Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

Resizable-LSH : 可変領域型の近似的類似検索

山﨑邦弘[†]　中村智浩[†]　舟橋卓也[†]　山名早人[††,†††]

本稿では閾値を可変にした近似的な類似検索手法を提案する．近年，距離を用いた類似検索手法の 1 つとして，Locality-Sensitive Hashing（局所性鋭敏型ハッシング，LSH）による近似的な類似検索が注目されている．LSH は，「距離が近い入力同士は高い確率で衝突する」特徴を持つハッシュ関数を用いたデータマッピング手法であり，高次元なデータに対しても高速に近傍検索を行うことができる．しかし LSH では，事前計算によって距離が近いデータ同士を同じハッシュ値にマッピングするため，検索時に類似度の閾値を変更することができない．閾値を変更するにはハッシュテーブルの再構築が必要になるため，ユーザが閾値を指定できるような類似検索は実現困難である．そこで本研究では，類似検索時に，クエリとハッシュ値が一致するデータに加え，ハッシュ値が近いデータも取得することで，ハッシュテーブルの再構築を行うことなく，閾値を指定できる類似検索を実現した．提案手法は，閾値に合わせてハッシュテーブルを逐次再構築する LSH と比較して，同程度の精度で，かつ 1,000 倍程度の高速化を達成できることを実験により確認した．

Resizable-LSH: An Approximate Similarity Search Algorithm for Resizable Range-Search

Kunihiro Yamazaki[†], Tomohiro Nakamura[†],
Takuya Funahashi[†] and Hayato Yamana[††,†††]

We introduce an efficient algorithm named “Resizable-LSH” for approximate similarity search, which enables resizing the search range flexibly. Nowadays, Locality-Sensitive Hashing (LSH) is drawing attention as an efficient algorithm for approximate nearest neighbor search. LSH adopts hash functions that collide with high probability if two vectors are close, so that LSH finds approximate nearest neighbors quickly even if the dataset is high-dimensional. However, LSH should generate hash tables preliminarily, that results in resizing the search range costs expensive because hash table regeneration is required whenever we face the needs to resize search range. To solve the problem, our proposed Resizable-LSH retrieves not only the same hash value of query, but also near hash values. Then Resizable-LSH achieves resizable range-search. As it turns out, the result of the experiments shows Resizable-LSH works about 1,000 times faster than LSH with almost the same quality in comparison with LSH.

1. はじめに

昨今，コンピュータが扱うデジタルデータは急激に増加・複雑化しており，大量に存在するデータの中から有益なデータを効率よく検索する必要が出てきている．キーワードによる検索はユーザの負担が小さいためよく利用されるが，キーワードによる表現の難しい画像などのデータにおいては，主観的な表現の違いも発生しやすい．しかし類似検索であれば，手書き画像や既存の他のデータから類似しているデータを検索することができ，主観的な表現の違いもユーザの負担も小さくて済む．そのため，今後のデータ量の増加・データの複雑化に伴い，類似検索技術の需要はさらに高まっていくことが予想される．

汎用的な類似検索を行う手法としては，類似度を距離に置き換えて類似度の比較を行う手法が主流である[3][7][9]．これは，各データを距離空間内のベクトルで表し，それらベクトルの「距離が近い ＝ 類似度が高い」となるようなデータ構造を構築することで，類似検索を近傍探索で実現する手法である．しかしそれらの多くは特徴量ベクトルが高次元になると計算量が爆発してしまい，線形探索と同程度の速度しか出ないことが知られている（次元の呪い）．そのため，近似解でこれを代用しようという研究がなされている[1][2][5][8]．近似的な近傍探索も次元の呪いの影響を受けるが，P. Indyk らによる Locality-Sensitive Hashing（LSH）[8]の出現以降，線形探索よりも高速な近傍探索が可能になっている．

本稿では，近傍探索型の類似検索における閾値に着目した．類似検索の閾値をユーザが変更可能にすることは，通常の Web 検索の様にランキング順で表示している場合は，あまり意味がない．しかし，よりインタラクティブな表示をする場合（例えば，クエリに似ている画像のみを 2 次元平面状にクラスタリングして視覚的に表示等），ユーザが変更可能な閾値があれば，ユーザのニーズに合わせて検索結果をコントロールできる．ゆえに，閾値を設定した領域探索としての類似検索もまた，近傍探索同様に有用性があると考えられる．そこで本稿では LSH を基に，閾値に合わせた探索を行う Resizable-LSH という手法を提案する．

本稿では以下の構成をとる．まず 2 節で近傍探索に関する関連研究をまとめる．次に 3 節で提案手法について述べた後，4 節で実験と評価を行い，最後に 5 節でまとめと今後の課題を述べる．

† 早稲田大学大学院基幹理工学研究科修士課程
Master’s Course of Graduate School of Fundamental Science and Engineering, Waseda University
†† 早稲田大学理工学術院
Faculty of Science and Engineering, Waseda University
††† 国立情報学研究所
National Institute of Informatics

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

2. 関連研究

2.1 概要

汎用的な類似検索を行う手法として，データを距離空間上の特徴量ベクトルで表し，類似度を距離に置き換えることで類似度の比較を行う手法が多数研究されている[3][7][8][9]．類似度を距離に置き換えると，距離が近いほどデータ同士が類似しているとみなすことができる．これにより，画像や文章等様々なデータに対する類似検索において，対象データの類似度の定義と探索手法とを別個に考案・評価することができる．定義された類似度に対して高速に類似検索を行う手法は，近傍探索問題や領域探索問題に帰着できるため，汎用性が高い．

