データの重複登録に基づく高速・省メモリな近似最近傍探索

岩村　雅一†　　武藤　大志†　　黄瀬　浩一†

† 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: {masa,kise}@cs.osakafu-u.ac.jp

あらまし　本稿では，近似最近傍探索において，探索精度を保ったまま処理時間とメモリ使用量を削減する手法を提案する．提案手法を使用することで，従来手法である LSH の 18%の処理時間と 90%のメモリ使用量で LSH と同等の精度を実現できることを実験により確認した．さらに，何故このような現象が起こるのかについて, [1] に示されている LSH の探索効率の基準 ρ を用いて考察する．

キーワード　近似最近傍探索，locality sensitive hashing，探索効率

1. はじめに

最近傍探索は，データベース S 中から，クエリ $\boldsymbol{q}$ と最も距離が近いデータ $\boldsymbol{p} \in S$ を発見する，情報処理において基本的な問題である．この問題はクエリと全てのデータとの距離を計算すれば必ず正しい解が求まるため，すこぶる単純である．この処理に必要な計算時間は，データの次元数 d とデータ数 n に比例し，$O(dn)$ と表される．

ところが，扱うデータの規模が大きくなれば，この単純な問題はたちまち容易には解けなくなる．例えば，20 億ベクトルをデータベースに登録しておき，物体認識を行うタスク [2] も存在する．このタスクを実時間で実行するためには，最近傍探索の高速化は不可欠である．

最近傍探索を高速化するためには，データベースを木構造などで構造化し，距離計算回数を減らすことが有効である [3]．しかし，構造化するとデータ以外の情報も記憶する必要があるため，必要なメモリ使用量が増加する。一般に計算時間とメモリ使用量にはトレードオフの関係があると考えられ，次元数が 2 より大きい場合にデータ数 n に対して計算時間が対数、メモリ使用量が線形に増加するアルゴリズムは知られていない [4]．

この限界を超えるために近年注目されているのが近似最近傍探索である．この方法は，最近傍探索において必ずしも最近傍データが得られなくても良いというように条件を緩和することによって，最近傍探索に比べて計算時間とメモリ使用量を大幅に削減できる．代表的な手法としては、木構造を用いる Approximate Nearest Neighbor(ANN) [4] やハッシュを用いる Locality Sensitive Hashing(LSH) [1], [5]，Spectral Hashing [6]，Minwise Hashing [7] などが知られている。近似最近傍探索では，精度（本稿では最近傍データが正しく求まる確率），計算時間，メモリ使用量にトレードオフの関係があると考えられる．そのため，ある精度を実現するために必要な計算時間とメモリ使用量をどれだけ削減できるのかが問題となる．

本稿では，ハッシュを用いる近似最近傍探索手法である LSH において，同一精度を実現するために必要な計算時間とメモリ使用量を削減する手法を提案する．通常このように計算リソースを削減する場合，最近傍探索を用いる識別器で行われているように，データベースに登録されるデータ数の削減を考える [8] のが一般的と思われるが，本稿では逆にデータベースに登録されているデータを重複登録し，データベースに登録されるデータ数を増加させることによって実現する．一見逆説的な提案手法を使用することによって，従来手法である LSH に対して 18%の計算時間と 90%のメモリ使用量で同等の精度を実現できることを実験で確認する．さらに，何故このような現象が起こるのかについて, [1] に示されている LSH の探索効率の基準 ρ を用いて考察する．

