DEIM Forum 2011 E2-6

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

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て最近傍候補 (最近傍点の候補) とし，これらのみを距離計算の対象として，近似的に最近傍点を探索する問題である．本稿ではハッシュ法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 では抽出の段階で真の最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．MVH はクエリを中心とする超球に近い形で距離計算の対象を抽出でき，心の最近傍点を漏らすことは少ないが、この処理自体に時間がかかる．そこで，本稿ではバケットベースのハッシュテーブルを作成することで、MVH と同等の精度で高速に距離計算対象の抽出を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

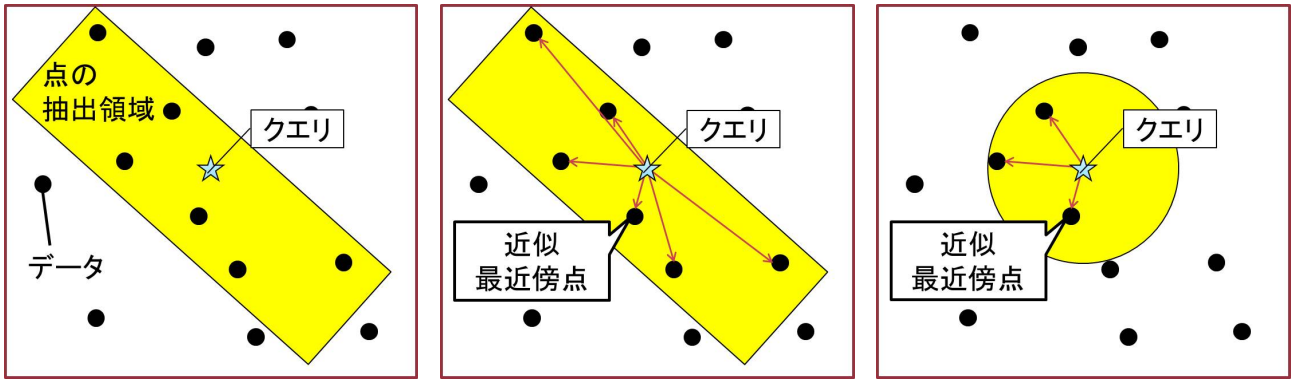
Tomokazu SATO†, Tomoyuki MUTOH††, Masakazu IWAMURA††, and Kouichi KISE††

† 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

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最も類似するもの，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．しかし探索の処理時間は，データベースのに登録されているベクトルの数と次元数に比例するので，これらが大きくなると処理時間が膨大になる．例えば、画像 1000 枚のデータベースから検索を行うのに約 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，予めクエリの最近傍点である可能性の高い点を抽出し，それらに対してのみ距離計算を行い，近似的に最近傍点を探索するものである．このように距離計算の対象を選択することによって，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．つまり近似最近傍探索は，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．そこで，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する．性能とは，処理時間（クエリ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 点を指す．



(a) 近傍点の抽出　(b) 距離計算　(c) 超球状の抽出

図 1　近似最近傍探索

近似最近傍探索には様々な手法が考案されているが，本稿ではハッシュ法を用いた探索法を扱う．ハッシュ法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て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 は真の最近傍点を漏らしやすいため，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距離計算の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，近傍点の抽出領域を小さくしても，真の最近傍点を漏らすことが少ない．しかし，MVH には近傍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，データベースに含まれる膨大な数の点に対して，何度も距離重みの加算処理が必要になる．この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 での近傍点抽出の遅延を招く「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を排除し，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 MVH と比べて，精度 94%〜95%で処理時間を約 81%，精度 30%〜31%では約 96%削減することができた．

2.　近似最近傍探索

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))，その中で距離計算を行い，最近傍点を探索する (図 1(b))．図 1(a) の抽出処理によって，距離計算の対象が著しく減少し，処理時間を大きく削減することができる．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であり，矩形領域が小さいほど真の最近傍点を漏らしやすい．つまり，精度と処理時間には，近傍点を抽出する領域を小さくほど，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．近傍点を抽出する領域を小さくしても，真の最近傍点を漏らさず抽出するには，図 1(c) のように，クエリを中心とする超球状に点を抽出することが重要となる．

