類似画像検索を実現する距離空間インデックスの実装及び評価

岩崎 雅二郎†

画像から抽出した特徴量を比較して類似する特徴量を検索する画像検索では特徴量が膨大になると高速に検索するために検索インデックスが必須となる．色のヒストグラムのように特徴量の類似度の算出において特徴量次元間の相関を考慮しなければならない場合には多次元空間インデックスを利用することができない．一方，既存の距離空間インデックスでは動的にオブジェクトの登録ができない，検索速度が十分でない，といった問題点がある．そこで本稿では距離空間インデックスに動的な登録及び検索の高速化の改良を加え，実際に類似画像検索に適用し評価した結果について述べる．

Implementation and evaluation of metric space indices for similarity search

MASAJIRO IWASAKI†

Retrieval indices are indispensable for faster retrieval especially when a similar image search system has an enormous amount of images. Multidimensional space indices are not suitable for retrieval indices that need to correlate between each feature value. On the other hand, it is impossible to use dynamic object insertion or achieve high speed searches using conventional metric space indices. Therefore, two new metric space indices that adopt dynamic insertion and improved search algorithm are proposed. A similar image retrieval system utilizing these indices is also evaluated.

1. はじめに

近年，CPUの高速化及び一次／二次記憶の大容量化により，画像を手軽に扱えるようになっただけでなく，WWWやデジタルカメラの普及により画像データを容易に取込めるようになり，身の回りに画像データが氾濫しつつある．こうして大量の画像データからユーザが所望する画像を検索する技術が脚光を浴び始めた．従来の画像検索では予め人手により画像に属性情報を付与し，この属性情報を基に検索する方法が一般的であった．しかし，人手により属性を付与していては画像データの急増には対応ができない．そこで，画像データから自動的に特徴量を抽出し，その特徴量を基に指定された問合わせ画像に類似する画像を検索する類似画像検索の研究が行われている．

画像から抽出される特徴量として代表的なものには色，テクスチャ，形状などが上げられる．類似画像検索では，これらの特徴量における距離を定義し，各画像から抽出した特徴量間の距離を求め類似する画像を検索する．実際には，これらの特徴量を予め画像から抽出し保管しておく．検索時には指定された問合わせ画像から同様に特徴量を抽出し，既に抽出されている特徴量と逐次比較（距離計算）を行い類似する画像を検索する．この方法では特徴量の数が少量の場合には問題がないが，大量になると必然的に遅くなる．そこで，特徴量のインデックスを生成し検索の高速化が重要となる．

検索用のインデックスを生成する場合に問題となるのが特徴量間の距離定義である．特徴量として代表的な色のヒストグラム特徴量での距離定義として，ユークリッド距離を利用することができる．この場合には検索インデックスとして多次元空間インデックスを利用することが可能である．しかし，ヒストグラムのビン（ヒストグラムを構成する各色に対する出現度数）間の色の類似性を考慮しないので類似度を表現する距離としては精度が低い．一方，ユークリッド距離ではなく，ビン間の相関を考慮する距離を利用することで精度を向上させることが可能であるが，この場合にはユークリッド距離に基づいた多次元空間インデックスを検索インデックスとして利用することが困難である．

多次元インデックスでは多次元空間上での特徴量の座標値を基にインデックスを生成するのに対し，距離のみに基づいてインデックスを生成する距離空間イン

† リコー ソフトウェア研究所
Software Research Center, RICOH Co., Ltd.

デックスの研究がある．距離空間インデックスでは特徴量の座標は全く関知せず，特徴量間の距離のみによりインデックスを生成する．従って，相関を考慮した距離であっても問題なくインデックスを生成できる．

既存の距離空間インデックスには，検索速度は十分だがインデックスを一括して生成した後，削除，更新といった処理ができない静的なインデックスと，検索速度は十分ではないが削除，更新といった処理が可能な動的なインデックスがある．

我々は類似画像検索データベースを実現することを目標としており，高速な検索ができ，かつ，動的なインデックスが必須である．そこで，既存の静的なインデックスを動的に改良したインデックス及び動的なインデックスの検索速度の高速化の改良を行ったインデックスをそれぞれ実際に類似画像検索として実装し，比較評価を行った．その結果，静的なインデックスを改良したインデックスが登録検索性能において他のインデックスより高い性能を示した．本稿では両インデックスの改良点及び比較評価の結果について述べる．

2. 距離空間インデックス

画像を O，特徴量抽出関数を $F(O)$，特徴量のオブジェクト（空間インデックスにおいては画像特徴量だけでなく，あらゆるデータを扱えるので以降特徴量をオブジェクトと呼ぶこととする）を $X = \{X_1, X_2, X_3, \ldots, X_n\}$ とすると $X = F(O)$ であり，画像 a, b 間の距離（類似度）は $D(F(O_a), F(O_b))$ と表される．類似画像検索では指定した問合わせ画像に類似する画像を検索する．つまり，問合わせ画像と各画像との距離を算出し距離の小さい画像を検索結果とすることである．なお，検索の指定方法として以下の二つがある．