近傍探索・領域探索に関する研究は，これまでに数多く行われている[3][7][9]．しかし多くの手法では，対象ベクトル空間が高次元になると，計算量が爆発してしまう．これは次元の呪いとよばれ，高次元空間に対しては線形探索と同程度の性能しか実現されていない．そのため厳密解の代わりに近似解を利用して近傍探索・領域探索を行う研究がなされている[1][2][5][8]．近似的な近傍探索もまた次元の呪いの影響を受けるが，P. Indyk らによって Locality-Sensitive Hashing（LSH）[8]が提案されて以降， 線形探索よりも高速な近傍探索が可能になっている．

そこで本節では，LSH とその実装手法について説明した後，本稿が対象としている「閾値を可変にした類似検索」における LSH の問題点について述べる．

2.2 Locality-Sensitive Hashing [8]

LSH とは，ハッシュ関数を用いたマッピングによる類似検索手法である．ここで使われるハッシュ関数とは，「距離が近い入力同士は，高い確率で衝突する」特徴を持つハッシュ関数を指す．このようなハッシュ関数を局所性に鋭敏な（Locality-Sensitive な）ハッシュ関数と呼ぶ．局所性に鋭敏なハッシュ関数を使用すると，距離が近いデータは高い確率で同じ値にマッピングされるようなハッシュテーブルを作成することができる．

局所性に鋭敏なハッシュ関数は，ステップ関数のように「ある一定の距離以下の入力同士のみ必ず同じハッシュ値になる」ことが理想的であるが，実際には難しい．そのため，実際には距離が近いほど高い確率で衝突するようなハッシュ関数を用いる．このようなハッシュ関数を複数用いることで，距離が閾値以上の場合に衝突する確率が急激に下がるようなハッシュテーブル群を作成することができる．

Andoni らが行った[1], ハッシュ関数を複数用いることによる衝突確率の変化の例を図 1 に引用する．横軸がデータ同士の距離を，縦軸はハッシュ値の衝突する確率を示している．$k = 1$, $L = 1$ が元々のハッシュ関数である．ここでは単純に衝突確率が距離に比例しているハッシュ関数[6]を考える．ハッシュ値を k 次元にして，k 個の値全てが一致したときを衝突とすると，距離が遠いにもかかわらずハッシュ値が衝突してしまう確率を下げることができる．また，k を増加させると衝突確率自体が下がってしまうが，ハッシュテーブルを L 個用意して，各テーブルでの検索結果の和集合をとるようにすると，衝突確率を上げることができる．図 1 の $k = 3$, $L = 10$ の破線と $k = 5$, $L = 50$ の実線を比較すると，後者の方が急角度になっており，精度が高くなる．これより，パラメタ k および L を調整することで任意精度の近似近傍探索を実現できることが分かる．ただし，検索コストは L に比例するので，より良い特性を示すハッシュ関数を構成することが，LSH の性能向上において重要である．

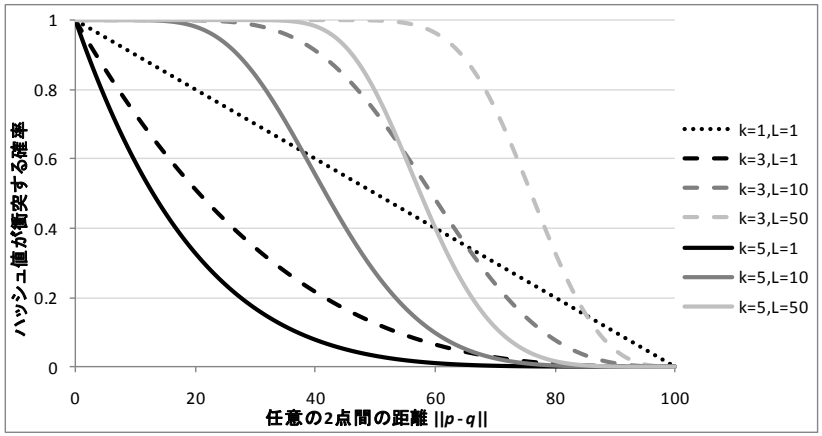

図 1　パラメタ k, L 調整時のハッシュ関数特性変化グラフ
（[1]の Fig.3 より引用．ただしこの図では 2 つのグラフを 1 つに纏めており，さらに各パラメタの値も異なっている．）

2.3 安定分布を使用した LSH [5]

前述の通り，LSH は，対象データセットに対して有効なハッシュ関数をいかにして見つけるかが重要である．そのため，多くのデータセットに適用できる汎用的なユークリッド空間で LSH を実現する実装手法が考案された[5].

[5]では，特性指数が s である安定分布を用いる．ユークリッド空間では l_2 ノルムを距離として用いるので，$s = 2$ の正規分布を用いることになる．具体的には，以下のようなハッシュ関数 $h(\boldsymbol{p})$によって実現する．

$$h(\boldsymbol{p}) = \lfloor (\boldsymbol{a} \cdot \boldsymbol{p} + b) / r \rfloor$$

$\boldsymbol{a}$ ：安定分布に従う独立な乱数値を要素として持つベクトル．$\boldsymbol{a} \in \mathbf{R}^d$
b ：ハッシュ値をランダムにシフトさせるための値．$b \in [0, \omega]$
r ：類似検索において，類似とみなす閾値

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

なお, $\lfloor x \rfloor$ とは床関数であり，x を超えない最大の整数を意味する．このハッシュ関数は, ベクトル $\boldsymbol{a}$ に垂直な超平面によって空間を等間隔に分割していることに等しい．k 次元のハッシュ値を作成するには，乱数を変化させた k 個のハッシュ関数を用意すればよい．図 2 に，このハッシュ関数による $\mathbf{R}^d$ 空間の分割の様子を示す．特徴量ベクトル（グレーの点）がどの区間にあるか，をハッシュ値とすることになる．図中の h の値のようにハッシュ値が決定される．ハッシュ関数を k 個にすると，これらが升目状に組み合わさることになる．