3.　ハッシュ法を用いた近似最近傍探索

ハッシュ法を用いた近似最近傍探索は，基本的にデータ空間を線形分割し，分割された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，共にハッシュ法を用いるが，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．

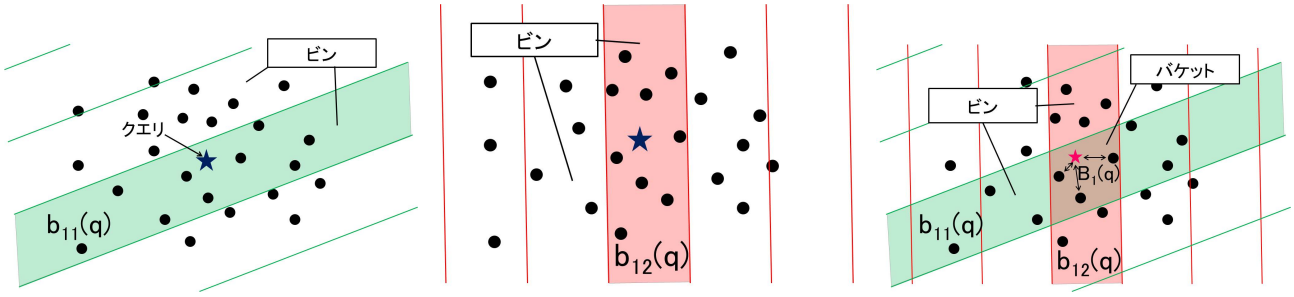
3.1　Locality Sensitive Hashing

Locality Sensitive Hashing(LSH) [3], [4] はハッシュ法を利用した最も代表的な近似最近傍探索手法の 1 つである．ここでは LSH の中でも，ベクトル空間で用いることが出来る，文献 [4] の LSH について述べる．

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 (Locality Sensitive) なハッシュ関数を用いるためである．局所性に鋭敏とは，距離が近い点ほど近いハッシュ値を取る確率が高くなる性質を指す．

LSH では，上記のように Locality Sensitive なハッシュ関数を k 個生成し，これらを 1 組の集合 (ハッシュ関数群) として，対応する k 個のハッシュテーブルを参照することにより最近傍候補を抽出する．このような処理を，L 群のハッシュ関数群で行い，各ハッシュ関数群から得られた最近傍候補の和集合を最終的な最近傍候補とする．つまり，LSH では $k \times L$ 個のハッシュテーブル作成することにより，最近傍候補を抽出している．

文献 [4] の LSH では次式のハッシュ関数を用いる．



(a) h_{11} に対応するビン　(b) h_{12} に対応するビン　(c) h_{11} と h_{12} によるバケット内で距離計算

図 2　Locality Sensitive Hashing

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

ただし，$i = 1 \ldots k$，$j = 1 \ldots L$ であり，$\boldsymbol{a}_{ji}$ 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 は空間を分割する幅であり，c_{ji} は区間 $[0, w]$ から一様に選ばれた実数である．

最近傍候補抽出の概要を以下に示す．$h_{ji}(q)$ 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．このように，1 つの軸にのみ着目して線形分割された 1 つ 1 つの領域を**ビン**という．図 2(a) の着色部分は $\mathrm{b}^h_{11}(q)$ を示したものであり，この領域に入った点はクエリの近傍にある確率が高い．