- 範囲指定検索 (range query)
  検索範囲を示す円の中心オブジェクト（問合わせオブジェクト）O_q 及び半径 R_q を指定し，$D(O_q, O) \leq R_q$ を満足するオブジェクト O の集合を求める．
- 件数指定検索 (k-nearest neighbor query)
  検索の中心オブジェクト O_q 及び検索件数 k を指定して O_q との距離 $D(O_q, O)$ が昇順に上位 k 件のオブジェクト O の集合を求める．

また，類似画像検索では類似する尺度を表す距離定義が重要となる．代表的な特徴量間の距離として，以下の式で表される距離がある．

$$D_e = \{\sum (X - Y)^k\}^{1/k} \quad (1)$$

この距離は $k = 2$ の時にはユークリッド距離であり，$k = 1$ の時には市街区距離となる．しかし，色のヒストグラム特徴量では，各特徴量間の相関を考慮する必要があり，以下の quadratic form 距離が提案[1]された．

$$D_q = (X - Y)^T A (X - Y) \quad (2)$$

R-tree[2] で代表される多次元空間インデックスの研究ではデータマイニングといった応用を背景に多種多様なインデックス[3)4)5)6)7)]が提案されている．しかし，多次元空間インデックスは一般にユークリッド空間（式1）に基づいているので，式2には単純に適用できない．平均色による多次元空間インデックスによって検索した後に誤検索を除去する方法[8]や同様に二段階の処理により色特徴量に限らず quadratic form 距離一般に利用できる方法[9]などが提案されている．しかし，quadratic form 距離以外の相関を考慮するような距離を新たに考案した場合にはこれらの手法を単純に適用できない．一方距離空間インデックスは以下の距離定義さえ満足すればインデックスを形成することができる．

(1) $D(X, X) = 0$
(2) $D(X, Y) > 0 \ (X \neq Y)$
(3) $D(X, Y) = D(Y, X)$ (対称性)
(4) $D(X, Y) \leq D(X, Z) + D(Z, Y)$ (三角不等式)

このように距離空間は相関があるような複雑な距離定義にも適応できるという利点により幾つかのインデックスが提案されている．距離空間インデックスは多次元空間インデックスと同様に空間を順次分割することで木構造を形成し，どのように空間を分割するかにより各インデックスを特徴づけている．

gh-tree[10](図1) は generalized hyperplane により空間を分割する．個々のノードは二つのオブジェクトを持ち，二つのオブジェクトから等距離にある線を generalized hyperplane と呼びこれにより空間を分割する．ただし二分木である gh-tree の場合には木構造が深くなる傾向があり，その結果，検索時の距離計算回数が多くなり検索速度の低下を引き起す．

GNAT[11](図2) では，中心オブジェクトを複数設定し，generalized hyperplane により空間を分割する．結果として空間はボロノイ分割となる．オブジェクトは最も近接する中心オブジェクトに属すように割り振られる．検索時に複数のノード空間のいずれに検索範囲が属すかを判断する上ですべての中心オブジェクトとの距離を求める必要があり検索速度の低下を招く．

距離空間インデックスの検索処理に占める時間は，二次記憶へのアクセス時間はもちろんのこと，距離計

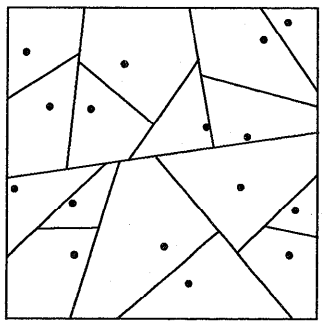


図1 GH-tree

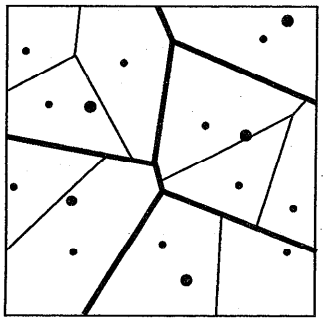


図2 GNAT

算時間が大きな割合を占める．特に quadratic form 距離のように相関を考慮した距離の場合には増大する．従って，距離計算の回数を減らすことが大きな課題となる．

そこで GNAT ではすべての二つの子ノードの組合せにおいて，子ノードの中心オブジェクトと，もう一方の子ノードに属すオブジェクトの最小最大距離を予め保持する．その距離から三角不等式に基づき検索範囲がいずれの子ノードに属さないかを距離計算なしに判断することができる．こうして距離計算回数の削減し検索速度の高速化を実現している．ただし，すべての子ノード間の距離を保持するのでノードのデータ量が増加するだけでなく，データ構造が複雑であるという問題点がある．

一方，vp-tree[12](図 3) では各ノードの空間は一つの中心オブジェクト（vantage point）と分割円（二次元空間ではないので実際には円ではないが説明上円と表わす）によって順次分割される．検索時にはルートノードから辿り，検索範囲が中心オブジェクトと半径によって分割されている領域のいずれに属すかを判断し辿る子ノードを決定する．これを繰り返して検索範囲に適合するリーフを探す．

vp-tree では一つの中心オブジェクトに対し複数の分割円を設定し多分木とすることで，中心オブジェクトからの距離を一回計算するだけで複数のノード領域のいずれに属すかを判断でき，距離計算回数を減らすことができる．しかし，特徴量の場合には空間を一つの中心オブジェクトのみで数多く分割しても各領域は極めて細いリング状になり，検索時にすべての分割領域が検索範囲と交差してしまい，分割の効果が薄れてしまう傾向がある．