2. 従来手法：Locality Sensitive Hashing

本節ではベクトルデータを対象とした LSH である p-stable LSH [1] を説明する．これは提案手法に対して従来手法に当たる．

LSH による近似最近傍探索は，次の 2 ステップで実現される．

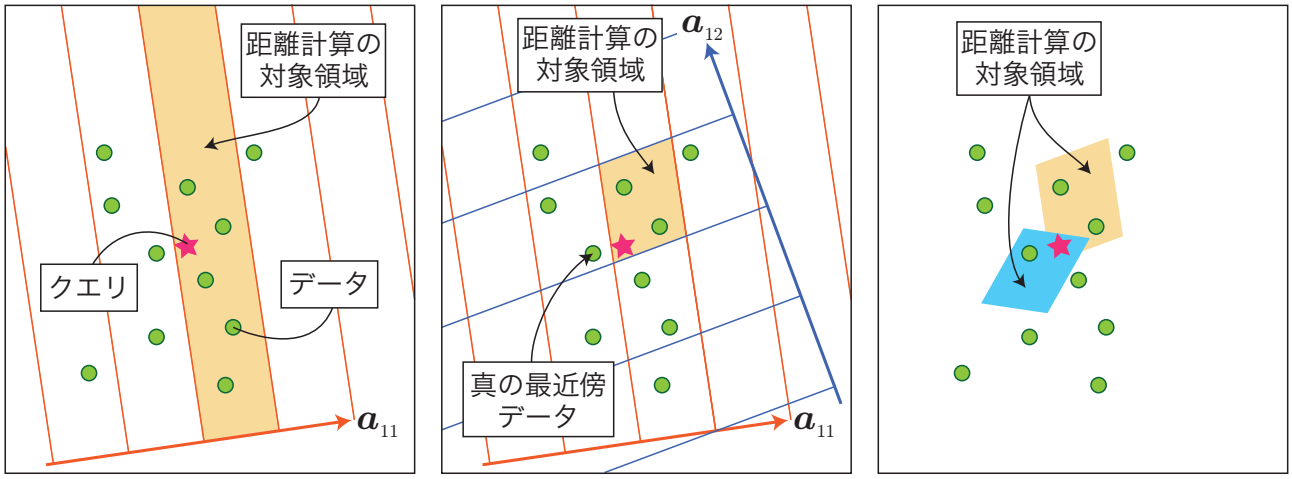
（1） クエリとの距離を計算すべきデータ（距離計算対象）を選択する．

（2） (1) で選択された距離計算対象のデータに対してクエリとの距離を計算し，最近傍データを決定する．

ここで注意しておきたいのは，(2) に近似処理は含まれていないことである．したがって，(1) で真の最近傍データが距離計算対象に含まれていれば探索は成功するが，含まれていなければ失敗する．以下では，LSH において精度を決定する (1) の処理，すなわち距離計算対象の絞り込みがどのようにされているかを説明する．

2.1 LSH による距離計算対象の絞り込み

LSH がハッシュ関数を用いて距離計算対象を絞り込む様子を図示したのが図 1 である．星はクエリで，丸は



(a) ハッシュ関数が 1 つの場合. (b) ハッシュ関数群（バケット）が 1 つの場合. (c) ハッシュ関数群（バケット）が 2 つの場合.

図 1: LSH による距離計算対象の絞り込みの様子.

データを表す.

まず，図 1(a) はハッシュ関数 1 つのみを用いて距離計算対象を決定する様子である．結果を先に述べると，図中のオレンジ色の部分に存在するデータが距離計算対象である．図中のベクトル $\boldsymbol{a}_{11}$ が何であるかを説明する前に，ここで用いられているハッシュ関数について述べておく.

LSH で使用されるハッシュ関数 $h(\cdot)$ は次式で与えらえれる(注1).

$$h(\boldsymbol{v}) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v}}{w} \right\rfloor \tag{1}$$

ここで引数 $\boldsymbol{v}$ にはデータ $\boldsymbol{p}$ あるいはクエリ $\boldsymbol{q}$ を与える．$\boldsymbol{a}$ はデータを射影する d 次元ベクトルであり，d 次元正規分布に従って定められる．図中の $\boldsymbol{a}_{11}$ はこれに当たる．w はハッシュ幅であり，図 1(a) のビンの幅を決めるためのパラメータである.

話を図 1(a) に戻そう．LSH では式 (1) のハッシュ関数を用いて，クエリと同じハッシュ値を持つデータのみを距離計算対象とする．その計算には前述のベクトル $\boldsymbol{a}$ が使用される．幾何的な説明をすれば，ベクトル $\boldsymbol{a}$ にデータやクエリを射影して，等間隔に区切られたビンのどれに属すかでハッシュ値が決定される．クエリと同じビンに入ったデータは同じハッシュ値を持っているので，距離計算対象となる．このことを改めて定義すると，「$h(\boldsymbol{q}) = h(\boldsymbol{p})$ を満たす全てのデータ $\boldsymbol{p}$ を距離計算対象とする」となる.

ここで図 1(a) を注意して見れば，クエリから遠い点も距離計算対象として選ばれていることがわかる．距離計算対象を削減するという観点から言えば効率が悪いのだが，データベース中の全データを 1 本のベクトルに射影しただけでは上手に空間分割できなくても仕方がないといえる．そこで，図 1(b) のようにハッシュ関数を複数（k 個）用いて距離計算対象を更に絞り込む．ここで，これら k 個のハッシュ関数から成るハッシュ関数群を次式のように定義する.

$$g_i(\boldsymbol{v}) = \{h_{i1}(\boldsymbol{v}), h_{i2}(\boldsymbol{v}), \ldots, h_{ik}(\boldsymbol{v})\} \tag{2}$$

複数のハッシュ関数は添字で区別する．図中のベクトル $\boldsymbol{a}$ に添字が付いて $\boldsymbol{a}_{ji}$ となっているのはそのためである．添字 i と j はそれぞれハッシュ関数とハッシュ関数群の番号を表す．このとき，各ハッシュ関数の距離計算対象の積集合をバケットと呼び，クエリのあるバケット（図 1(b) のオレンジ色の領域）内のデータを距離計算対象とする.

図 1(a) と図 1(b) を改めて注意して見れば，実はどちらの図においても真の最近傍データは距離計算対象に含まれていないことに気付く．そこで真の最近傍データが距離計算対象に含まれる確率を上げるために図 1(c) のようにバケットを複数（L 個）用いることとする．そして，各バケットの距離計算対象の和集合を最終的な距離計算対象とする．図 1(c) では，オレンジ色の領域と水色の領域のいずれかに含まれるデータが距離計算対象となる.

2.2 バケット数 L と LSH の性能の関係

LSH では，ハッシュ関数で使用する w の他に，2 つのパラメータ k と L によって性能が決まる．このうちバケット数 L に注目して，L と精度，計算時間，メモリ使用量の関係を考える.

精度

L が増えるほど距離計算対象が単調に増加するため，精度は単調に増加する.