ところがデータ空間が高次元の場合，領域 $\mathrm{b}^h_{ji}(q)$ が大きくなり，明らかに最近傍点になり得ない点も抽出してしまう．そこで，LSH は複数のハッシュ関数を用いてハッシュ関数群 g を構成し，この問題を緩和する．図 2(a) はハッシュ関数 h_{12} に対応するビンの様子を表したものであり，ハッシュ関数群 g_1 を $g_1 = \{h_{11}, h_{12}\}$ とすると，データ空間は図 2(c) のように分割される．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候補にする．また，このような積領域つまり $\mathrm{B}^g_j(q) = \bigcap_{i=1}^{k} \mathrm{b}^h_{ji}(q)$ を**バケット**と呼ぶ．k 個のハッシュ関数でハッシュ関数群を構成する場合には，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，この領域に入った点のみを最近傍候補として抽出する．

さらに LSH は，L 個のハッシュ関数群を用いて最近傍候補を増やし，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候補すると，候補となる点数を増やすことができる．LSH は $g_j (1 \leqq j \leqq L)$ において，それぞれのハッシュ関数群で抽出された点の和集合を最近傍候補とし，精度を向上させている．

3.2　Multi-Valued Hashing (MVH)

Multi-Valued Hashing (MVH) はクエリの近傍点を超球に近い形で抽出することができる既存手法である [5]．ここでは文献 [5] の MVH1 を MVH として，詳細を説明する．

MVH は，予めクエリからの距離の概算値を求めておくことで，クエリを中心とする超球に近い形でクエリの近傍点を抽出することが可能となる．MVH も Locality Sensitive なハッシュ関数を用いる．

距離の概算値を求めるには，k 個のハッシュ関数で構成されたハッシュ関数群が必要となる．それぞれのハッシュ関数で，クエリと各点の距離の概算値直交基底に対して図 3(a) のように，基底に対して垂直に空間を等分してビンを作成し，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．そして各点に，入ってい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，各基底での距離重みの和を取っている（図 3(a)〜 3(c)）．本節では Multi-Valued Hashing の詳細と，その問題点を示す．

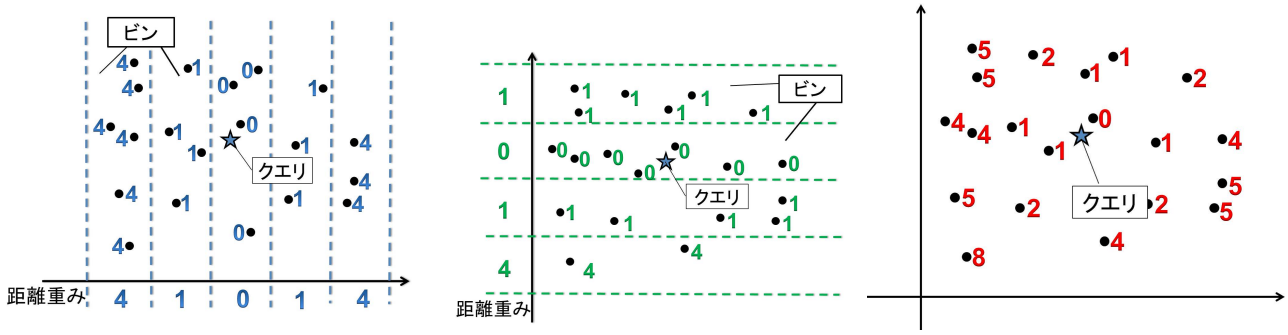
3.2.1　手法の概要

ここでは，L^2 距離に対応した MVH を例に示す．MVH では，

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

で示されるハッシュ関数を用いる．ただし，$\boldsymbol{\Psi}_i$ は正規直交基底，$\boldsymbol{p}$ はデータ点，w は空間を分割する幅である．近傍点の抽出には $h_1 \ldots h_k$ の k 個のハッシュ関数でハッシュ関数群を構成し，それぞれのハッシュ関数でデータベース内の点にクエリからの距離重みを与える．

図 3(a) はハッシュ関数 h_1 における投票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 と同様の処理を行う．このような処理を k 個全ての h_i に対して行う．次に h_i で付加された距離重みを全て加算する．するとこの値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．