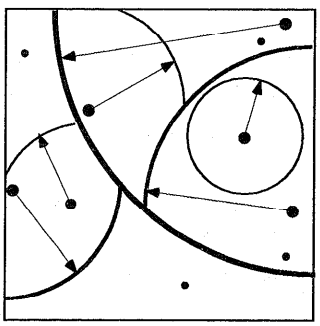


図3 vp-tree

mvp-tree[13] はデータ構成は複雑となるが中心オブジェクトを複数設定して複数の中心オブジェクトにより空間を分割することで，分割円が細いリング状になることを防ぎ，かつ，多分木を構築することで検索を高速化している．

以上述べたインデックスはすべて静的にインデックスを構築するアルゴリズムであり，インデックスを構築した後にオブジェクトを追加登録したり削除したりといった操作ができない問題がある．

一方 M-tree[14](図 4) は以上のインデックスとは異なり動的にオブジェクトの登録を考慮し，かつ，バランスのとれた木構造を保つ．M-tree の各ノードは中心オブジェクトと半径によって形成されノードは複数の子ノードを有する．図 4 のようにノードの円は子ノードの円を完全に包含し，ノードの円は必ずいずれかの子ノードの円に接する．M-tree では各ノードが中心オブジェクトと，ノードに属すすべての子ノードの円を包含する円の半径を保持する．さらに各ノードは中心オブジェクトと親ノードの中心オブジェクト間の距離をもつ．GNAT と同様にこの距離により三角不等式に基づいて検索時に距離計算回数を削減している．

また，オブジェクトが登録され木構造が成長する時には B 木と同様にリーフで生成されたノードが繰り上がってルートノードで木構造全体が繰り下がるように成長する．このことに起因してノード空間が大きくなり，検索速度が低下する傾向がある．

我々は類似画像データベースを実現することを目標としており，検索を高速に行え，かつ，登録，削除と

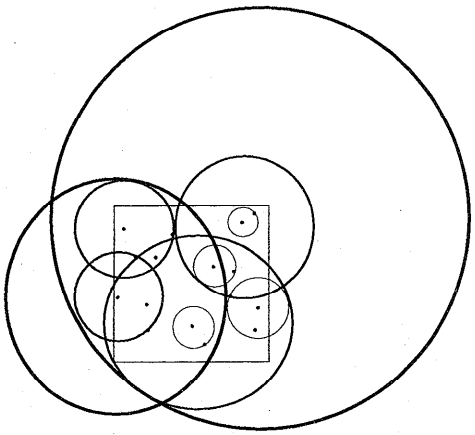


図 4 M-tree

いった処理が随時高速に処理できることが必須条件となる．しかし，前述のように静的なインデックスは動的にオブジェクトの登録，削除ができない，また，動的なインデックスでは検索速度が十分でないという問題点があり，どちらもこの条件を満足しない．しかも，どちらのインデックスを改良した方がこの条件に，より適合するインデックスとなるかが不明であった．そこで，静的なインデックスに関しては動的に登録できるように改良を行い，また，動的なインデックスに関しては検索速度の向上を目的とした改良を行った．その上で実際に類似画像検索としてそれぞれ実装し比較評価を行った．

なお，静的なインデックスに関しては，前述のインデックスのうち vp-tree 以外はデータ構造が複雑であり，動的なアルゴリズムに変更することが極めて困難である．さらに，実装できたとしてもデータ構造の複雑さから登録，削除（検索木の縮退）の処理時間の増大が問題になることが容易に予想された．そこで，本稿では，比較的簡便なアルゴリズムである vp-tree に改良を加えることとした．また，動的なインデックスに関しては M-tree が唯一のインデックスであるので，これに対して改良を行った．

3. データ構造

木構造のデータ構造は基本的に vp-tree と M-tree の両方で共有できる構造とした．図 5 に木構造のデータ構造を示す．木構造はルートノード，内部ノード，リーフノード，そしてオブジェクトからなる．ルートノードは木構造の起点となるノードであり，リーフノードは末端のノードである．リーフノードは個々のオブジェクトへのリンクをもつ．本稿では検索の高速化のためにノードデータはすべてメモリ上に展開する

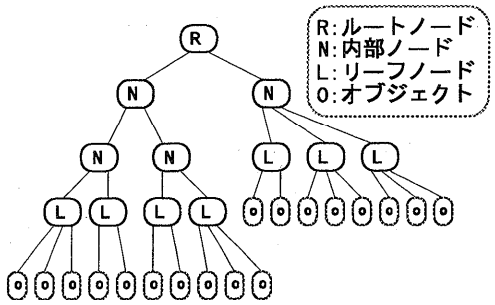


図 5 木構造

が，オブジェクトデータは容量が大きいので二次記憶に置くこととした．大容量のメモリを確保できるのであればメモリ上に展開することも可能である．ルートノード，内部ノード，リーフノードは以下のデータで構成される．

- ノード ID
- ノード種別（リーフノード，内部ノードの識別）
- 子ノードのノード ID リスト（リーフノードの場合にはオブジェクトの ID リスト）
- 分割円又はノード円の中心オブジェクト
- 分割円の半径 (vp-tree 用)
- ノード円の半径 (M-tree 用)
- 距離リスト（リーフノードの場合にリーフノードにリンクする全オブジェクトと中心オブジェクト間の距離）

