシュ関数を用いる．図 4 では，セル状になっている一つ一つの区画がバケットである．バケットに振られている数字の並びは，バケットを作るビンのインデックスを示す．また，この数字の並びはデータ空間内のバケットの位置を示す位置ベクトルとして考えることができる．ここで，バケットを次のように定義する．バケットを作るビンの数 (バケットの次元) を k，それぞれのビンを b_i^h とすると，バケット B は，次のように表される．

$$B =< b_1^h \ldots b_k^h > \tag{3}$$

そして，図 4 を見て分かるように，同じバケットに入っている



(a) h_1 の距離重み　(b) h_2 の距離重み　(c) h_1 と h_2 の距離重みの和

図 3　Multi-Valued Hashing

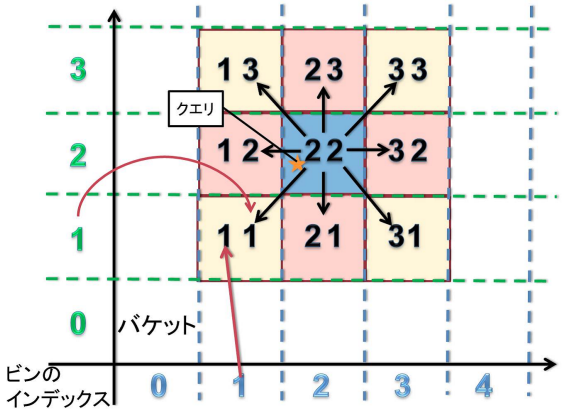


図 4　バケット距離による探索

点は，MVH での距離重みの加算が等しくなる位置にいるので，必ず同じ距離の概算値をもつ．また，バケットの位置ベクトルから，バケット間の距離 (バケット距離)D を定義でき，クエリの入ったバケットを $B(q)$，任意の点 p^* が入ったバケットを $B(p)^*$ とすると，D は

$$D(B(q), B(p^*)) = \sum_{l=1}^{k} (h_i(q) - h_i(p^*))^2 \qquad (4)$$

と表せる．すると，$(h_i(q) - h_i(p^*))^2$ がハッシュh_i での距離重みに等しく，その和をとっているので，バケット距離が MVH においてバケット内の点に与えられる距離の概算値と同じ値を取る．クエリが入ったバケットが分かれば，任意のバケット距離だけ離れたバケットのインデックスを知ることができるので，バケット距離の小さいバケットから順にデータを参照していけば，MVH のように距離重みの加算処理を行うことなく距離の概算値の小さい点のみを距離計算対象とすることができる．

図 4 を例に示す．図は，クエリのビンのインデックスが $<2,2>$ のときの例である．まず，このバケットに登録されているデータと距離計算を行う．次に，バケット距離 1 のバケットを探索する．例では $<1,2>$, $<2,1>$, $<2,3>$, $<3,2>$ である．このバケットを順に探索していく．バケットを作成する際にバケットにバケット内のデータ数を覚えさせておけば，探索した点の数を監視するのは容易であるので，この時点で十分なデータ数を探索できていれば探索を終了する．また，まだ不十分であればさらに遠いバケットに探索範囲を広げていく．

加算を行わずに，距離の概算値から点へのアクセスが可能になったため，高速化が期待できる．また，既存手法では K 近傍探索をする場合などを考えると，距離の概算値が小さいものからいくつか選ぶ処理が必要になるが，提案手法では概算値の小さいバケットから順に見ていけばよいので，クエリ周辺に点が少ない場合は広く探索するなど，探索範囲の調節が簡便である．

4.2　有効な基底の選択

距離の概算値を求めるためには，初めに直交基底をいくつか選んでおく必要がある．MVH では元のデータ空間の軸をランダムに選んでハッシュ関数を作る．しかし，これではあまり距離の概算値の計算に寄与しない基底を選んでしまう可能性がある．そこで，データの各軸が持つ情報量の大きさを考えると，分散が良い指標となる．提案手法では，予めデータベース内のデータの分布を調べておき，軸を分散の大きい順に並べ替え，大きいものから順に選んでハッシュ関数を作る．このようにすることで，情報量の多い軸から距離の概算値を求めることができ，用いる軸の数が少なくても，大きな情報量を得ることができる．また，これにより距離計算の打ち切り効率も上昇する．

