Vol.2010-AL-131 No.5
2010/9/22

Resizable-LSH による 閾値可変の近似的類似検索手法の高速化

山﨑邦弘† 山名早人††,†††

本稿では，筆者らがこれまでに提案している Resizable-LSH を改善し，広域な閾値に対しても類似検索を可能にする手法を提案する．近年，類似度の尺度として距離を用いた高速な類似検索手法の 1 つとして，Locality-Sensitive Hashing（局所性鋭敏型ハッシング, 以下 LSH）による近似的な類似検索が注目されている. LSH 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用いたデータマッピング手法である．しかし事前計算の際に類似度の閾値を固定とすることで高速化しているため，エンドユーザが閾値を変更しながら類似検索をすることはできない．そこで Resizable-LSH ではハッシュ値が近いデータも検索することで，閾値を可変とした類似検索を実現しているが，閾値が大きくなると検索量も膨大になるため，速度低下を招いていた．本稿では Resizable-LSH のハッシュ値検索方式を変更してハッシュ値探索とデータ探索を同時に行う手法を提案する．本手法により，従来の Resizable-LSH と比較して 800 倍以上の速度向上を達成し，広域な閾値を設定した場合でも高速な類似検索を実現する．

Speed-Up of Resizable-LSH for Similarity-Based Range Query

Kunihiro Yamazaki† and Hayato Yamana††,†††

In this paper, we improve our previously proposed Resizable-LSH that enhances the range query on approximate similarity search faster. Nowadays, Locality-Sensitive Hashing (LSH) is drawing attention as an effective 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 uses fixed search range on generating hash tables, so resizing the search range costs expensive. As a solution, we've proposed the algorithm named Resizable-LSH that achieves resizable range-search. Resizable-LSH retrieves not only the same hash value of query, but also near hash values. Even though, these hash retrieving methods have a drawback in expanding search range. More specifically, large-range causes slowdown of searching. To solve the problem, we bring forward the remediation of Resizable-LSH, which allows searching hash value and data concurrently. The experiments show our improved Resizable-LSH works over 800 times faster than plain Resizable-LSH without any quality reduction.

1. はじめに

昨今，コンピュータが扱う電子データは急激に増加・複雑化しており，大量の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．キーワード検索はユーザの負担が小さいためよく利用されるが，画像などのデータに対しては，キーワードでは主観的な表現の違いが発生しやすい．しかし類似検索は，手書き画像や既存の他のデータから類似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. そのため今後類似検索技術の需要は高まっていくと予想される．

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検索を行う手法が主流である[3][7][10]. 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）. そのためこれを近似解で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，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．

LSH はハッシュを使うため近傍探索に強いが，従来手法[3][7][10]で可能であった閾値を指定しての範囲検索には向いていない．閾値をユーザが変更可能になると，ランキング順でないよりインタラクティブな表示（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表示等）をする際に，ユーザがニーズに合わせて検索結果を調整できるようになる，という利点がある．そのため，我々は閾値を指定可能とした類似検索手法として Resizable-LSH をこれまでに提案している[11]. これにより, 高次元なデータに対しても現実的な時間で閾値指定の類似検索が可能となった．

しかし, Resizable-LSH ではクエリとともに閾値を指定することで類似検索を行うが, 閾値を大きくしていくと速度が著しく低下していく，という問題点がある．極端にいえば，閾値を無限大にすれば時間も無限大にかかってしまう．そこで本稿では，Resizable-LSH のハッシュ値検索方式を変更することで, 巨大な閾値が指定された際にも現実的な時間内での類似検索を可能とする手法を提案する．

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 Resizable-LSH と，提案内容である Resizable-LSH の更なる高速化手順について述べた後，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

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][10]．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][10]．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，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，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[1]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．