計算量

L の増加に伴ない，参照するハッシュテーブル数が単調に増加し，また距離計算対象も単調に増加するため，精

(注1)：式 (1) は本稿において重要でない項 b を省いている．本来のハッシュ関数は次式で与えられる．$h(\boldsymbol{v}) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v} + b}{w} \right\rfloor$ ここで b_{ji} は区間 $[0, w]$ から一様乱数により定められた実数である.

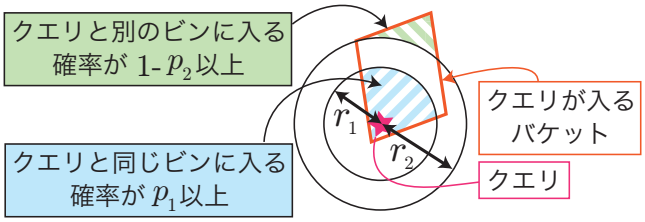


図 2: 局所性に鋭敏なハッシュ関数

度は単調に増加する．

メモリ使用量

LSH ではハッシュテーブルを構築する際にメモリを使用する．L の増加に伴ない，必要なハッシュテーブル数が単調に増加するため，メモリ使用量も単調に増加する．

2.3 局所性に鋭敏なハッシュ関数と探索効率

局所性に鋭敏なハッシュ関数は，その探索効率が [1] に示されているので紹介する．ここでの記述は 5. で述べる考察の基盤となる．

式 (1) のハッシュ関数は局所性に鋭敏 (Locality-Sensitive) なハッシュ関数と呼ばれる．局所性に鋭敏なハッシュ関数は，近いベクトル同士は同じハッシュ値を取る確率が高く，遠いベクトル同士は同じハッシュ値を取る確率が低いという性質を持つ．数式を用いてより具体的に書けば次のようになる（図 2 は以下の記述を図示したものである）．

$$\begin{cases} \mathrm{P}[h(\boldsymbol{q}) = h(\boldsymbol{p})] \geqq p_1 & \text{for } \boldsymbol{p} \in B(\boldsymbol{q}, r_1) \\ \mathrm{P}[h(\boldsymbol{q}) = h(\boldsymbol{p})] \leqq p_2 & \text{for } \boldsymbol{p} \notin B(\boldsymbol{q}, r_2) \end{cases} \tag{3}$$

ここで $B(\boldsymbol{q}, r)$ はクエリ $\boldsymbol{q}$ から半径 r 以内にある点の集合を意味する．したがって式 (3) は，クエリ $\boldsymbol{q}$ から距離 r_1 以内の点は確率 p_1 以上でクエリと同じハッシュ値を持ち，クエリ $\boldsymbol{q}$ から距離 r_2 以上の点は確率 $(1 - p_2)$ 以上でクエリと別のハッシュ値を持つということを意味する．ここで $r_1 < r_2$ かつ $p_1 > p_2$ を満たすとする．

局所性に鋭敏なハッシュ関数を用いる LSH の探索効率は，次式で与えられる基準 ρ を用いて記述される．

$$\rho = \frac{\log 1/p_1}{\log 1/p_2} \tag{4}$$

[1] によると，必要なメモリ使用量は $O(dn + n^{1+\rho})$ で，計算時間のほとんどは $O(n^\rho)$ 回の距離計算で占められる．このとき，$O(n^\rho \log_{1/p_2} n)$ 回のハッシュ関数の評価が必要である．

3. 提 案 手 法

データベースにデータを重複登録することによって近似最近傍探索手法に必要な計算時間とメモリ使用量を削減する手法を提案する．

提案手法について述べる前に LSH の特性を確認しておく．2.2 で述べた LSH の性能をまとめたものが図 3 である．図 3 に示すように，L の増加に伴って精度は上昇するものの，計算時間とメモリ使用量も同時に増加する．LSH にとって望ましい性能（図中の星印）は，L が大きなときの精度と，L が小さなときの計算時間とメモリ使用量であるので，これらを両立する方法が求められる．

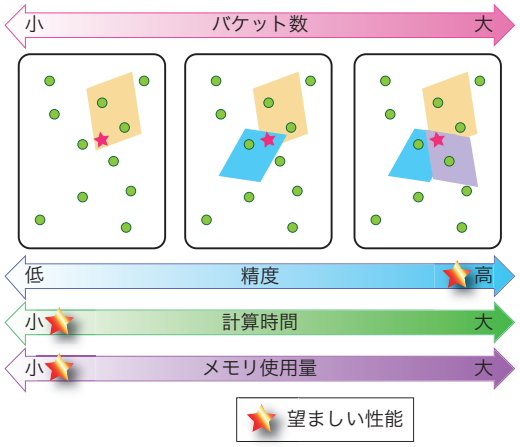


図 3: バケット数 L と LSH の性能の関係.

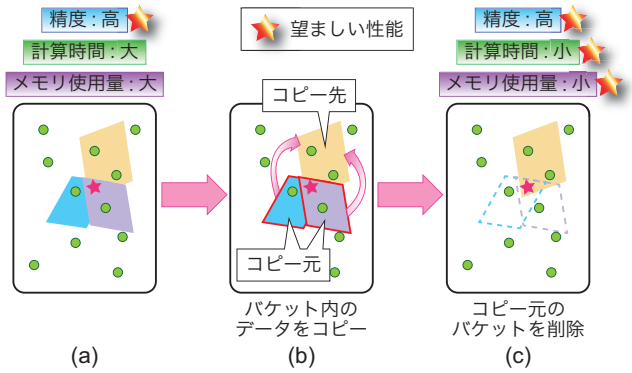


図 4: 提案手法の概要.