4. Vp-tree の改良

Vp-tree に対し動的インデックス生成への変更を行っただけでなく，検索の高速化のためにリーフノードにおける検索アルゴリズムの改良を行った．なお，以降改良した vp-tree は dvp-tree(dynamic vantage point tree) と呼ぶこととする．

4.1 動的インデックス生成

vp-tree では登録するオブジェクトセットが予め与えられていることを想定しており，木構造を生成するときに順次このオブジェクトセットからオブジェクトをサンプリングする．従って，動的にオブジェクトを登録することができない．そこで，動的に登録するオブジェクトをサンプリングしたオブジェクトかのように扱うことで動的にデータを登録できるように改良した．

登録のアルゴリズムを図 6 に示す．登録オブジェクトはルートノードからオブジェクトを包含するノードを順次辿り，最終的にオブジェクトを包含するリーフノードに到達する．そのリーフノードに空きがあればオブジェクトを追加し，なければリーフノードを分割

```
Insert(N,Oi) // In:N,Oi
{
  if (N がリーフノードではない) {
    if (D(O,Oi) ≤ R) {
      Insert(Nci,Oi);
    } else {
      Insert(Nco,Oi);
    }
  } else {
    // N がリーフノードである
    if (N に空きがある) {
      N に Oi を追加する;
    } else {
      S = リーフノードの全オブジェクト + Oi;
      // リーフノードの分割
      // In:S,Out:Si,So,O,R
      Split(S,Si,So,O,R);
      リーフノード Nci を生成し Si を設定する;
      Nci の距離リストを設定する;
      リーフノード Nco を生成し So を設定する;
      Nco の距離リストを設定する;
      N の子ノードを Nci,Nco とし O,R を
      設定する;
    }
  }
}
N:カレントノード
Oi:登録オブジェクト
R:カレントノードの分割円の半径
O:カレントノードの中心オブジェクト
Nci,Nco:分割円内部, 外部の子ノード
S:オブジェクト集合
Si,So:分割円内部, 外部のオブジェクト集合
```

図 6 dvp-tree の登録処理
Fig. 6 Insertion of dvp-tree

する．分割するアルゴリズムが Split() である．リーフノードは内部ノードとなり分割されたリーフノードの親ノードとなる．

図中のリーフノードを分割する Split() のアルゴリズムは様々考えられるが，本稿では次のようなアルゴリズムを用いた．オブジェクト内の任意のオブジェクトを一つ選択し，そのオブジェクトから最遠にあるオブジェクトを分割円の中心オブジェクトとする．中心オブジェクトから各オブジェクトとの距離を求め中間点に位置するオブジェクトとの距離を分割円の半径とする．そして，分割円の内部，外部によりオブジェクトを二分する．

4.2 検索アルゴリズムの改良

vp-tree では検索時にルートノードから検索範囲に適合するノードを辿り，最終的に辿り着いたリーフノードにリンクされているオブジェクトに逐一アクセスし，距離を算出し検索範囲に適合するか否かを調べる．従って，リーフノードにおけるこの処理によって距離計算回数だけでなくオブジェクトへのアクセス回数を増大させている要因となっている．

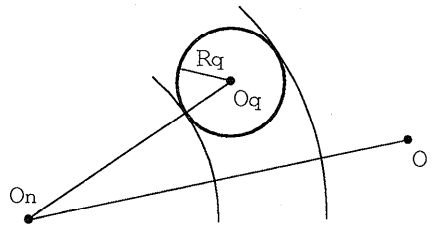


図 7 検索範囲とオブジェクトの関係
Fig. 7 Relation between a query range and an object

そこで，検索速度を向上させるためにリーフノードではリーフノードの中心オブジェクトと各オブジェクト間との距離を距離リストとして保持する．中心オブジェクトはリーフノードにリンクするオブジェクトのうちの任意の一オブジェクトとする．この中心オブジェクトと各オブジェクト間との距離により三角不等式を利用して距離計算回数の削減を行う．検索範囲の中心オブジェクトを O_q，半径を R_q，リーフノードの中心オブジェクトを O_n，リーフノードにリンクするオブジェクトを O とすると，これらの関係は図 7 となり，以下の定理が成り立つ．

定理 1 $|D(O_n,O) - D(O_n,O_q)| > R_q$ が成り立つならば，オブジェクトは検索範囲に存在しない

なぜなら，三角不等式

$$D(O_n,O_q) + D(O_q,O) \geq D(O_n,O)$$

より，

$$D(O_n,O) - D(O_n,O_q) > R_q$$

は

$$D(O_q,O) > R_q$$

となり，交差しないことがわかる．

$$-D(O_n,O) + D(O_n,O_q) > R_q$$

も同様にして，

$$D(O_q,O) > R_q$$

となる．従って，定理 1 は成立する．

定理 1 の $D(O_n,O)$ 及び R_q はいずれも既知であり，$D(O_n,O_q)$ は各リーフノードに対して一回計算すればよく，各オブジェクトとの距離を逐一計算せずに，オブジェクトが検索範囲に存在しないことが判断できる．従って距離計算回数やオブジェクトへのアクセス回数を大幅に削減できる．ただし，定理 1 の式が成立しないからといって必ず検索範囲に存在するわけではないので，成立しない場合にはオブジェクトとの距離