5.　実験・考察

提案手法の有効性を確認するため，MVH と LSH との比較実験を行った．用いた計算機は，CPU が Opteron8439SE (2.8GHz)，メモリは 128GB である．距離尺度は L^2 ノルムを用いた．

正規分布に従うデータに対して，クエリを与えたときの MVH, LSH, BDH の速度，精度，メモリ使用量を比較した．各手法で結果が複数あるのは，それぞれの手法において，様々なパラメータを与えて実験を行ったためである．正規分布のデータは，各軸で平均 0，分散 σ の正規分布に従う 30 次元，100 万点の人工データである．ただし，σ は 100〜400 の範囲からランダムに選ぶ．実験結果を図 5(a)〜 6(b) に示す．図 5(a)， 5(b) は横軸が処理時間 (ms)，縦軸が精度 (%) であり，図 6(a)， 6(b) は横軸が処理時間 (ms)，縦軸がメモリ使用量 (GB) を示して

いる．(a)，(b) はそれぞれの精度帯での結果を示している．

5.1 結　　果

全探索による処理時間は 95.44ms であったので，グラフには処理時間がこれ以下のものを示す．図 5(a) から分かるように，同じ精度で比較すると概ね提案手法が既存手法の処理速度を上回っている．同一精度で比較をすると，MVH の処理時間を 1 とすれば，提案手法の処理時間は，精度 95%〜96%では，約 0.63，精度 30%〜31%なら，約 0.066 であり，特に低精度帯で飛躍的な高速化が実現されている．

図 5(b) を見ると，提案手法では 99%以上の精度が出ていないことが分る．提案手法では，分散の大きい軸に着目してバケットを作成し，着目した軸で値が離れていれば，バケット距離は遠くなる．つまり，バケットの作成にデータ空間の全ての軸を用いることはできないので，分散の小さい軸の値はバケット距離に影響しない．すると，最近傍点が分散の大きい軸では値がクエリと離れているが，他の大部分の軸で値が近いという場合，最近傍点が遠くのバケットに入ってしまう．このような場合，最近傍点が含まれるバケットにアクセスする前に処理を打ち切ってしまうので，探索が失敗する．このような場合でも，探索できるようにすれば，膨大な数のバケットへの参照が必要になるので，速度が大きく低下する．

また，図 6(a)， 6(b) から，同一のメモリ使用量で処理速度を向上させていることが分かる．

6. 結　　び

バケットベースのハッシュテーブルの導入により，少ない処理で効率的に近傍点を抽出する手法を考案した．また，データの分散を考慮して基底を選ぶことで，探索に有効なハッシュテーブルの作成を可能にした．実験の結果，精度 95%〜96%では，処理時間を約 63%に，精度 30%〜31%では，約 6.6%に抑えることができた．

今後の課題としては，バケット内のクエリの入った位置に応じて，同じバケット距離のバケットの中でも優先順位を与えることができるように改良することが挙げられる．これを実現するには，バケット内にさらに小さなバケットを仮想的に用意することで，バケット内のクエリの位置を細かく特定することが必要となる．これによって，同じバケット距離の中にも近さの順位をつけることができるので，効率良く近い領域から探索が行える．