(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

そして任意の閾値 v よりも小さな点のみを距離計算の対象とすることで，近似的にクエリを中心とする近似超球領域から近傍点を抽出することができ，距離計算コストの削減を図る．

3.2.2　問　題　点

MVH は，各点に距離を反映した距離重みを与えて距離の概算値を計算することで，効率の良い点の抽出を可能にするが，この概算値の計算に大きな処理時間がかかる．ビンの領域は 1 つの基底にのみ着目して空間を分割したものなので，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) のように，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．そのため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 での近傍点抽出の計算量の増大を招く「多数のデータに対するアクセス」を排除し，「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抽出する方法を示す．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．提案手法ではこの衝突が，計算量が大き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．バケットはビンに比べて 1 つの領域が非常に小さく，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

提案手法では，バケットを元にハッシュテーブルを作成する．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，同じバケットに入る点は，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

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

と表せる．すると，$(h_i(q) - h_i(p))^2$ が MVH のハッシュ関数 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位置ベクトル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，クエリが入ったバケットを構成するビンのインデックスが $< 2,2 >$ の場合である．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．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，MVH 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，探索範囲の調節が容易である．

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

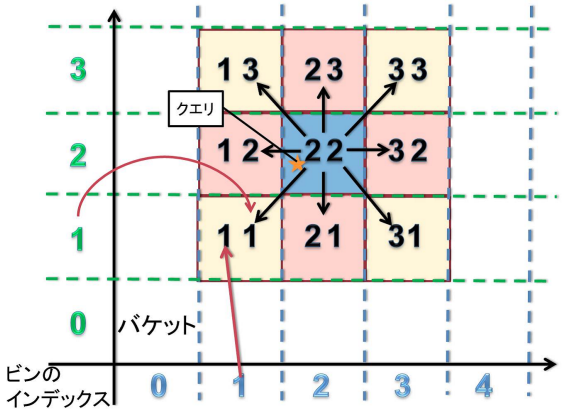


図 4　バケット距離による探索

選んでおく必要がある．MVH では元のデータ空間の基底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，データの各基底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，基底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，情報量の多い基底から距離の概算値を求めることができ，用いる基底の数が少なくても，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提案手法と MVH，LSH の速度（クエリを与えてから解が得られるまでの時間），精度 (真の最近傍点が得られた割合)，メモリ使用量 (データベース自体のメモリ使用量を除く) 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，各基底で平均 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．

図 5(a) 5(b) は各手法の処理速度と，精度のトレード・オフの関係を示した．図 6(a) 6(b) はそれぞれ精度が 30% 〜35% の場合と，90% 〜95% の場合で，処理速度とメモリ使用量のトレード・オフの関係を示している．

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完全に提案手法が既存手法の処理速度を上回っている．提案手法は MVH と比べて処理時間を，精度 94%〜95%では約 81%を削減，精度 30%〜31%では 96%を削減した．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かる．この原因は，クエリからデータへの真の距離と，バケット距離が持つ情報量の違いにある．バケット距離は概ね真の距離を反映して作成されるが，その計算には全ての基底を用いないため，クエリから見たデータの方向によっては，バケット距離と真の距離は大きく異なる．これにより，本当はクエリから遠い場所にあるデータのバケット距離が非常に小さく見積もられ，真の最近傍点よりも近くなる場合がある．その結果，相対的に真の最近傍点のバケット距離が大きくなるため，このような場合には真の最近傍点が距離計算候補から漏れる．

しかし，図 6(a)， 6(b) から，メモリ使用量は既存手法に比べて増加している．これは，提案手法のハッシューテーブルがバケットベースであるために，既存手法と比べてハッシュサイズが大きくなってしまうことが原因である．

6. 結　　び

MVH では最近傍候補の抽出に時間がかかるという問題があった．本稿ではバケットベースのハッシュテーブル導入により，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，MVH と比べて，精度 94%〜95%では，処理時間を約 81%，精度 30%〜31%では，約 96%削減することができた．