を算出し，検索範囲にあるか否かを調べなければならない．

実際の範囲指定検索のアルゴリズムを図 8 に示す．ノードがリーフの場合の一つ目の if 文での判定はオブジェクトを獲得せずに，かつ，個々のオブジェクトとの距離計算なしに判定ができる．二つ目の if 文の判定ではオブジェクトを獲得し，かつ，距離計算をしなければ判定できない．したがって，一つ目の判定により二つ目の時間を要する判定をせずに検索半径に包含されないことが判別でき，検索の高速化が可能である．また，件数指定検索 (k-nearest neighbor query) は範囲指定検索のアルゴリズムに基づいており，以下に述べるアルゴリズム [15] を採用した．検索初期値では検索半径は無限大とし，ルートから辿りオブジェクトを検索結果リストに加えていく．検索結果リストの検索数が指定された検索数を越えたら距離が最大の検索オブジェクトを検索結果リストから削除し，検索結果リストの検索数が指定された検索数を越えないようにする．さらに検索結果リストの最大の距離を検索半径とする．これを繰り返して行うことによって検索半径が絞られ最終的に指定件数分の検索結果を得られる．

```
Search(N,O_q,R_q,L) // In:N,O_q,R_q,Out:L
{
  if (N がリーフノードではない) {
    if (D(O,O_q) ≤ R + R_q) {
      // 分割円の内部と交差する
      Search(N_ci,O_q,R_q,L);
    }
    if (D(O,O_q) + R_q > R) {
      // 分割円の外部と交差する
      Search(N_co,O_q,R_q,L);
    }
  } else {
    // N がリーフノードである
    foreach O_c ( リーフノードの全オブジェクト ) {
      if ( |D(O,O_q) − D(O,O_c)| ≤ R_q ) {
        if ( D(O_q,O_c) ≤ R_q ) {
          L に O_c を加える;
        }
      }
    }
  }
}
O:カレントノードの中心オブジェクト
R:カレントノードの半径
O_q:検索の中心オブジェクト
R_q:検索の半径
O_c:オブジェクト
L:検索結果リスト
```

図 8 dvp-tree の検索処理
Fig. 8 Search of dvp-tree

5. M-tree の改良

動的にインデックス生成が可能である M-tree には検索速度が十分ではないという問題点がある．これは木構造の成長方式に起因するで木構造の成長のアルゴリズムを改良した．

5.1 成長アルゴリズムの改良

M-tree は B 木と同様にボトムアップの木構造の成長が大きな特徴となっている．新たなオブジェクトを挿入する時には検索時と同様にルートから木構造を辿り適切なリーフノードにオブジェクトを加える．加えた結果，オブジェクト数が最大分岐数を越えた場合にはリーフノードのオブジェクト集合を二つのリーフノードに分割する．一方は元のリーフと置換え，他方は親ノードに加える．その結果親ノードの子ノード数が最大分岐数を越えた場合には同様に親ノードの子ノード群を分割する．分割した結果ある最大分岐数を越えるとさらに親ノードへと遡っていく．ルートノードまで遡った場合にルートノードが最大分岐数を越えるとルートノードを分割し，木構造はルート部分で全体的に繰り下がる形態で成長する．従って，木構造は常にバランスを保っている．木構造がバランスを保っているので，如何なる検索もほぼ同等の検索速度となる利点があるが，木構造がルートから繰り下がる形態で成長する時に子ノード円を包含する円を設定するので，成長するごとに円が急激に増大する傾向がある．ノード円が巨大になると各ノード円のオーバーラップが増大し，探索枝の絞り込みが悪く検索速度が低下する傾向がある．

そこで，このようなボトムアップによる木構造の成長をやめ，R-tree のようなトップダウンの成長方式を取り入れた．トップダウンの成長方式とすることで当然木構造のバランスが悪くなるという欠点は発生するが，子ノード円のオーバラップを減少させることでバランスが悪くなることによる検索速度の低下を十分補えると判断した．

実際の M-tree のオブジェクトの挿入手順を述べる．オブジェクト挿入時にはルートから順に，オブジェクトを包含する子ノードがある場合にはその子ノードを辿り，オブジェクトを包含する子ノードがない場合にはオブジェクトを包含する円を生成するのに最小限の円の拡張で済む子ノードを辿る．こうして，リーフノードまで辿り着き，そのリーフノードに空きがあるならオブジェクトを追加し，空きがなければオブジェ

クトを加えた上でリーフノードを分割する．具体的なアルゴリズムを図9に示す．

図9中のSplit()はSをS_1とS_2のオブジェクト群に分割する．また，S_1，S_2の中心オブジェクトO_1，O_2，半径R_1，R_2を同時に求める．二分するアルゴリズムは様々考えられるが，本稿では以下の式を満たすオブジェクトO_i，O_jを中心オブジェクトとする二つの集合に分割する．

$$\min_{O_i,O_j \in S} \{ \max_{O_n \in S_1} (D(O_i,O_n)) + \max_{O_n \in S_2} (D(O_j,O_n)) \}$$

なお，

$$S_1 = \{O | O \in S \, and \, D(O_i,O) \leq D(O_j,O)\}$$
$$S_2 = \{O | O \in S \, and \, D(O_i,O) > D(O_j,O)\}$$