文　　献

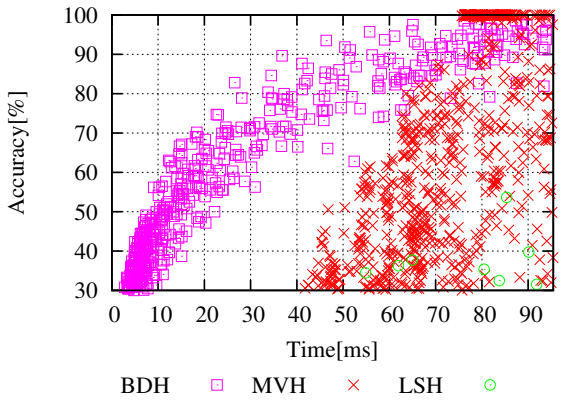
[1] K. Kise, K. Noguchi, and M. Iwamura, “Robust and efficient recognition of low-quality images by cascaded recognizers with massive local features,” Proc. 1st Int’l Workshop on Emergent Issues in Large Amount of Visual Data (WS-LAVD), pp.2125–2132, Oct. 2009.

[2] Y. Ke and R. Sukthankar, “PCA-SIFT: A more distinctive representation for local image descriptors,” Proc. CVPR2004, pp.506–513, 2004.

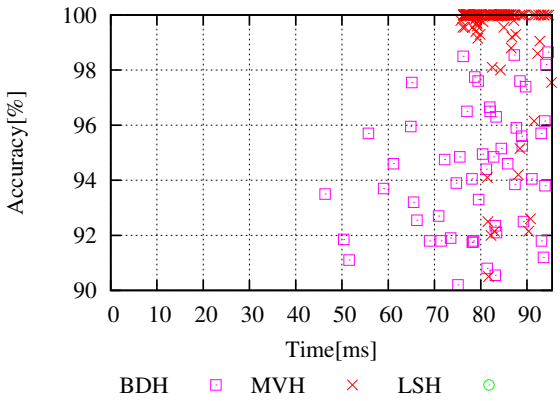
[3] P. Indyk and R. Motwani, “Approximate nearest neighbor: Towards removing the curse of dimensionality,” Proc. 30th Symposium on Theory of Computing, pp.604–613, 1998.

[4] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” Proc. 20th annual symposium on Computational geometry, pp.253–262, 2004.

[5] 多田匡志，武藤大志，岩村雅一，黄瀬浩一，“近さの多段階表現に基づく近似最近傍探索の一般的な分布への拡張，” データ工学と情報マネジメントに関するフォーラム論文集，Feb. 2010.

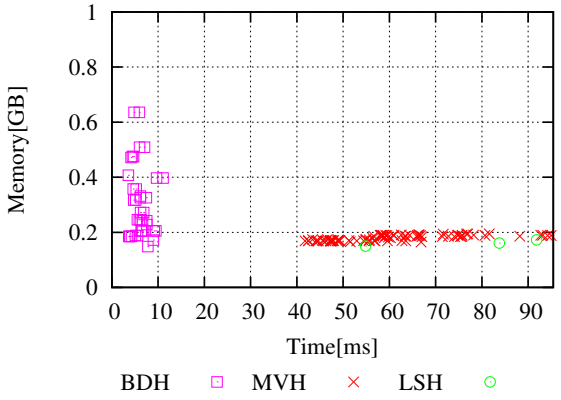


(a) 30% ～100%

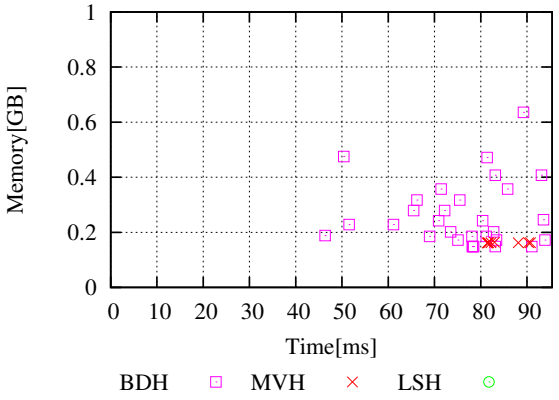


(b) 90% ～100%

図 5　処理時間ー精度



(a) 30% ～ 35%



(b) 90% ～ 95%

図 6　処理時間ーメモリ使用量