実装が容易で汎用性が高いため，良く用いられる実装方法の 1 つである．

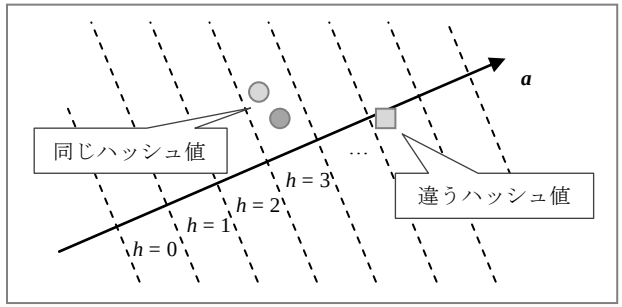

図 2　安定分布を用いたハッシュ関数概略図（$d = 2, k = 1$）

2.4　超球分割を使用した LSH [2]

[2]は，2.3 の方法を更に高速化する方法である．2.3 の方法は，元データ空間 $\mathbf{R}^d$ を k 次元の升目状に分割するため，元データである d 次元ベクトルの各次元の値がハッシュ値に影響を与える度合いが偏ってしまう．すると k の値を大きくすることの効果が小さくなってしまう．

そこで[2]が提案する手法では，まず元の d 次元データを次元削減によって k 次元空間（$k < d$）に射影した後，ランダムに複数の超球を k 次元空間に敷き詰め「次元削減後の特徴量ベクトルがどの座標の超球に入るか」をハッシュ値とする．便宜上この超球をマッピング超球と呼ぶ．

次元削減には，各要素が標準正規分布に従う k 行 d 列の行列 $\boldsymbol{A}$ を用いる．元のベクトル $\boldsymbol{p} \in \mathbf{R}^d$ に対して次元削減後のベクトルを $\boldsymbol{p}' = \boldsymbol{A} \cdot \boldsymbol{p} / \sqrt{k}$ とすれば，k が十分な値の場合，実用上十分なほど高い確率で元の空間における任意の 2 点間の距離を維持したまま次元削減が行える[8]．なお，次元削減を行うのは，空間分割に必要なマッピング超球の数は次元数に対して指数的に増加してしまうためである．

空間分割をするには，マッピング超球はランダムに敷き詰められていた方が良い．しかし完全にランダムにすると検索コストが増大してしまう．そこで，格子状に並んだマッピング超球群をランダムな量シフトさせることで検索コストを抑える手法が採用されている．格子状に並んだマッピング超球群の例を図 3 に示す．各マッピング超球の半径を r，中心点の間隔を $4 \times r$ とする．この格子を各方向へランダムにシフトさせたものを複数用意することで空間を分割する．その後，ベクトルがどの超球内にマップされているかを調べ，当該超球の中心座標（k 次元の実数値）をハッシュ値とする．格子毎にどのマッピング超球が一番近いかは容易に決定できる．これにより，全マッピング超球を探索せずに, 現実的な時間内でハッシュ値を計算することができる．

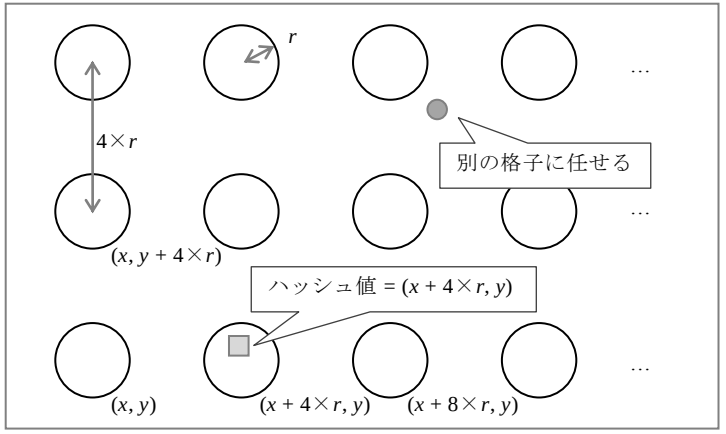

図 3　格子状に並んだマッピング超球群の図（$k = 2$）

2.5　問題点

LSH による手法は，近似的な探索ではあるが，次元数が増加しても時間および空間計算量は指数的ではなく線形に準ずるオーダーで増加する．そのため，高次元なデータに対しても，LSH は線形探索より高速に近傍探索を達成できる．

しかし，LSH は類似度の閾値をもとに事前計算をし，ハッシュテーブルを構築するため，領域探索には不向きである．これは，構築時に指定した閾値に対しては効率よく検索できるが，その閾値を変更することは困難であることを意味する．既存の LSH では，閾値を変更して領域検索を行いたい場合，精度を大きく犠牲にするか，ハッシュテーブル全体を再構築することになり，望ましくない．

以上より，高次元なデータに対して閾値を可変にした類似検索には，LSH はそのままでは実用性に乏しいことが分かる．そこで，2.4 に示した LSH を基に，高次元なデータに対しても閾値を可変にした類似検索を行える手法を次節で提案する．

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

3. 提案手法

3.1 概要

本稿が提案するのは，d 次元ユークリッド空間で表現されるデータセットに対して，任意半径の超球状の領域をもつ近似領域探索をハッシュテーブルの再構築なしに実現する手法である．提案手法では，2.4 と同様の手法によりハッシュテーブルを構築した後，「検索領域に含まれるデータがマップされうるハッシュ値」の一覧を取得することで近似領域探索を実現する．2.4 の手法とは，距離が近似的に維持されるような次元削減の後の k 次元空間にマッピング超球を配置して，データがどの領域にマッピングされるかをハッシュ値とする手法である．