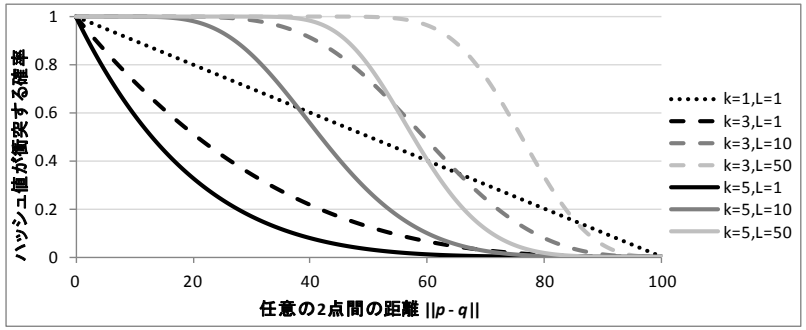

図 1　パラメタ k, L 調整時のハッシュ関数特性変化グラフ
（[1]の Fig.3 より引用．ただしこの図では 2 つのグラフを 1 つに纏めており，さらに各パラメタの値も異なっている．）

2.3 超球分割を使用した LSH [2]

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[2][5].

[5]は安定分布を用いて元データ空間 $\mathbf{R}^d$ を k 次元の升目状に分割する手法である．しかし升目状に分割して k 次元にするため，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 2 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．

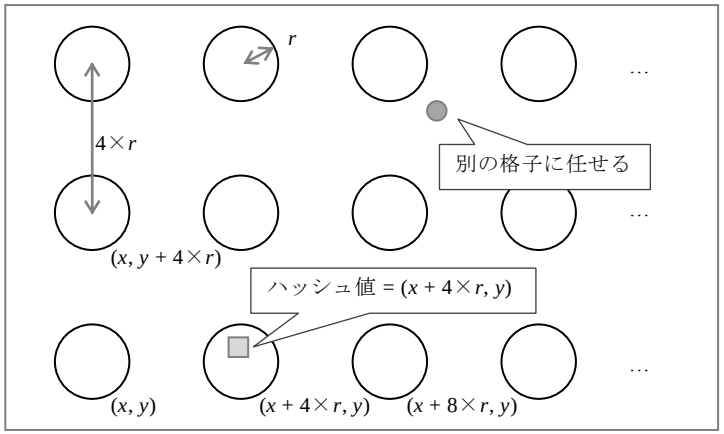

図 2　格子状に並んだマッピング超球群の図（$k=2$）

2.4 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，2.3 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．

3. Resizable-LSH とその改善方式

3.1 概要

本稿で提案する改善方式は，Resizable-LSH[11]を基にしており，特にハッシュ値の算出方法は同一である．また Resizable-LSH は，超球分割を使用した LSH を基にした，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．2.3 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」を順次辿ることで近似領域探索を実現する．その探索方法は，旧来の Resizable-LSH では予め類似データがとりうるハッシュ値候補を全て取得していたが，本改善手法では，候補の列挙は行わずに，ツリー上に保存されたハッシュ値群を直接辿ることによって速度改善を図っている．本節では Resizable-LSH のハッシュテーブルの作成方法と，改善された類似検索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，2.3 で説明した手法とほぼ同一である．Resizable-LSH のハッシュ値作成方式を図 3 に示す．図 3 に示すように，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．相違点としては，検索の効率化のため，ハッシュ値自体を整数の k +1 次元ベクトルとしている点にある．

まず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

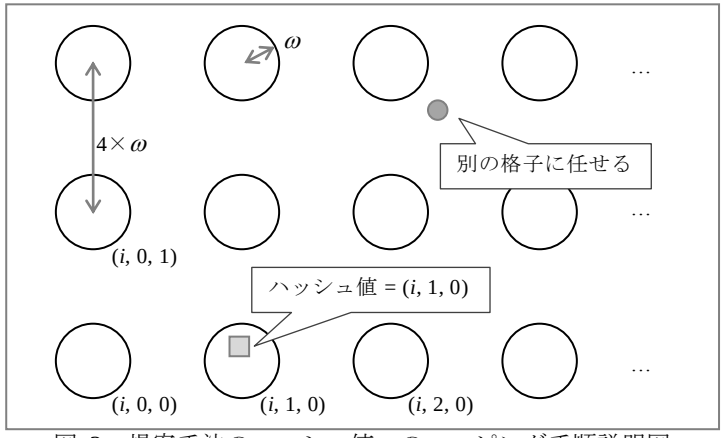

図 3　提案手法のハッシュ値へのマッピング手順説明図

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．

マッピングされた超球からハッシュ値が決定される．「マッピングされた超球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．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