3.1 提案手法の概要

本研究では，LSH において L が大きいときの精度を，L が小さいときの処理時間とメモリ使用量で実現する．

提案手法の概要を図 4 に示す．まず，図 4(a) では大き目の L を持つ LSH を構築する．そして図 4(b) のように，1 つのバケット（「コピー先」バケット）に残りのバケット（「コピー元」バケット）の情報をコピーし，図 4(c) のようにコピー元バケットを削除する．これにより，少数のバケットのみを用いてバケット数が大きいときの性能を実現できると考えられる．

3.2 提案手法の詳細

提案手法の詳細な処理を図 5 を参照しながら説明する．

最初に図 4 の「コピー元バケット」と「コピー先バケット」の代わりに，「コピー元ハッシュ関数群」と「コピー先ハッシュ関数群」を用意する．図のように，どちらのハッシュ関数群にも同じデータが登録されているが，

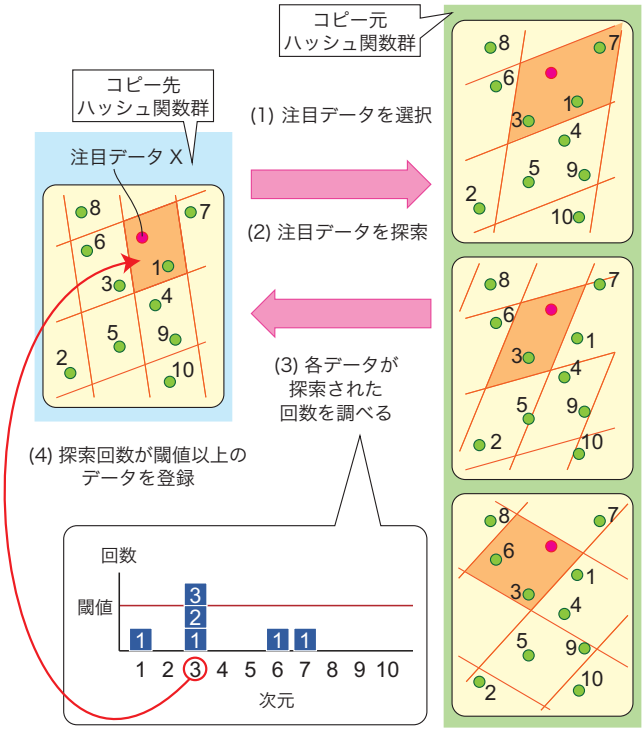


図 5: 提案手法

ハッシュ関数群を構成するハッシュ関数が異なる．次にコピー先ハッシュ関数群に登録されているデータを 1 つ選ぶ．説明の都合上，このデータを X と呼ぶ。X をクエリに見立ててコピー元ハッシュ関数群で探索する．そして，コピー元ハッシュ関数群で X と同じバケットに入った各データについて，同じバケットに入った回数を数える．回数が閾値 t 以上のデータのみを選択して，コピー先ハッシュ関数群の X が属していたバケットに登録する．ただし，既に登録済みの場合は追加登録しない．この処理を，X とするデータを変えながら一定数のデータに対して行う．全データのうち，この処理に用いるデータの割合を α で表す（$0 \leqq \alpha \leqq 1$）．最後にコピー元ハッシュ関数群を破棄し，コピー先ハッシュ関数群のみで構成される LSH を作成する．なお，LSH には「バケットににデータを登録する」という概念はないため，データの追加登録の際には当該バケットを構成する全てのハッシュ関数にデータを登録する．

データの追加登録処理はクエリがより多く発生するバケットで行うのが効果的である．通常はクエリの分布がわからないため，本研究ではデータの分布とクエリの分布が同一だと仮定する．その場合，X とするデータを一様乱数に基づいて選択すればデータの分布が密なバケットは多数選ばれ，粗なバケットはあまり選ばれないため，合理的である．そのため，本稿では X とするデータを一様乱数に基づいて選択する．

なお，本研究ではデータを重複登録するため，重複登録によってメモリ使用量が大幅に増加しないようにした．具体的には，距離計算に用いるベクトルデータを保持しておくテーブルをハッシュテーブルとは別に用意し，ハッシュテーブルには各ハッシュ値を持つデータの番号のみを保持した．このことにより，重複登録によって増加するメモリ使用量はデータの番号を表す分のみとなる．

4. 実　　験

4.1 精度，計算時間，メモリ使用量

LSH と提案手法の結果を比較し，提案手法の有効性を確かめる実験をした．実験には Multi-PIE Face database [9] に含まれる 754,200 枚の画像に顔検出 [10] を適用し，得られた 316,089 枚の画像に対して正規化 [11] を施し，928 次元の LBP 特徴 [12] を抽出し，さらに主成分分析によって 100 次元に圧縮した．これらの特徴ベクトルからランダムに 10,000 をデータベース登録用に選び，別の 10,000 をクエリ用とした．