提案手法では，検索領域に含まれるデータがどのハッシュ値にマップされるかを調べ，マップされうるハッシュ値の一覧を取得する．すると，得たいデータは高い確率でそれら一覧のいずれかのハッシュ値をとることになる．よってハッシュ値の計算結果がその一覧に含まれるようなデータを取得すれば，近似領域探索を達成できる．次元削減後の空間では通常データ間の位置関係は維持されないが，領域の形状を超球と限定することにより，ハッシュ値一覧の取得を可能にしている．

次節以降で，ハッシュテーブルの作成と類似検索の実行方法について述べる．

3.2 ハッシュテーブルの作成

ハッシュテーブルの作成手法は，2.4 で説明した手法とほぼ同一である．次元削減を行った後，超球分割によりハッシュ値を決定する．

各要素が標準正規分布に従う k 行 d 列の行列 $\boldsymbol{A}$ を用意し，各特徴量ベクトル $\boldsymbol{p} \in \mathbf{R}^d$ を次元削減する．次元削減後の k 次元ベクトルは $\boldsymbol{p}' = \boldsymbol{A} \cdot \boldsymbol{p} / \sqrt{k}$ となる．

次に，次元削減後のベクトルをマッピング超球によってハッシュ値にマッピングする．ただしここでマッピング超球の半径は，検索超球半径 r と同じ値を用いることができない．なぜなら，r は可変値になるので，事前計算の時点では使えないからである．そのためマッピング超球の半径を ω とおく．なお，ω は r の下界となる．

具体的には，図 4 に示すようにしてハッシュ値にマッピングする．[2]では，図 3 に示すようにハッシュ値としてマッピング超球の中心座標を使っていた．しかし提案手法では「原点に一番近いマッピング超球から見て何番目の超球か」の値をハッシュ値としている．そのため，実際には格子番号 i も合わせた $k+1$ 次元の整数のハッシュ値となる．また，複数の超球が重なっている部分では，[2]同様格子番号の小さい方をハッシュ値とする．以上により，元データからハッシュ値を作成することができる．

3.3 閾値を指定した検索

提案手法は，閾値を指定して検索を行えるようにするために，ハッシュテーブルの検索方法を変更する．閾値 r を指定した領域探索は，クエリから半径 r の範囲内にある全データを検索することに等しい．

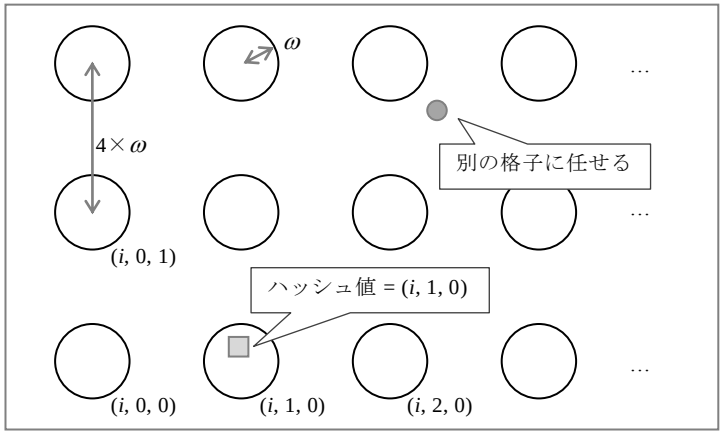

図 4　提案手法のハッシュ値へのマッピング手順説明図

3.2 に示したハッシュテーブルの作成方法について考える．このハッシュテーブルでは，距離が ω より近いデータ同士は高い確率で衝突するようになっている．しかし，距離が ω より遠くても，位置が近ければそれに準ずる確率でハッシュ値が値として近くなっているはずである．k が十分大きな値の場合，元の空間の距離関係を概ね維持したまま次元削減が行えるためである[8].

これより，クエリから半径 r の超球範囲内にある特徴量ベクトルは，次元削減後も十分高い確率で半径 r の超球範囲内にあると考えられる．そのため，半径 r の超球範囲に含まれる任意の特徴量ベクトルは，この超球と重複部分のあるマッピング超球内に高確率で存在していることになる．すなわち，次元削減後のクエリを中心とした半径 r の超球と重複部分のあるマッピング超球を列挙することができれば，閾値を指定した検索が達成できるということになる．

ここで，格子の内でクエリベクトルに一番近いマッピング超球の中心を原点とみなし，各超球の中心の間隔 $4\times r$ を 1 に縮小すると，図 5 のように，中心 $\boldsymbol{q}'$ 半径 r' の超球に含まれる点のうち各座標が整数である点を探す問題に置き換えて考えることができる．

格子状のマッピング超球群を $G(\boldsymbol{e})$で表す．ここで，格子の間隔は $4\times\omega$で，原点に一番近い超球の中心座標を $\boldsymbol{e} = [0,\ 4\times\omega]^k \in \mathbf{R}^k$ とする．すると，各マッピング超球の中心座標は $\boldsymbol{e} + 4\times\omega\times\boldsymbol{c}$ $(^{\forall}\boldsymbol{c} \in \mathbf{Z}^k)$で表現できる．特にクエリ $\boldsymbol{q} \in \mathbf{R}^k$ に一番近いマッピング超球の中心座標を $\boldsymbol{n} = \boldsymbol{e} + 4\times\omega\times\boldsymbol{z}$ $(^{\exists}\boldsymbol{z} \in \mathbf{Z}^k)$とおく．このとき，任意の点 $\boldsymbol{p} \in \mathbf{R}^k$ を $\boldsymbol{p}' = (\boldsymbol{p}-\boldsymbol{n}) / (4\times\omega)$ へと変換する．すると，ハッシュ値一覧を取得する問題は，中心 $\boldsymbol{q}' = (\boldsymbol{q}-\boldsymbol{n}) / (4\times\omega)$，半径 $r' = r / (4\times\omega)$の超球に含まれる，各座標が整数である点を