Resizable-LSH では，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 LSH のものとは変更している．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．

3.2 に示したハッシュ関数では，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．しかし同時に距離が ω より遠くても，位置が近ければそれに準ずる確率で近いハッシュ値に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次元削減後のクエリベクトル $\boldsymbol{q}$ から半径 r の範囲内にあると考えられる．そのため，取得したい特徴量ベクトル群は，$\boldsymbol{q}$ から半径 r 内にあるマッピング超球の中に高確率で存在していることになる．すなわち，$\boldsymbol{q}$ 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．

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 $4\times\omega$ を 1 に縮小すると，図 4 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．各超球の中心の間隔を 1 に縮小することで，変換された空間とハッシュ値の座標系の縮尺が等しくなる．ハッシュ値の値が 1 ずれることと，変換された空間で 1 ずれることが等しくなる．

問題を定式化して考える．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．

3.4 ハッシュ値一覧の取得とその問題点

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/ω 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．

そこで旧来の Resizable-LSH[11]では，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」を取得するようにしている．

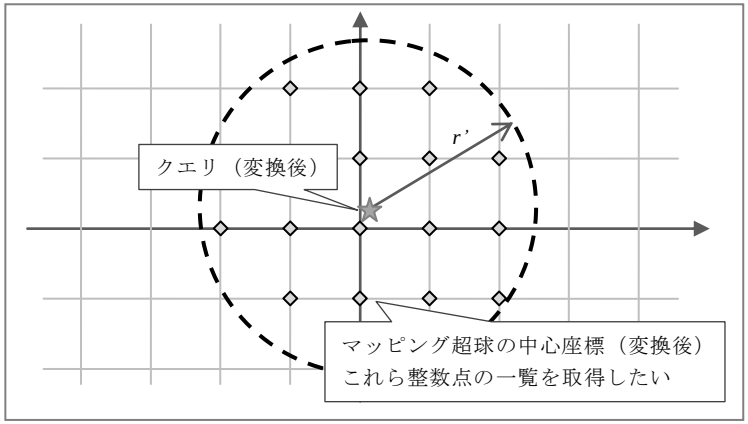

図 4　整数点を探す問題の概略図（$k=2$）

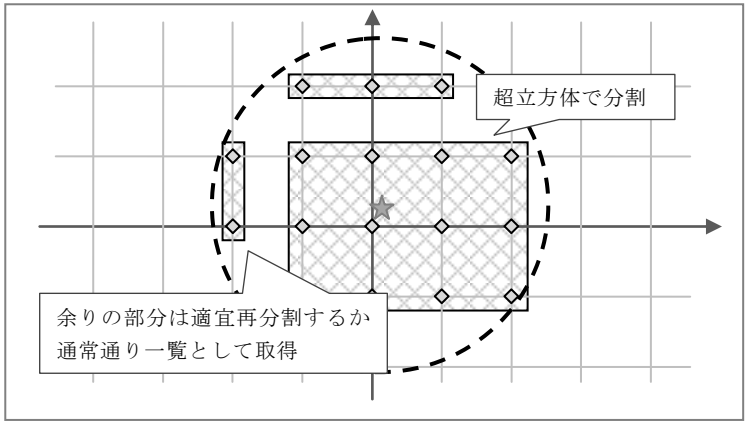

図 5　超立方体の配置による超球の分割概念図

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 5 に示す．超立方体であれば，左上と右下の 2 点の情報だけで，図の網目で記された多数のハッシュ値を表現できる．これにより，旧来の Resizable-LSH では r/ω の値が大きい場合でも現実的な時間で取得・検索できるようにしていた．