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．

謝　　辞

本研究は SCAT 研究費助成ならびに科研費補助金基盤研究 (B)(22300062) の補助による．

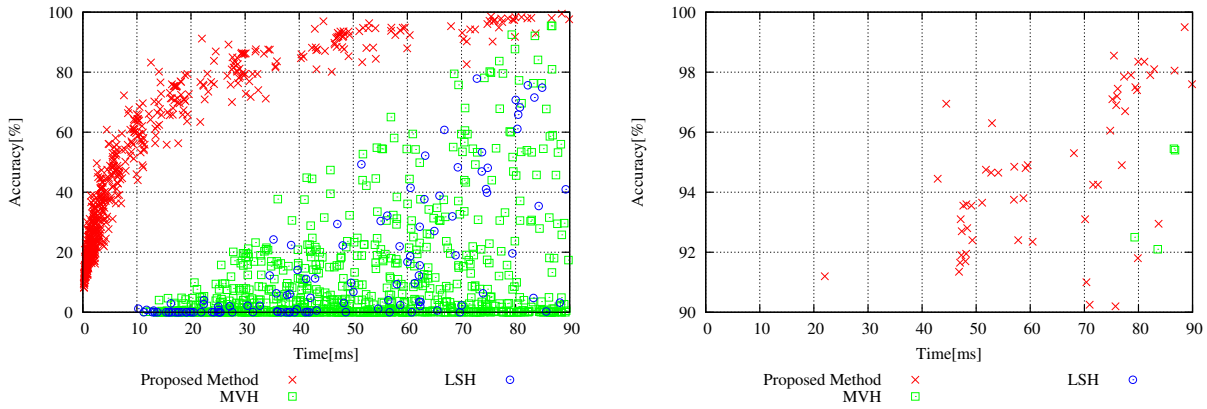
文　　献

[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.

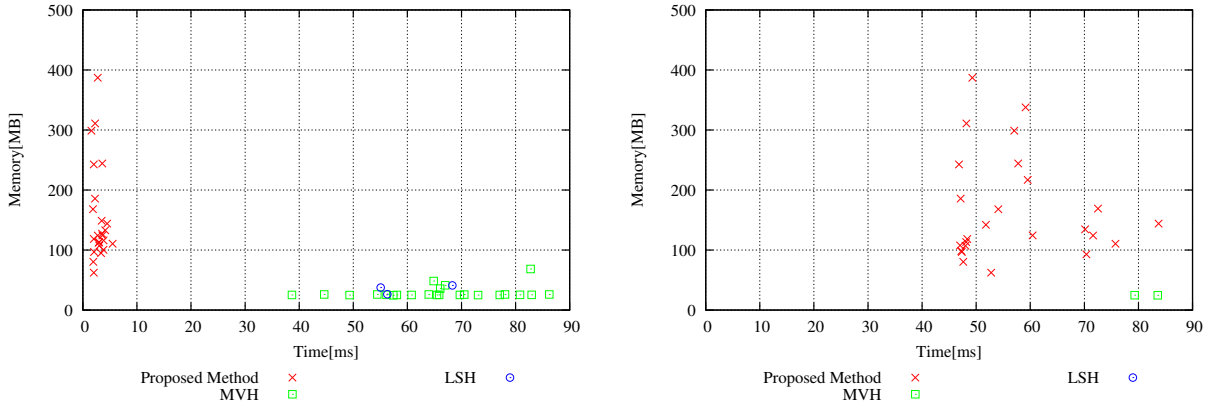
[3] P. Indyk and R. Motwani, “Approximate nearest neighbor: Towards removing the curse of dimensionality,” Proc. 30th Symposium on Theory of Computing, pp.604–613, 1998.

[4] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni,



(a) 精度：0% ～100%　　(b) 精度：90% ～100%

図 5　処理時間ー精度



(a) 精度：30% ～ 35%　　(b) 精度：90% ～ 95%

図 6　処理時間ーメモリ使用量

"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.