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

探す問題に帰着することができる．見つかった整数点に $\boldsymbol{z}$ を足すと，実際のハッシュ値を得ることができる．以上により，この問題を解くことでハッシュ値の一覧を取得することができる．

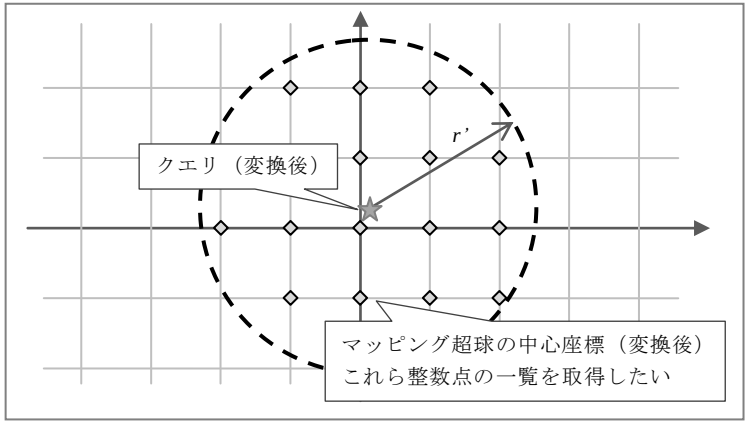

図 5　整数点を探す問題の概略図（$k=2$）

3.4 ハッシュ値一覧の取得

3.3 に示した問題を単純に解こうとすると，計算時間が現実的ではないことが分かる．それは，次元削減後の次元数 k および半径比 $r\ /\ \omega$ の値が共に大きい場合（$k = 10$ など，元データの次元数 d に比べれば十分小さいような値であっても），取得すべきハッシュ値の量が膨大になってしまうためである．

そこで，検索領域が超球状であることを利用し，「ハッシュ値一覧」の代わりに「ハッシュ値の範囲のリスト」を取得するようにする．具体的には，超球内を超立方体で再帰的に分割する．分割の概略を図 6 に示す．超立方体であれば，後述の 3.5 および 3.6 の方法により，現実的な時間内で検索が達成できる．

格子状のマッピング超球 $G(\boldsymbol{e})$毎に，図 6 に示す方式で超球を分割していくことで，「検索領域の超球に含まれるハッシュ値一覧」を「いずれかの超立方体に含まれるハッシュ値一覧」として取得することができる．超球の場合はベクトル毎に中心からの距離を計算する必要があるため全ハッシュ値一覧が必要となるが，超立方体の場合は各要素について 1 次元的な範囲比較を繰り返すことによって，全ハッシュ値一覧そのものを用意せずとも領域検索が可能となる．

例えば $k = 2$ の時，「i 番目の格子に対して，ハッシュ値が矩形(0〜5, 1〜2)の範囲」と表現すると，次の 12 個のハッシュ値情報をこの情報だけで表現できることになる．

(i, 0, 1),	(i, 1, 1),	(i, 2, 1),	(i, 3, 1),	(i, 4, 1),	(i, 5, 1),
(i, 0, 2),	(i, 1, 2),	(i, 2, 2),	(i, 3, 2),	(i, 4, 2),	(i, 5, 2),

取得すべき情報量が小さくなるので，k および $r\ /\ \omega$ の値が共に大きい場合でも現実的な時間で取得・検索できるようになる．これを各格子 $G(\boldsymbol{e})$に対して行うことで，ハッシュ値の一覧を「ハッシュ値の範囲のリスト」として取得することができる．

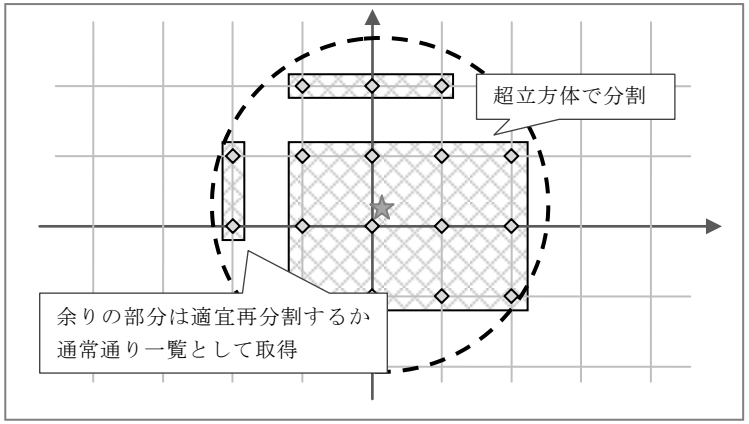

図 6　超立方体の配置による超球の分割概念図

3.5 データの格納方式

3.4 節にて示した通り，提案手法では，ハッシュ値の範囲を示す超立方体から，マッチするデータの一覧を取得する必要がある．通常のハッシュテーブルの構造をとると，ハッシュ値一覧の総量が膨大になった場合，探索する回数もそれに比例して膨大になってしまう．そのため，3.4 で取得したハッシュ値の範囲のリストを用いて効率よく検索・値の取得ができるような構造が求められる．