しかしこの方法には問題点が 2 つある．最初の問題は，実際にはデータが含まれていないようなハッシュ値も列挙されているという点であり，二番目の問題は，次元削減後の次元数 k が大きい場合超立方体による分割の効率が低下するという点である．

実際にデータをハッシュテーブルに格納した場合，全てのハッシュ値に均一にデータが入っていることはない．k 次元空間は無限だがデータ数は有限であるからである．しかし Resizable-LSH では事前にハッシュ値を列挙するため，指定される閾値が大きければ大きいほど，データの量にかかわらず列挙するハッシュ値の量が増加し，遅くなってしまう．これが最初の問題である．例えば閾値を無限大に設定した場合，Resizable-LSH では無限大に近い時間が必要になる．しかし実際には，全データを返せばよいので, データセットを考慮しながら検索すれば有限時間で終えることができる．

また, Resizable-LSH では検索超球内を超立方体で分割することで高速化をしているが，超球に内接する超立方体の体積は，次元数 k が大きくなるにつれて急速に小さくなる．それにより，超立方体で超球内を分割しても，ほとんどの部分が超立方体の外部となってしまう，これが二番目の問題である．k 次元空間において，単位超球の体積は $\pi^{k/2}/\Gamma((k/2)+1)$，単位超球に内接する超立方体の体積は $(2/\sqrt{k})^k$ である．ここで Γ とはガンマ関数である．これらをグラフにして比較した物を図 6 に示す．図中の体積比とは，単位超球の体積を 1 とした場合の内接超立方体の体積である．グラフからも明らかなように，高次元化に伴い，内接超立方体の体積は単位超球の体積と比較して指数的に小さくなる．そのため，k が大きくなると超立方体 1 つ 1 つの中に含まれるハッシュ値の個数が少なくなってしまい，効率が低下してしまう．

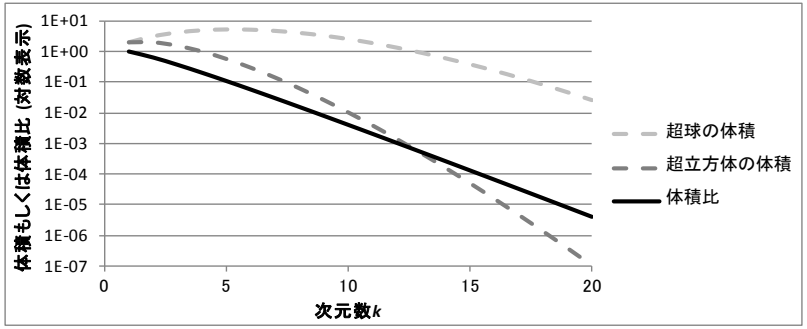

図 6　次元数に対する内接超立方体の体積比較グラフ

3.5 データの格納方式

本稿が提案する改善手法では，前述の問題点を解消するために，ハッシュ値候補を探索しながら同時にデータを探索する．データを格納している木を辿りながら同時に目的ハッシュ値か判定することで不要なハッシュ値を列挙してしまう問題を解消し，さらに判定時に超立方体ではなく超球を用いることで分割効率低下の問題を防ぐ．

データの格納方式は，Resizable-LSH と同一である．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．

なお，もしも 2.3 と同じハッシュ関数を用いた場合，ハッシュ値が実数のベクトルとなるので，各ノードから分かれる子ノードの量が肥大化してしまい．各ノードが持つ情報量が大きくなってしまう．Resizable-LSH ではハッシュ値ベクトルを 1 次元増やして整数値化することにより，探索をより容易にできるようにしている．