精度は，事前に全探索で求めておいた最近傍データと，それぞれの近似最近傍探索手法で求めた近似最近傍データが一致した割合である．計算時間は，クエリが与えられてから近似最近傍点を出力するまでに必要な時間である．用いた計算機は Opteron 6174 (2.2GHz) である．追加登録に用いるデータ数に関するパラメータ α を $0.001, 0.01, 0.1$ とした．すなわち，それぞれ 10 個，100 個，1000 個のデータを追加登録に用いることを意味する．

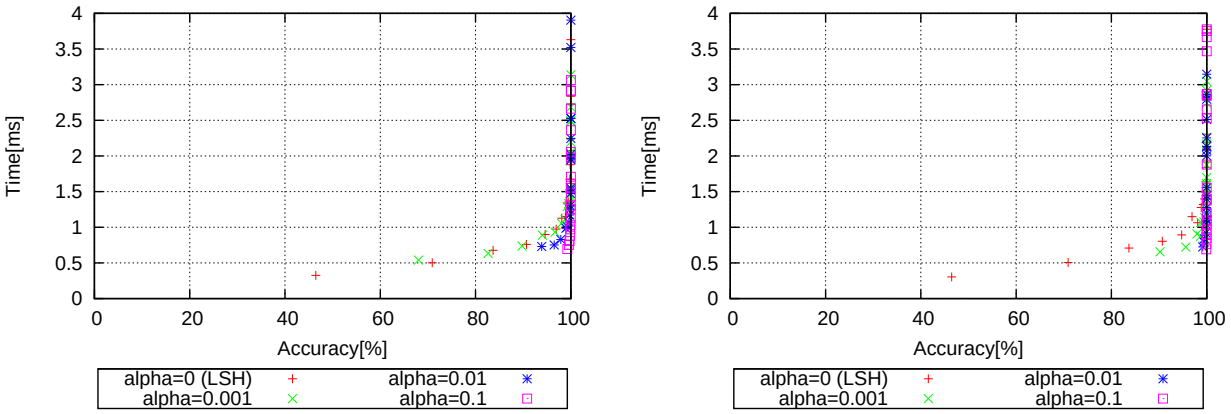
精度と処理時間の関係を表したグラフを図 6 に，精度とメモリ使用量の関係を表したグラフを図 7 に示す．また，表 1 に結果を抜粋する．コピー先ハッシュ関数群のパラメータは w_1，k_1，L_1 と添字 1 をつけて表し，コピー元ハッシュ関数群のパラメータは w_2，k_2，L_2 と添字 2 をつけて表す．

図 6，7 を見ると，提案手法は LSH と比べて，同一の処理時間のときやメモリ使用量のときの精度が著しく向上している．同等の精度のときで比べると，処理時間もメモリ使用量も削減できる事が分かる．表 1 から，$\alpha = 0.1, L_2 = 20, t = 1$ のときに精度 99.9%，計算時間 0.69ms，メモリ使用量 16.4MB を達成している．LSH において $L = 20$ のときは精度 99.9%，計算時間 3.91ms，メモリ使用量 18.3MB であるので，これらを比べると，計算時間が 18%に，メモリ使用量は 90%に削減されたことがわかる．これは，LSH と同等の精度を出すのに必要なハッシュ関数群の数 L が小さくて済む事が大きく寄与していると考えられる．L_2 と t を変化させたときの性能を比べると，L_2 が大きいときのほうが性能が良く，t が大きいときは性能が悪かった．

5. 考　　察

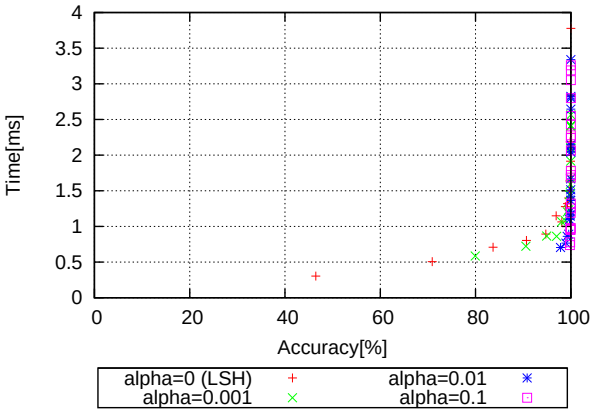
5.1 提案手法による ρ の変化

前節で提案手法の実験的な有効性を確認したが，本節では解析的に提案手法の有効性を示す．具体的には 2.3 で述べた LSH の探索効率の基準 ρ（式 (4)）が提案手法

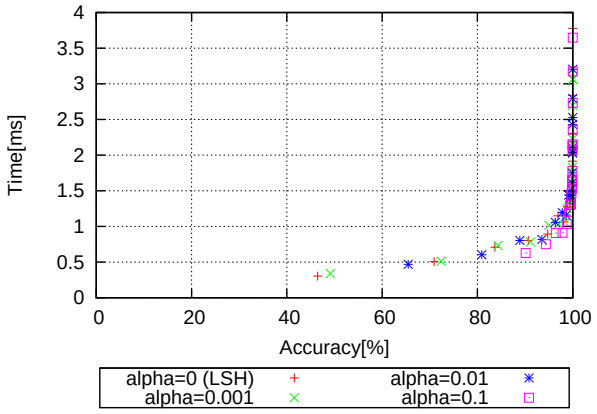


(a) $L_2 = 1$, $t = 1$.