そこで提案手法では，図 7 に示すような深さ $k+1$ の多分木を構築する．深さ i のノードがハッシュ値の i 次元目の要素に対応している．すなわち，根ノードがハッシュ値 $\boldsymbol{h} \in \mathbf{Z}^{k+1}$ の第 0 要素 h_0, 根の子ノードが $\boldsymbol{h}$ の第 1 要素 h_1, さらにその子ノードが h_2, …と対応している多分木である．根ノードからの深さが i であるようなノードを T_i と表記すると，根ノード T_0 は格子状のマッピング超球 $G(\boldsymbol{e})$の通し番号をキーとしており，また葉ノード T_k は子ノードではなく元データの配列を格納していることになる．各ノード T_i は，指定したハッシュ値の範囲内となる子ノードの一覧を取得できるように実装する．例えば「$h_i = 1$〜3 の範囲の子ノード一覧」等が取得できるようにする．これはソートされた配列によって容易に実現できる．

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

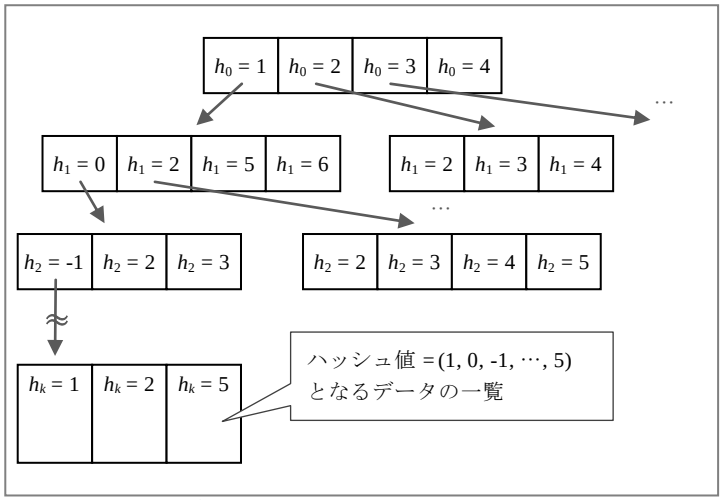

図 7　多分木型ハッシュテーブル構造概念図

3.6 ハッシュ値の範囲からの検索手順

3.4 で取得したハッシュ値の範囲のリストを用いた検索手順をフローチャートにまとめると，図 8 のようになる．図中で，R_n は n 番目の格子状マッピング超球 $G(\boldsymbol{e}_n)$から取得した「ハッシュ値の範囲のリスト」を，T_i はノードを意味する．また，$T_i[h_{i\_a}, h_{i\_b}]$ とは「$h_i = h_{i\_a} \sim h_{i\_b}$ となる範囲の子ノード一覧」を表している．ここで R_n は以下のように表される．先頭の n は格子番号である．

$$R_n = [n, (h_{1\_a}, h_{1\_b}), (h_{2\_a}, h_{2\_b}), \ldots, (h_{k\_a}, h_{k\_b})]^{k+1}$$

ハッシュテーブルが持つ全ての $G(\boldsymbol{e}_n)$および $G(\boldsymbol{e}_n)$に対する全ての R_n に対して 図 8 の操作を繰り返すことで，目的としている検索領域に含まれるデータを高い確率で取得することができる．この確率は，次元削減後の次元数 k とハッシュテーブル数 L を適当に調節することで高くすることができる．

4. 評価

4.1 実験内容

提案手法を評価するために，単純な類似画像検索システムを作成した．実験に使用した計算機の諸元は表 1 のとおりである．入力データには，Web クローリングによって取得した 484,010 枚の画像から 66 次元の特徴量ベクトルを抽出して用いた[10].

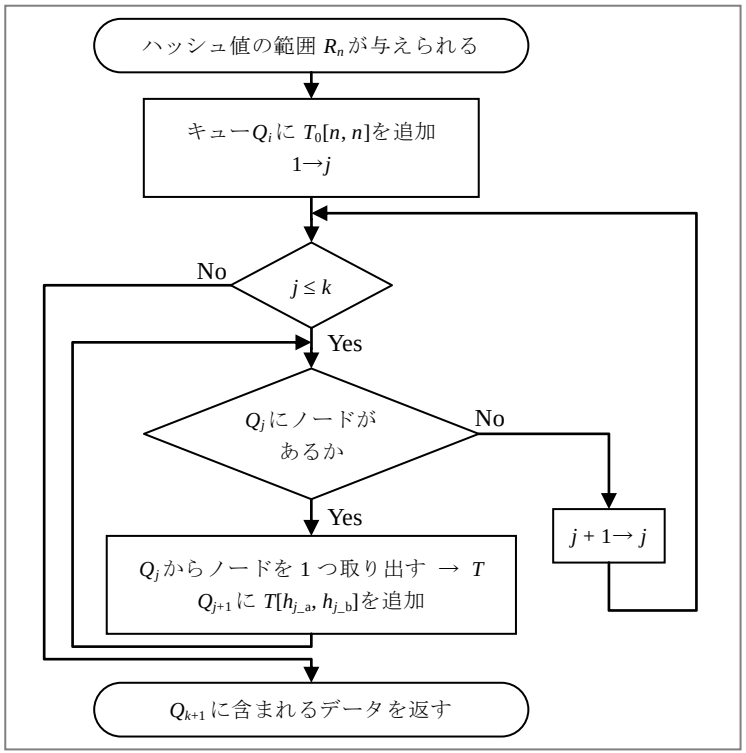

図 8　ハッシュ値範囲からのデータ取得フローチャート

実験は，閾値すなわち検索半径 $r = 0.5$ の場合について検索を行う．提案手法では，基準検索半径$\omega = 0.1$ としてデータ構造を構築した．比較対象としては，2.4 で説明した超球分割を使用した LSH を用いた．既存手法では指定した閾値 $r = 0.5$ に合わせてハッシュテーブルセットを逐次再構築しながら検索を行う．