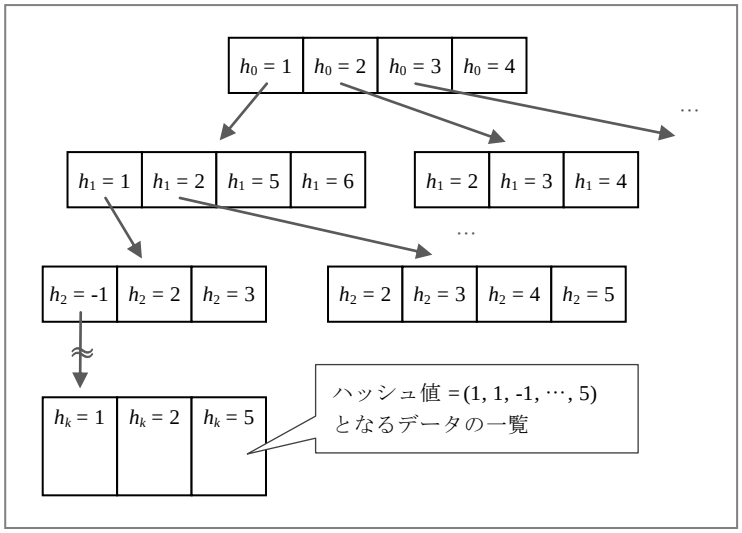

図 7　多分木型ハッシュテーブル構造概念図

3.6 ハッシュ値の検索手順

以下では，本稿の改善内容の中核である，ハッシュ値の検索手順について説明する．

図 7 の根ノードは，マッピング超球の格子の格子番号を表している．そのため根ノードの子ノード以降について考える．クエリのハッシュ値ベクトルを $\boldsymbol{q} \in \mathbf{Z}^{k+1}$，検索したい閾値を r とする．q_0 は格子番号であるのでこれも除外する．木の探索方法を図 8 に示す．任意のハッシュ値 $\boldsymbol{h}$ を考えた時，$\boldsymbol{q}$ との距離は，少なくとも $|q_1 - h_1|$ 以上であることは自明である．他の各要素が同じ値であった場合 $|q_1 - h_1|$ がそのまま $\boldsymbol{q}$ と $\boldsymbol{h}$ の距離になるからである．これより，クエリとの距離が r 以下となるようなハッシュ値は，全て $|q_1 - h_1| \le r$ となる．このことを利用すると，各ノード T_1 から $(q_1 - h_1)^2 \le r^2$ となる h_1 が指す子ノードを取得すれば，最終的に取得したいデータはそのノード以下にあることになる．2 段目以降も同様にして，$(q_1 - h_1)^2 + (q_2 - h_2)^2 \le r^2$ となる h_2 を図 8 に示すように再帰的に幅優先探索で辿ればよい．2 段目の各ノード T_2 は対応する h_1 の値が異なるが，探索時には各 T_2 は自分が対応している h_1 の値を知っているので，単純に木の幅優先探索を行うことで最終的に目的のデータを取得できる．しかしこのままでは，各ノード T_i から範囲内の子ノード一覧を取得する際に毎度根ノードまで順次辿り $h_1 \sim h_{i-1}$ を取得する必要があり，効率が悪い．