(b) $L_2 = 20$, $t = 1$.



(c) $L_2 = 20$, $t = 5$.



(d) $L_2 = 20$, $t = 10$.

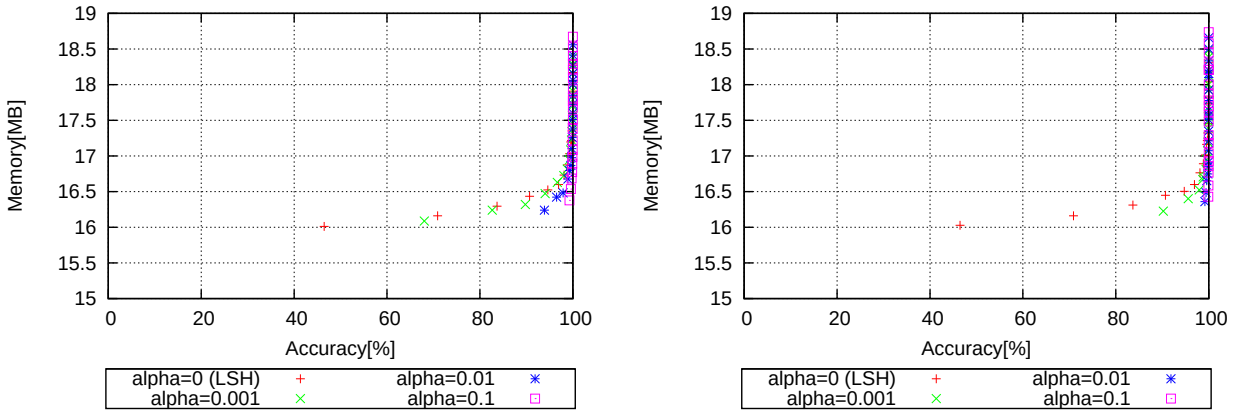
図 6: 精度 vs 処理時間．$k_1 = k_2 = 1$, $w_1 = w_2 = 1000$ は共通で，L_1 を変化させた．

表 1: 精度，計算時間，メモリ使用量の比較．パラメータとして $k_1 = k_2 = 1$, $w_1 = w_2 = 1000$ を用いた．

手法	パラメータ			精度 (%)	計算時間 (ms)	メモリ使用量 (MB)
	α	L_2	t			
LSH($L = 1$)				46.5	0.32	16.0
LSH($L = 20$)				99.9	3.91	18.3
提案手法 ($L_1 = 1$)	0.001	1	1	68.0	0.54	16.0
提案手法 ($L_1 = 1$)	0.01	1	1	93.9	0.73	16.2
提案手法 ($L_1 = 1$)	0.1	1	1	99.3	0.69	16.4
提案手法 ($L_1 = 1$)	0.001	20	1	90.2	0.66	16.2
提案手法 ($L_1 = 1$)	0.01	20	1	99.2	0.72	16.4
提案手法 ($L_1 = 1$)	0.1	20	1	99.9	0.69	16.4
提案手法 ($L_1 = 1$)	0.001	20	5	79.9	0.58	16.2
提案手法 ($L_1 = 1$)	0.01	20	5	97.8	0.70	16.4
提案手法 ($L_1 = 1$)	0.1	20	5	99.8	0.73	16.4
提案手法 ($L_1 = 1$)	0.001	20	10	49.1	0.34	16.0
提案手法 ($L_1 = 1$)	0.01	20	10	65.4	0.47	16.0
提案手法 ($L_1 = 1$)	0.1	20	10	90.1	0.63	16.2

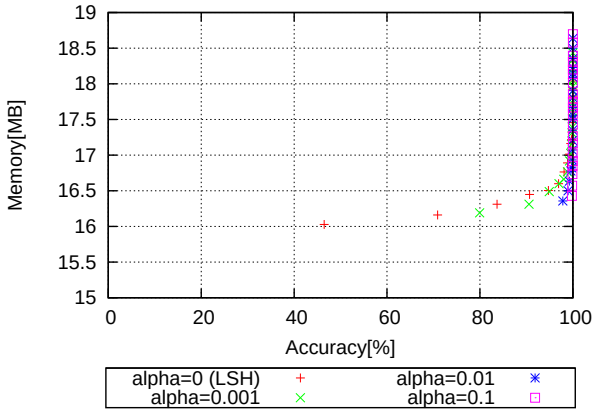
では LSH に比べて小さくなることを示す．

まず，LSH は局所性に鋭敏なハッシュ関数を使用しているため，図 5 の手順 (2) のように注目データの近傍点をコピー元ハッシュ関数群で探索したとき，注目データに近い点ほど高い確率で探索され，遠い点ほど低い確率で探索される．そのため，コピー先ハッシュ関数群ではクエリと同じビンに属するデータが増える．すなわち，p_1 と p_2 は共に上昇する．ただし，注目データに近い領域ではより多くのデータが探索されるようになるため，p_1 は p_2 より大きくなる．これにより，ρ は減少する．

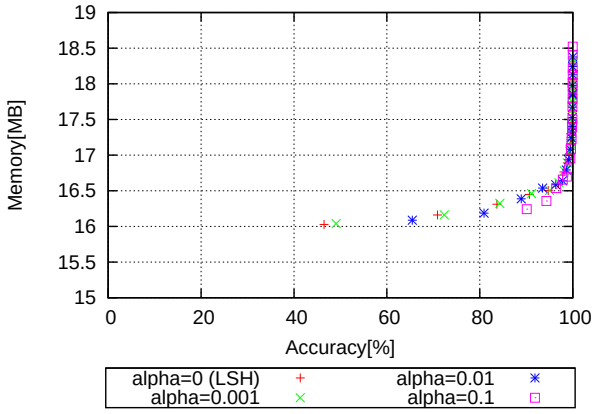


(a) $L_2 = 1$, $t = 1$.