表 1　実験に使用した PC のスペック表

CPU	Intel® Xeon® CPU X5355 2.66GHz
Memory	16 GB
Sequential Read	399 MB/s
Sequential Write	733 MB/s

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

評価のため，ハッシュ値の次元数 k および使用テーブル数 L を変化させ，速度と精度の変化を計測した．精度の評価尺度としては，適合率，再現率と F 値を使用した．

4.2 使用した特徴量

66 次元の特徴量は，[10]が用いた手法と同様にして作成した．図 9 のように，CIE-Lab 色空間[4]による色ヒストグラムの情報 16×3 次元と，エッジ方向スペクトラムの情報 18 次元を合わせたものである．CIE-Lab 色空間とは，明度 L*と 2 つの色相 a*，b*の 3 つの値で色を表現する方式である．色ヒストグラムは，各値 L*, a*, b*別に，画像内のピクセル値を 16 段階で分けてピクセル数を数えることで求める．一方エッジ方向スペクトラムは，X 方向・Y 方向微分フィルタによってエッジの方向と強さ（勾配）を計算し，エッジの方向を 18 段階でわけて勾配を合算することで求める．各ヒストグラムとスペクトラムは，合計が 1 になるように正規化されている．

なお，本実験は実際の類似画像検索システムとしての精度ではなく，特徴量ベクトルによって決定された正解セットをもとに提案手法を評価しているため，特徴量自体の精度は実験結果には影響しない．

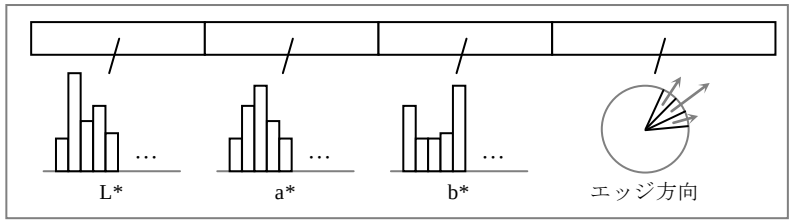

図 9　特徴量ベクトル内訳図

4.3 結果

結果は図 10～図 14 のようになった．各グラフは共通して，既存手法を点線，提案手法を実線で示してある．

図 10 は検索速度を表している．横軸が L，縦軸が検索にかかった秒数（対数表示）である．k = 1, 3, 5 の場合について L を 1～50 まで変化させた場合の，既存手法と提案手法の検索速度を比較している．

図 11，図 12，図 13 は検索精度を表している．横軸が L，縦軸がそれぞれ適合率，再現率，F 値である．k = 1, 3, 5 の場合について L を 1～50 まで変化させた場合の，既存手法と提案手法の検索精度を比較している．

図 14 もまた検索精度を表している．こちらは図 13 と違い横軸が k，縦軸が F 値となっている．L = 10, 30, 50 の場合について k を 1～5 まで変化させた場合の，既存手法と提案手法の検索精度を比較している．