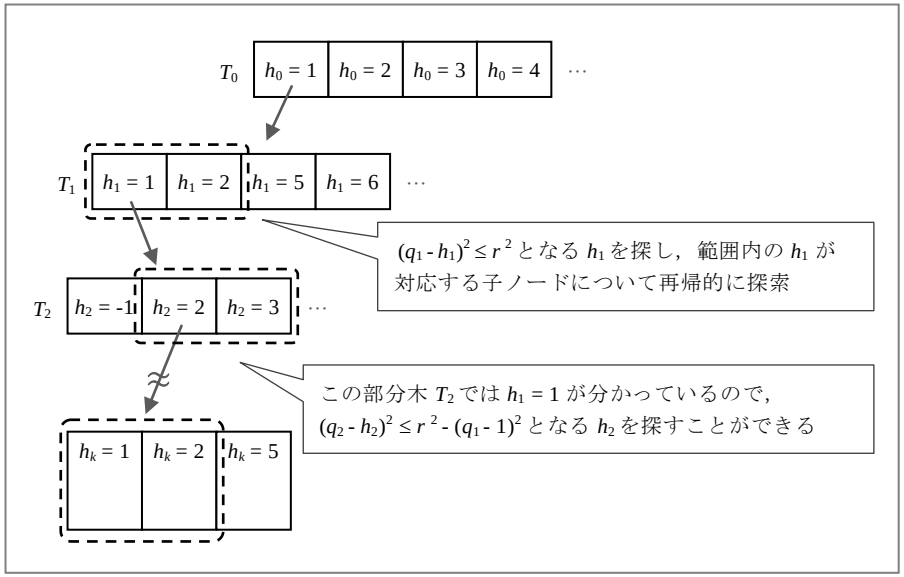

図 8　ハッシュテーブル探索手順概念図

左記判定式を変更すると，$(q_2 - h_2)^2 \le r^2 - (q_1 - h_1)^2$ となる．3 段目以降も同様にして，$(q_3 - h_3)^2 \le r^2 - (q_1 - h_1)^2 - (q_2 - h_2)^2$ となる．この時，$\alpha_1 = r^2$，$\alpha_i = \alpha_{i-1} - (q_{i-1} - h_{i-1})^2$ とおくと，i 段目の各ノード T_i での判定式は $(q_i - h_i)^2 \le \alpha_i$ と記述することができる．このことを利用し，幅優先探索は通常キュー等を用いて実装されるが，そのキューに入れる情報として，ノードへのポインタに加えて α_i の値も保存しておく．すると，根ノードまで辿らずとも自身の情報だけで次に辿るべき h_i の範囲が取得できる．さらに α_i は漸化式であるので，α_i から α_{i+1} を算出することも容易である．これより，効率的にハッシュ値候補を探索しながら同時にデータを探索することができる．

以上の方法により，前述の 2 つの問題点を解消できる．実際にデータが含まれていないハッシュ値は，探索途中で子ノードがなくなるので早い段階で足きりができる．例えば $h_1 = 1$, $h_2 = 3$ となるようなハッシュ値がハッシュテーブル内に存在しない場合，改善前の Resizable-LSH では $h_1 = 1$, $h_2 = 3$, $h_3 = 2$, …となるハッシュ値も全て列挙していた．しかし本改善手法では木を辿る際に $h_1 = 1$, $h_2 = 3$ 以降の部分木が存在しないので，探索の足きりができ，不要な列挙の削減ができる．また，ハッシュ値の検索に超立方体ではなく超球を直接使っているので，次元数 k が増えても木の段数増加によるコスト増加だけで済む．そして α_i を用いることで，そのコストの増加量も小さく抑えることができる．以上より，Resizable-LSH の速度改善が達成できる．

4. 評価

4.1 実験内容

性能評価をするために，単純な類似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．データセットが高次元な場合と低次元な場合とで既存手法と提案手法の性能比較をするために，データセットを 2 種類用意した．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データセット[9]と，Twitter Streaming API[12]によって取得した半日分の呟き情報 479,040 件から取得した文字 uni-gram(文字カウント)ヒストグラムである．データセットの詳細については 4.2 と 4.3 で述べる．

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

実験は，閾値すなわち検索半径 $r = 0.1$〜1.7 として，データセットからランダムに選んだデータをクエリとして検索を 100 回行い，所要時間と結果の精度を平均する．提案手法では，次元削減後次元数 $k = 5$，使用テーブル数 $L = 10$，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，2.3 で説明した超球分割を使用した LSH[2]のほかに，SR-Tree[10]も対象とした．SR-Tree はツリーを用いた非近似による類似検索手法であるが，閾値を指定した検索ができ，低次元データに対しては有力な手法の 1 つであるため，比較対象とした．既存の LSH 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．SR-Tree は片山らが公開しているライブラリ[13]を用いた．

精度の評価には F 値を用いた．F 値は適合率と再現率の調和平均である．また，特徴量の選択が精度へ与える影響を除外するため，クエリとの距離が閾値 r 以内である特徴量ベクトルを正解とした．

4.2 画像データセット

画像データセットは 66 次元の特徴量ベクトル 484,010 件からなる．[9]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．