つまり任意の二つのオブジェクトを中心オブジェクトとして選択し他のすべてのオブジェクトを距離の近い方の中心オブジェクトに振り分ける．こうして各中心オブジェクトに属すオブジェクトを包含する二つの円の半径の合計を求める．これをすべての二つの組み合せについて行い，半径の合計が最も小さくなる二つのオブジェクトを新しいオブジェクトの中心オブジェクトとする．なお，検索アルゴリズムに関しては改良を加えていない．

6. 性能測定

以上に述べたdvp-tree，改良M-tree，そして比較のためにM-treeを実際に色特徴による類似画像検索に実装した上で性能評価を行った．使用マシンはSun Ultra 60(UltraSPARC-II 296MHz)，Solaris 2.6で，二次記憶にはUltra Wide SCSIハードディスク(7200rpm)を使用した．登録画像としてフォト画像やクリップアート画像などの25,458画像を用いた．M-tree及び改良M-treeの内部ノードの最大分岐数及びリーフノードの最大分岐数は共に10とした．dvp-treeの内部ノードの分岐数は2とし，リーフノードの最大分岐数は10とした．

画像特徴量として画像全体の色のヒストグラムを用い，特徴量の距離として以下の式で与えられるquadratic form距離を利用した．

$$D_h = (X-Y)^T A (X-Y) \quad (3)$$
$$= \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij}(X_i - Y_i)(X_j - Y_j) \quad (4)$$

なお，a_{ij}はヒストグラムのi番目のビンとj番目のビンの類似度であり，以下のように表される．

$$a_{ij} = 1 - d(c_i,c_j)/d_{max}$$

$d(c_i,c_j)$はi番目とj番目のビンの色空間上での距離(色差)，d_{max}はその最大値である．ヒストグラムのビン数（次元数）は54であり，上記式で示されるように各ビンの相関をすべて計算するので距離計算の処理時間は，ユークリッド距離に比べて膨大となる．なお，距離空間インデックスは多次元空間インデックスと異なり，距離のみに基づきインデックスを形成するので性能は次元数とは直接関係しない．しかし，距離定義により与えられた距離分布の傾向により性能が左右されるので，登録画像の距離分布を図10に示す．登録した画像からランダムに1,000組の画像の組を選

```
Insert(N,Oi) // In:N,Oi
{
  if (N がリーフノードではない) {
    if (D(Oi,Oc) ≤ Rc を満たす子ノードがある) {
      D(Oi,Oc) が最も小さい子ノード Nc を求める;
      Insert(Nc,Oi);
    } else {
      D(Oi,Oc) − Rc が最も小さい子ノード Nc を
      求める;
      Insert(Nc,Oi);
    }
  } else {
    // N がリーフノードである
    if (N に空きがある) {
      N に Oi を追加する;
    } else {
      // N に空きがない
      S=リーフノードの全オブジェクト+Oi;
      // リーフノードの分割
      // In:S,Out:S1,S2,O1,O2,R1,R2
      Split(S,S1,S2,O1,O2,R1,R2);
      リーフノード N1 を生成し S1,O1,R1 を設定
      する;
      リーフノード N2 を生成し S2,O2,R2 を設定
      する;
      N の子ノードを N1,N2 とする;
    }
  }
  子ノード又はリーフノードの全オブジェクトを包含する
  ように R を設定する;
}
N:カレントノード
Oi:登録オブジェクト
R:カレントノードの半径
Nc:子ノード
Rc:子ノードの半径
Oc:子ノードの中心オブジェクト
S,Sn:オブジェクト集合
On:Sn の中心オブジェクト
Rn:Sn の半径
```

図9 改良M-treeの登録処理
Fig. 9 Insertion of M-tree

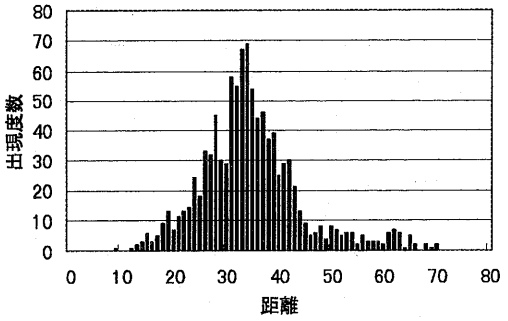


図 10 距離分布
Fig. 10 Distance distribution

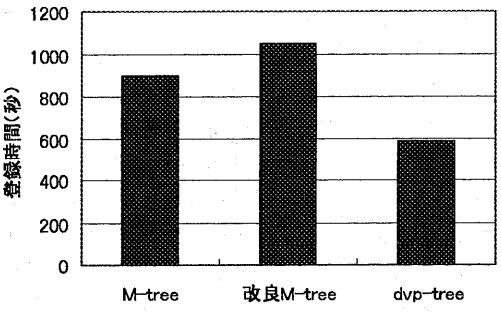


図 11 登録時間
Fig. 11 Processing time of building

択し，その画像間の距離を求め，距離を 1 単位に区分し各区分での出現度数をカウントした．

図 11 に全特徴量の登録時間を示す．登録時間には画像からの特徴量抽出時間は含まれていない．従って，実際の登録処理全体では特徴量の抽出処理によりかなりの時間を要する．図より改良 M-tree は登録速度の面で不利である一方 dvp-tree は他者より格段に速く M-tree に対して 34 %削減できている．