(b) $L_2 = 20$, $t = 1$.



(c) $L_2 = 20$, $t = 5$.



(d) $L_2 = 20$, $t = 10$.

図 7: 精度 vs メモリ使用量．$k_1 = k_2 = 1$, $w_1 = w_2 = 1000$ は共通で，L_1 を変化させた．

5.2 コピー元ハッシュ関数群におけるハッシュ関数群の数 L_2 と閾値 t の関係

図 5 の手順 (3) では探索回数が閾値 t 以上のデータのみを登録する．この閾値をうまく調整することで，近傍点である確率が高い点を選択的に追加し，ρ の減少を促すことができる．本節ではどのような閾値が良いのかを考察する．

コピー元ハッシュ関数群で L_2 個のハッシュ関数群を探索して，あるデータ X が x 回みつかったとする．この事象が起こる確率は次式の二項分布で与えられる．

$$\mathrm{P}(x) = \binom{L_2}{x} p^x (1-p)^{L_2-x} \tag{5}$$

ここで

$$\binom{L_2}{x} = \frac{L_2!}{x!(L_2-x)!} \tag{6}$$

である．また，確率 p は注目データからデータ X までの距離の関数であり，近いほど大きくなる．

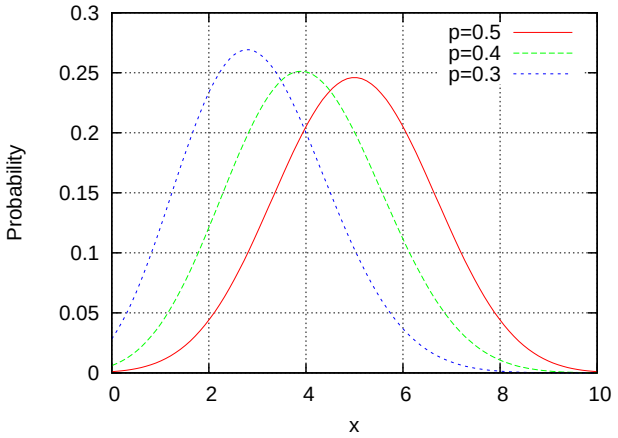
図 8 は式 (5) をプロットしたもので，コピー元ハッシュ関数群で x 回探索される確率を表している．大きな p を持つ X に近い点（例えば $p = 0.5$）と小さな p を持つ X から遠い点（$p = 0.3$）では分布が異なり，近い点のほうが探索される回数が多いことがわかる．

図 9 は閾値 t とデータが選択される確率の関係を表したもの（データが $x \geqq t$ 回以上探索される確率）である．ここで閾値 t を $t = 1$ としたときよりも，$p = 0.5$ のときの期待値 $0.5L_2$ としたとき（すなわち，$L_2 = 10$ では $t = 5$，$L_2 = 20$ では $t = 10$）のほうが近い点と遠い点の確率差は大きいことから，適切に閾値を設定することが必要であること，そしてその値は最近傍点が探索される確率を基準に定められる可能性があることがわかる．

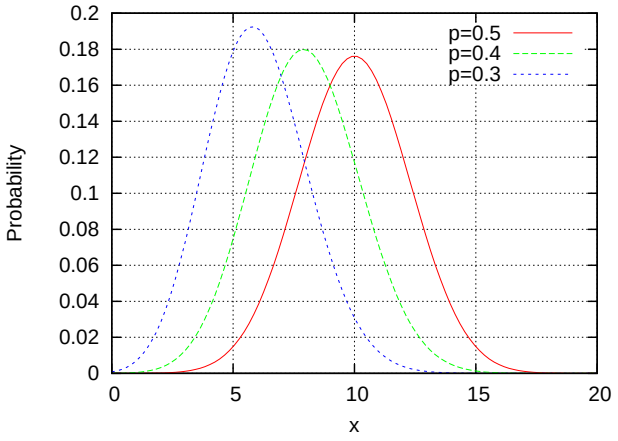
次に，コピー元ハッシュ関数群におけるハッシュ関数群の数 L_2 が提案手法の性能に及ぼす影響を見るために図 9(a) と図 9(b) を比較する．閾値 t を先程と同様に $p = 0.5$ のときの期待値 $0.5L_2$ としたとき，図 9(a) よりも図 9(b) のほうが近い点と遠い点の確率差は大きい．したがって，コピー元ハッシュ関数群におけるハッシュ関数群の数 L_2 を大きくすれば，性能が向上する場合があると考えられる．