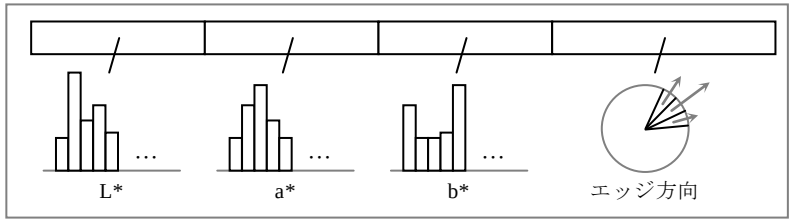

図 9　特徴量ベクトル内訳図

4.3 呟きデータセット

呟きデータセットは 9,365 次元のベクトル 479,040 件からなる．Twitter Streaming API[12]によって取得した多様な言語の呟きデータを用いて，各文字の出現回数をヒストグラムとした．1 日分のデータ内で 9,365 種類の文字が出現したため，次元数は 9,365 次元となっている．こちらも合計が 1 になるように正規化されている．

4.4 結果

結果は図 10〜図 13 のようになった．

図 10，図 11 は検索速度を表している．横軸が検索半径 r，縦軸が検索にかかった秒数（対数表示）である．なお，図中の Resizable-LSH が改善前の手法，提案手法が改善後の手法を意味している．改善前の Resizable-LSH では，$r = 1.7$ の場合メモリ不足で異常終了してしまうため，プロットしていない．

図 12，図 13 は検索精度を表している．横軸が r，縦軸が F 値である．Resizable-LSH の改善前後で精度は変化しないため，Resizable-LSH については表示していない．また，SR-Tree は近似手法ではなく厳密解が返されるので，F 値は常に 1 である．