次に，検索件数を 10,20,50,100 とした場合の件数指定検索を行った．応用での類似画像検索では類似する度合を指定（範囲指定）せずに問合わせ画像に対して類似している上位数十件の画像を所望するニーズが多い．また，件数指定検索は範囲指定検索のアルゴリズムをベースにしており，件数指定検索のアルゴリズム自体はどちらのインデックスでも同様に実装されているので，件数指定検索を評価すれば範囲指定検索も評価することとなる．こういった理由により本評価では件数指定検索のみを行った．

検索時間に占める大きな要素の一つはオブジェクト間の距離計算である．quadratic form 距離のように計算コストが大きい場合には検索時間の大きな割合を距離計算が占める．従って，距離計算回数が少ないほど高速な検索が可能となる．検索時間に占めるもう一つの大きな要素は二次記憶へのアクセス時間である．近年メモリが安価になったとはいえ膨大なデータをすべてメモリ上に展開することは現実的ではない．インデックスはメモリに展開してもオブジェクトのデータ自体は二次記憶に配置するのが自然であろう．その結果，二次記憶へのアクセスは無視できない．本稿でもノードデータはメモリ上に，オブジェクトデータは二次記憶に配置している．

まず検索時間に大きな割合を占める一つ目の要素である距離計算回数を図 12 に示す．図より M-tree, 改良 M-tree，dvp-tree の順で距離計算回数が減少していることがわかる．また，検索数が増加するといずれのインデックスも距離計算回数が増加しているが，検索数に正比例するほどは増加していない．

次に二つ目の大きな要素である二次記憶へのアクセス時間としてノードのリード回数を図 13 に，オブジェクトのリード回数を図 14 に示す．いずれも距離計算回数と同様に M-tree，改良 M-tree，dvp-tree の順でリード回数が減少しており，検索件数が増えると増加傾向が見られる．検索件数が 10 件前後ならばノードのリード回数はオブジェクトのリード回数とほぼ同数で，検索件数が 100 件前後では半数程度に達する．従って，ノードデータを二次記憶ではなくメモリ上に置くことが検索速度の向上に大きく貢献していると判断できる．

次に実際の検索時間を図 15 に示す．なお，検索時間には問合わせ画像から特徴量を抽出する時間は含まれていない．図中の全件検索とはインデックスを利用せずにすべての特徴量と逐一距離計算を行い検索することである．距離計算回数およびオブジェクトリード回数から予想される通りの傾向を示しており，M-tree，改良 M-tree，dvp-tree の順で検索速度が速くなり，当然全件検索よりも格段に速い．改良 M-tree は M-tree に対して 15 ～ 16 %，dvp-tree では 49 ～ 62 %削減できている．

7. 結　　論

類似画像検索における画像間の類似度として利用される quadratic form 距離は多次元データの次元間の相関を考慮する．従って特徴量であるオブジェクトの検索インデックスとしてユークリッド空間に基づいた多次元空間インデックスを容易に利用できない．それに対して距離空間インデックスは距離定義を満しさえすれば利用できる．

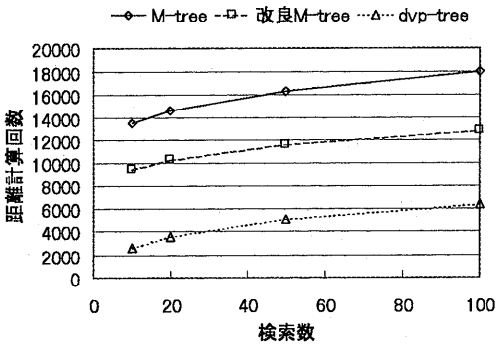


図 12 件数指定検索の距離計算回数
Fig. 12 Distance computations by k-nearest neighbor query

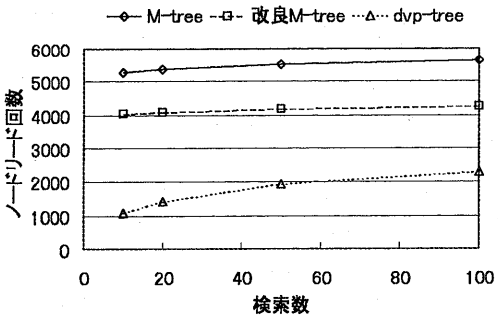


図 13 件数指定検索のノードリード回数
Fig. 13 Nodes visited by k-nearest neighbor query

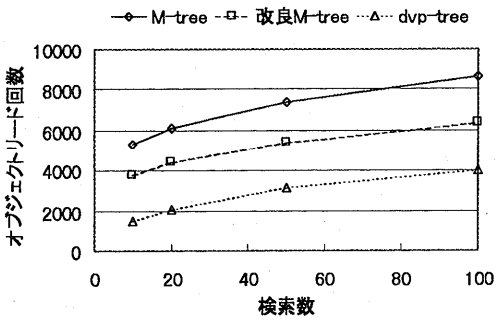


図 14 件数指定検索のオブジェクトリード回数
Fig. 14 Objects visited by k-nearest neighbor query

しかし，静的な距離空間インデックスの vp-tree は検索速度は十分であるが，動的にオブジェクトを登録できないという問題点がある．一方，動的な距離空間インデックスの M-tree は検索速度が十分ではないという問題点がある．そこで，静的なインデックスである vp-tree は動的にオブジェクトを登録できるように改良し，検索の高速化のために検索アルゴリズムにも改良を加えた．また，動的な距離空間インデックスで