6. ま と め

本稿では，ハッシュを用いる近似最近傍探索手法である LSH において，同一精度を実現するために必要な計

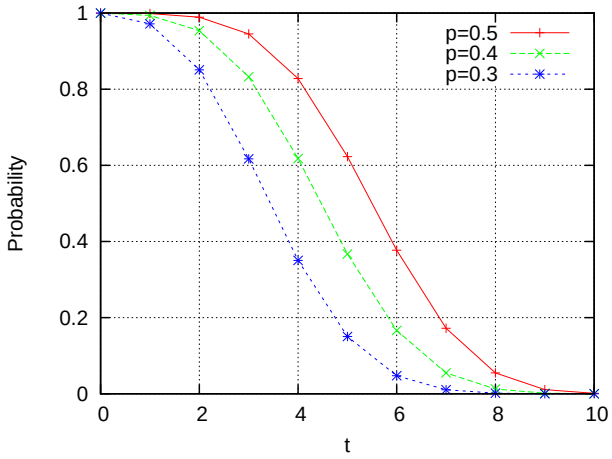


(a) $L_2 = 10$ のとき.

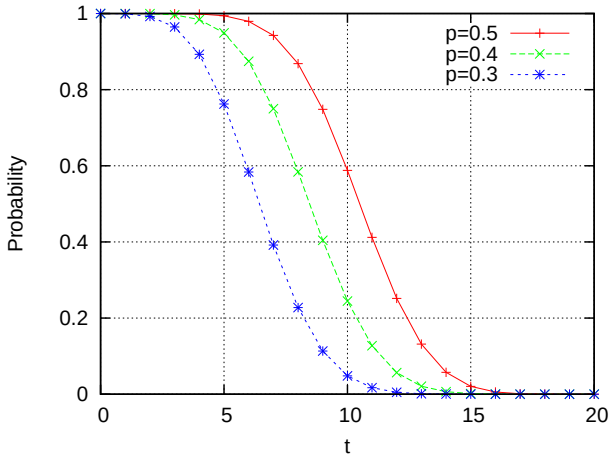


(b) $L_2 = 20$ のとき.

図 8: データがコピー元ハッシュ関数群で x 回探索される確率.



(a) $L_2 = 10$ のとき.



(b) $L_2 = 20$ のとき.

図 9: 閾値 t とデータが選択される確率の関係.

算時間とメモリ使用量を削減する手法を提案した．提案手法はデータベースに登録されているデータを重複登録することでこれを実現している．実験により提案手法の有効性を確認し, [1] で用いられている LSH の探索効率の基準 ρ を用いて解析的に性能が向上する理由を示した．

データを余分に登録することによって性能に悪影響が出るであろうという直感的予測に反して性能が向上した理由は，クエリ（の予測値）の近傍点のみを選択的にバケットに登録する仕組みが働き，真の最近傍点がクエリと同じバケットに入る確率が増えたためである．そして，これによる計算時間とメモリ使用量の上昇がごくわずかであったことが考えられる．解析的には，LSH の探索効率の基準 ρ を減少することができたためと考えられる．

今後の課題としては，提案手法の更なる理論解析ならびに Spectral Hashing 等，LSH 以外の手法への適用が挙げられる．

謝辞 本研究の一部は平成 22 年度科学技術戦略推進費「安全・安心な社会のための犯罪・テロ対策技術等を実用化するプログラム」の一環で実施され，科研費補助金基盤研究 (B)(22300062) ならびに科学技術振興機構 CREST の補助を受けた．株式会社東芝から顔検出と正規化プログラムの提供を受けた．ここに記して感謝する．

文　献

[1] 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.

[2] 黄瀬浩一，野口和人，岩村雅一，“参照特徴ベクトルの増加による低品質画像の高速・高精度認識,” 信学論 D, vol.J93-D，no.8，pp.1353–1363，Aug. 2010.

[3] 片山紀生，佐藤真一，“Sr-tree：高次元点データに対する最近接検索のためのインデックス構造の提案,” 電子情報通信学会論文誌 D，vol.J80-D1，no.8，pp.703–717，Aug. 1997.

[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu, “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions,” Journal of the ACM, vol.45, no.6, pp.891–923, Nov. 1998.

[5] 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.

[6] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” Advances in Neural Information Processing Systems, vol.21, pp.1753–1760, 2008.

[7] A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher, “Min-wise independent permuta-

tions," Journal of Computer and System Sciences, vol.60, pp.630–659, 2000.
[8] 和田俊和, "空間分割を用いた識別と非線形写像の学習: (1) 空間分割による最近傍識別の高速化," 情報処理, vol.46, no.8, pp.912–918, Aug. 2005.
[9] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "Multi-pie," Proc. 8th IEEE Int'l Conf. on Automatic Face and Gesture Recognition, 2008.
[10] T. Mita, T. Kaneko, B. Stenger, and O. Hori, "Discriminative feature co-occurrence selection for object detection," IEEE Trans. PAMI, pp.1257–1269, July 2008.
[11] T. Kozakaya and O. Yamaguchi, "Face recognition by projection-based 3d normalization and shading subspace orthogonalization," Proc 7th Int' Conf. on Automatic Face and Gesture Recognition, pp.163–168, 2006.
[12] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face description with local binary patterns: Application to face recognition," IEEE Trans. PAMI, vol.28, no.12, pp.2037–2041, Dec. 2006.