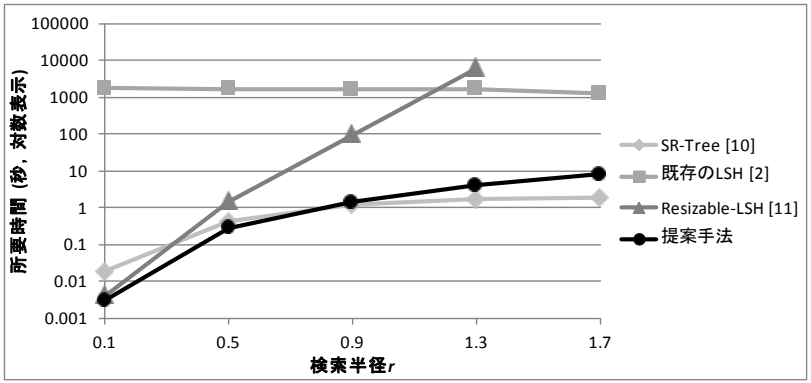

図 10　画像データセットにおける検索半径－検索所要時間比較グラフ

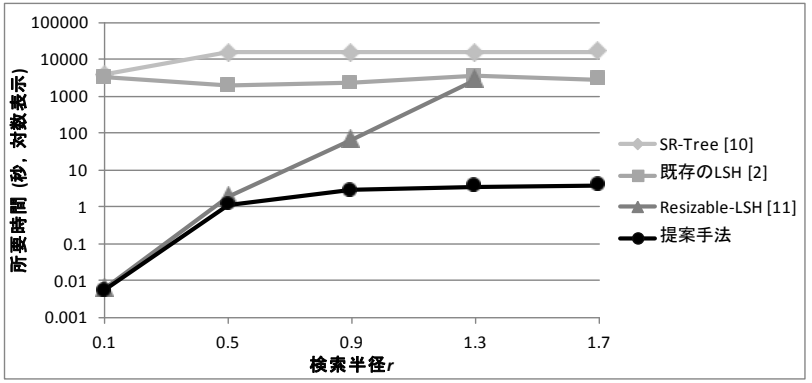

図 11　呟きデータセットにおける検索半径－検索所要時間比較グラフ

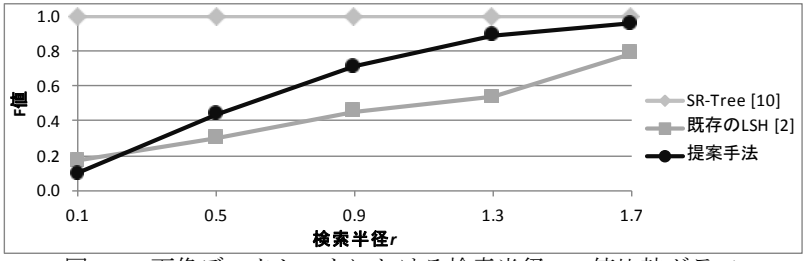

図 12　画像データセットにおける検索半径－F 値比較グラフ

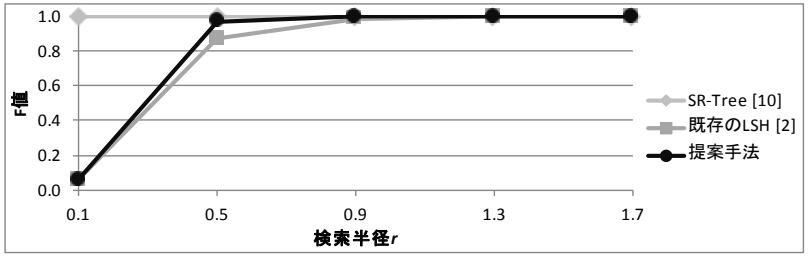

図 13　呟きデータセットにおける検索半径－F 値比較グラフ

4.5 考察

図 10 と図 11 を見ると，閾値が大きい場合でも提案手法は旧来の Resizable-LSH よりも高速に動作していることが分かる．特に $r = 1.7$ の場合は，ハッシュ値一覧がメモリとスワップ上に乗りきらないため改善前の Resizable-LSH では計測できなかったのに対し，本改善手法では 10 秒以下で実行できている．更に，高次元なデータセットを用いた図 11 の $r = 1.3$ の時を見ると，改善前の Resizable-LSH では 47 分，既存の LSH では 60 分，SR-Tree では 4 時間 25 分かかっているところを，提案手法では 3.5 秒で検索を終えており，800 倍以上の速度向上を達成している．また図 12 と図 13 を見ると，r が小さい時 F 値も小さいが，既存の LSH による手法と比較して，提案手法は F 値が低下していないことも読み取れる．

低次元なデータセットを用いた図 10 では，SR-Tree が提案手法よりも高速に動作している．しかし，高次元なデータセットを用いた図 11 では，SR-Tree は既存の LSH よりも遅くなっている．一方提案手法では，高次元なデータセットに対しても速度低下がほとんど見られず，現実的な時間で閾値を指定した類似検索を達成できている．

ただし，F 値に関しては厳密解を得られる SR-Tree の方がよい．提案手法や LSH，Resizable-LSH は，k の値を増加させると F 値を上昇させることはできるが，同時に L の値も大きくする必要がある[1][11]ので，速度とのトレードオフになってしまう．そのためパラメタ k および L を調整することが今後の更なる改善にとって重要である．

5. おわりに

本稿では，閾値以内のデータ一覧を取得する近似領域探索を高速に実現する手法 Resizable-LSH の，検索時間の更なる改善方法を提案した．本手法により，巨大な閾値に対しても高次元なデータに対しても，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．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，検索半径が小さい場合 F 値が低い値に留まってしまった．今後は，精度を向上させるため，パラメタ k および L を調整して本手法の適用及び改善を行う予定である．

謝辞　本研究の一部はＪＳＴ情報基盤戦略活用プログラム「多メディア Web 解析基盤の構築及び社会分析ソフトウェアの開発」及び科学研究費補助金（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) 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).
10) 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).
11) 山﨑邦弘, 中村智浩, 舟橋卓也, 山名早人 : “Resizable-LSH : 可変領域型の近似的類似検索,” 情報研報，Vol.2009-DBS-148 No.22，pp.1-8 (2009).
12) dev.twitter.com: Streaming API Documentation, http://dev.twitter.com/pages/streaming_api
13) 片山紀生；HnSRTree-2.0beta6，http://research.nii.ac.jp/~katayama/homepage/research/srtree/