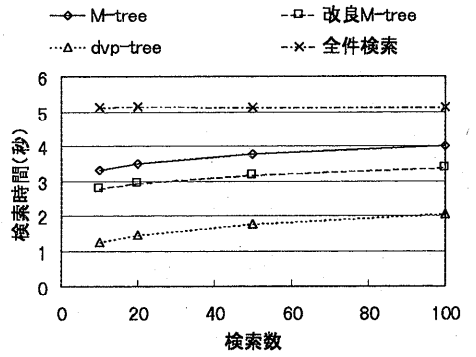


図 15 件数指定検索の検索時間
Fig. 15 Query time by k-nearest neighbor query

ある M-tree についてはボトムアップの木構造成長からトップダウンに変更した．さらに，vp-tree を改良した dvp-tree 及び M-tree を改良した 改良 M-tree を実際に類似画像検索として実装し評価を行った．

その結果，インデックスの生成及び検索のいずれの点においても dvp-tree が M-tree 及び改良 M-tree を上回り dvp-tree の有効性が確認された．一方，M-tree はその特徴であるボトムアップの木構造成長よりもトップダウンの木構造成長の方が検索性能の向上が得られることを確認した．しかし，dvp-tree に優ることはできず，M-tree の空間分割の方式に起因する問題が大きいと判断できる．

本稿では静的なインデックスである vp-tree を改良することで他のインデックスより優れた登録検索性能をもつインデックスを実現できた．しかし，実用的な大量画像のデータベースを想定した場合には，より高速な検索速度が望まれる．今後はさらなる検索の高速化を目指し改良を加える予定である．

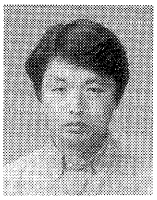
参 考 文 献

1) Ioka, M.: A Method of Defining the Similarity of Images on the Basis of Color Information, Technical Report RT-0030, IBM Tokyo Research Lab. (1989)
2) Guttman, A.: R-trees: A Dynamic Index Structure for Spatial Searching, Proc. ACM SIGMOD Int. Conf. on the Management of Data, pp. 47–57 (1984)
3) Bentley, J. L.: Multidimensional Binary Search Trees Used for Associative Searching, Communications of the ACM 18(9), pp. 509–517 (1975)
4) Samet, H.: The Quadtree and Related Hierarchical Data Structure, ACM Computing Surveys 16(20), pp. 187–260 (1985)

5) Beckmann, N., Kriegel, H., Schneider, R. and Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. ACM SIGMOD Int. Conf. on the Management of Data, pp. 322–331 (1990)
6) White, D. A. and Jain, R., Similarity Indexing with the SS-tree, In Proc. 12th IEEE Int. Conf. on Data Engineering, pp.516-523 (1996)
7) Kurniawati, R., Jin, J. S. and Shepherd, J. A.: The SS+-tree: An Improved Index Structure for Similarity Searches in a High-Dimensional Feature Space, In SPIE Storage and Retrieval Image and Video Databases IV, vol.3022, pp. 110–120 (1997)
8) Faloutsos, C., Barber, R., Flickner, M., Hafner j., Niblack, W., Petkovic, D. and Equitz, W.: Efficient and Effective Querying by Image Content, Journal of Intelligent Information Systems, Vol.3, pp. 231–262, July (1994)
9) Seidl, T. and Kriegel, H.-P.: Efficient User-Adaptable Similarity Search in Large Multimedia Databases, Proc. of the 23rd VLDB Conf. Greece, pp. 506–515 (1997)
10) Uhlmann, J. K.: Satisfying General Proximity/Similarity Queries with Metric Trees, Information Processing Letters 40, pp. 175–179 (1991)
11) Brin, S.: Near Neighbor Search in Large Metric Spaces, Proc. of the 21st VLDB Conf. Switzerland, pp. 574–584 (1995)
12) Yianilos, P. N.: Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces, ACM-SIAM Symp. on Discrete Algorithms, pp. 311–321, (1993)
13) Bozkaya, T. and Ozsoyoglu, M.: Distance-based Indexing for High-dimensional Metric Spaces, ACM SIGMOD, pp. 357–368, Tucson, AZ, (1997)
14) Ciaccia, P., Patella, M. and Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces, Proc. of the 23rd VLDB Conf. Greece, pp. 426–435 (1997)
15) Roussopoulos, N., Kelley, S. and Vincent, F.: Nearest Neighbor Queries, Proc. ACM SIGMOD Int. Conf. on the Management of Data, pp. 71–79 (1995)

(平成 10 年 9 月 20 日受付)
(平成 10 年 12 月 27 日採録)

(担当編集委員 片岡 良治)



岩崎雅二郎 (正会員)

1987 年早稲田大学理工学部工業経営学科卒業. 1989 年同大学院理工学研究科機械工学専攻修士課程修了. 同年日本電気 (株) 入社. 平成 2 年退社. 同年 (株) リコー入社. 現在同社ソフトウェア研究所に勤務. 情報検索の研究開発に従事.