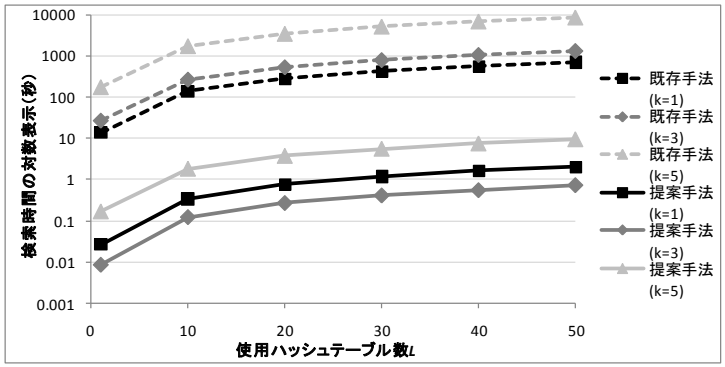

図 10　既存手法と提案手法とのテーブル数－検索時間所要比較グラフ

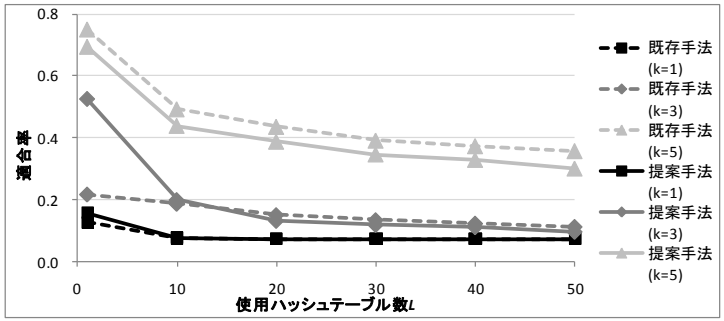

図 11　既存手法と提案手法とのテーブル数－適合率比較グラフ

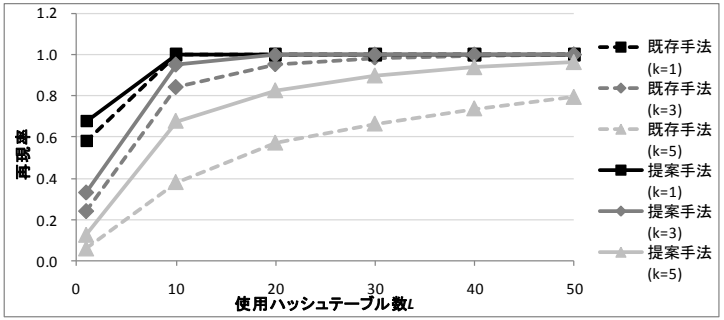

図 12　既存手法と提案手法とのテーブル数－再現率比較グラフ

Vol.2009-DBS-148 No.22
Vol.2009-FI-95 No.22
2009/7/28

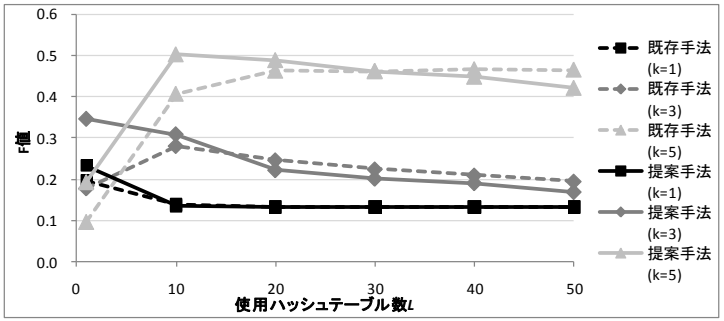

図 13　既存手法と提案手法とのテーブル数－F 値比較グラフ

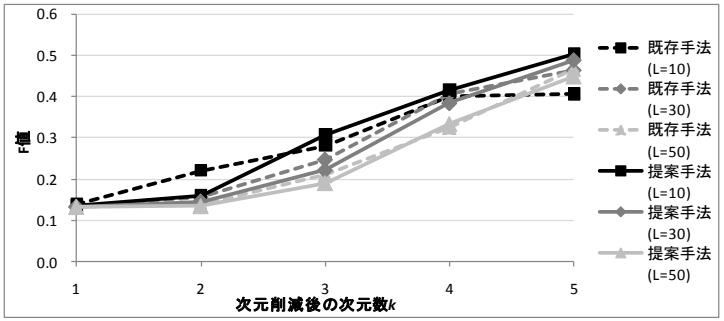

図 14　既存手法と提案手法との次元削減後次元数－F 値比較グラフ

4.4 考察

図 10 を見ると分かるように，同じ次元削減後の次元数 k において，提案手法は既存手法の 1,000 倍以上の速度を示している．また図 13 を見ると，既存手法と比較して提案手法は F 値が低下していないことも読み取れる．よって提案手法は，精度を変化させずに 1,000 倍以上の高速化を実現できたと言える．その一方で，試行したパラメタの範囲が小さいために，図 11 が示すようにハッシュテーブル数 L が大きいとき適合率が低くなってしまっており，その影響で F 値自体が低い値になってしまった．しかし，図 14 から，k の値を増加させると F 値が上昇することが分かる．そのため，パラメタ k および L の値を適切に調整することができれば精度の問題は改善できると考えられる．また，図 12 を見ると，L が十分大きな値であれば，取得したいデータが取得できていることも分かった．これより，false positive を下げるようにパラメタ k および L を調整することが F 値の改善にとって重要であることが分かる．

5. おわりに

本稿では，Locality-Sensitive Hashing を基にして任意半径の超球状の領域をもつ近似領域探索を実現する手法 Resizable-LSH を提案した．提案手法により，コストのかかるハッシュテーブルの再構築をせずに現実的な時間で閾値のある類似検索を行うことができた．また提案手法では，既存手法と比較して検索精度の低下を発生させずに高速化を達成した．しかし，実験に用いた次元削減後の次元数が小さいために，F 値が 0.5 前後と低い値に留まってしまった．また，比較対象が LSH のみであり，他の手法との比較ができていない点も問題である．今後は，他の手法とも比較した上で，精度を向上させるため，より高い次元数に対して本手法の適用及び改良を行う予定である．

謝辞　本研究の一部は科学研究費補助金（特定領域研究）「情報爆発に対応する高度にスケーラブルなモニタリングアーキテクチャ」（課題番号 18049068）によるものである．

参考文献

1) Andoni, A. and Indyk, P.: “Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions,” Comm. of ACM, Vol.51, No.1, pp.117-122 (2008).
2) Andoni, A. and Indyk, P.: “Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions,” Proc. of 47th IEEE FOCS, pp.459-468 (2006).
3) Bentley, J.L.: “Multidimensional Binary Search Trees Used for Associative Searching,” Comm. of ACM, Vol.18, No.9, pp.509-517 (1975).
4) Connolly, C. and Fliess, T.: “A Study of Efficiency and Accuracy in the Transformation from RGB to CIELAB Color Space,” IEEE Trans. on Image Processing, Vol.6, No.7, pp.1046-1048 (1997).
5) Datar, M., Immorlica, N., Indyk, P. and Mirrokni, V.S.: “Locality-Sensitive Hashing Scheme Based on p-Stable Distributions,” Proc. of 20th ACM SCG, pp.253-262 (2004).
6) Gionis, A., Indyk, P. and Motwani, R.: “Similarity Search in High Dimensions via Hashing,” Proc. of 25th VLDB, pp.518-529 (1999).
7) Guttman, A.: “R-trees: A Dynamic Index Structure for Spatial Searching,” Proc. of ACM SIGMOD Int’l Conf. on Management of Data (ICMD), Vol.14, No.2, pp.47-57 (1984).
8) Indyk, P. and Motwani, R.: “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality,” Proc. of 30th ACM FOCS, pp.604-613 (1998).
9) 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-I，Vol.J80-D-I，No.8，pp.703-717 (1997).
10) 馬越健治, 糟谷勇児, 山名早人;“大規模画像 DB からの類似画像検索による改変画像検出，” DEWS2007，L1-3，pp.1-8 (2007).