社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2003－FI－ 72　(2)
2003－NL－157　(2)
2003／9／29

多次元データの高速近傍検索アルゴリズム

北 研二 [1]　獅々堀正幹 [2]　大恵俊一郎 [1]

[1] 徳島大学高度情報化基盤センター
[2] 徳島大学工学部

概要：高次元空間における最近傍検索 (nearest neighbor search) は、マルチメディア・コンテンツ検索、データ・マイニング、パターン認識等の分野における重要な研究課題の１つである。高次元空間では、ある点の最近点と最遠点との間に距離的な差が生じなくなるという現象が起こるため、効率的な多次元検索手法を設計することが極度に困難となる。本稿では、線形探索アルゴリズムにおける距離計算中の不要な演算を削減することにより、きわめて高速な最近傍検索アルゴリズムを提案する。さらに、不必要な演算を早期検出するために、要素の分散値を用いた次元ソート法、並びに主成分分析に基づくデータ変換法を提案する。実験によると、従来の SR-tree や VP-tree 等よりも 20 倍～50 倍高速であり、高次元の場合にも性能の劣化はほとんどない。

Fast Multidimensional Nearest Neighbor Search Algorithm

Kenji Kita [1]　Masami Shishibori [2]　Shun'ichiro Oe [1]

[1] Center for Advanced Information Technology, Tokushima University
[2] Faculty of Engineering, Tokushima University

Abstract: Nearest neighbor search in high dimensional spaces is an interesting and important problem which is relevant for a wide variety of applications, including multimedia information retrieval, data mining, and pattern recognition. For such applications, the curse of high dimensionality tends to be a major obstacle in the development of efficient search methods. This paper addresses the problem of designing an efficient algorithm for high dimensional nearest neighbor search. The proposed algorithm is based on a simple linear search algorithm and eliminates unnecessary arithmetic operations from distance computations between multidimensional vectors. Moreover, we propose two techniques, a dimensional sorting method and a PCA-based method, to accelerate multidimensional search. Experimental results indicate that our scheme scales well even for a very large number of dimensions.

1　はじめに

計算機の高性能化や記憶容量の大容量化および低価格化にともない、情報のマルチメディア化が急速に進行しており、このような背景のもと、マルチメディア・コンテンツに対する情報検索技術の必要性がますます大きくなってきている。マルチメディア・コンテンツ検索では、マルチメディア情報そのものから得られる特徴量に基づき類似検索を行なうという内容型検索 (content-based retrieval) が近年の主流であるが、多くの場合、複数の特徴量を多次元ベクトルで表現し、ベクトル間の距離によりコンテンツ間の類似性を判定している。たとえば、文書検索の場合には、索引語の重みベクトルで文書や検索質問を表現することができるし [20]、画像の類似検索の場合には、カラーヒストグ

ラム、テクスチャ特徴量、形状特徴量などから成る特徴量ベクトルにより画像コンテンツを表現する [13, 19]。

特徴量ベクトルに基づくコンテンツの類似検索は、検索質問として与えられたベクトルと距離的に近いコンテンツ・データベース中のベクトル (検索対象ベクトル) を見つけるという最近傍検索 (nearest neighbor search) の問題に帰着することができる。データベース中のベクトルと逐次的に比較する線形探索では、データベースの規模に比例した計算量が必要となるため、最近傍検索を効率的に行なうための多次元インデキシング技術の開発が重要な課題として、従来より活発に研究されてきた [1, 14]。

ユークリッド空間における多次元インデキシング手法には、R-tree [15], R+-tree [22], R*-tree [7], SS-tree [23], SS+-tree [18], CSS+-tree [16], X-tree [8], SR-tree [17] などが提案されており、また、より一般の距離空間を対象にしたインデキシング手法としては、VP-tree [24], MVP-tree [11], M-tree [12] などが提案されている。これらのインデキシング手法は、多次元空間を階層的に分割することにより、探索範囲を限定することを基本としている。しかし、高次元空間では、ある点の最近点と最遠点との間に距離的な差が生じなくなるという現象が起こるため [6, 9]、探索する領域を限定することができず、線形探索に近い計算量が必要になってしまうという問題点がある。

空間分割型とは異なる手法として、VA-File (vector approximation file) が提案されている [10]。VA-File では、多次元ベクトルデータを低次のビット表現で近似し、このビット表現に対し線形探索を行うことにより候補の絞り込みを行い、絞り込まれた結果に対してだけ、元の多次元ベクトルと検索質問ベクトルとの間の距離計算を行う。VA-File 自体は非常に単純であるが、最近傍検索の高速化という観点からは、空間分割型手法よりも優れていることが実証されている。

本稿では、単純で基本的な線形探索を改善することにより、きわめて高速な最近傍検索アルゴリズムを提案する。提案するアルゴリズムでは、ベクトル間の距離計算において不要な演算をスキップすることにより高速化を図るが、不必要な演算を早期検出するための多次元データの変換法についても同時に提案する。また、提案アルゴリズムの有効性を類似画像検索実験において評価する。

2 高速最近傍検索アルゴリズム

2.1 最近傍検索問題

多次元空間における最近傍検索 (nearest neighbor search) は、n 次元空間中に配置された N 個のデータベクトル (検索対象ベクトル) $\mathbf{x}_i$ $(i = 1, 2, \cdots, N)$ の中から、与えられたベクトル (検索質問ベクトル) $\mathbf{q}$ と距離的に近いものを見つけるという問題である。代表的な最近傍検索には、以下に示す 2 種類がある。

- k 最近傍検索 (件数指定検索)
  与えられた検索質問ベクトル $\mathbf{q}$ と距離的に近い k 個の検索対象ベクトルを見つける。

- ε 最近傍検索 (範囲指定検索)
  与えられた検索質問ベクトル $\mathbf{q}$ から距離 ε 以内にある検索対象ベクトルを見つける。すなわち、$d(\mathbf{q}, \mathbf{x}_i) < \varepsilon$ を満たす検索対象ベクトル $\mathbf{x}_i$ を求める。

なお、距離関数 $d(\cdot, \cdot)$ の定義にはさまざまなものがあるが、本稿では、ユークリッド距離を主に考える。

2.2 基本的なアイデア

本稿で提案する高速最近傍検索アルゴリズムでは、ベクトル間の距離計算において不要な演算をスキップすることにより高速化を図る。まず最初に、本提案手法における基本的な考え方を簡単な例で説明する。

いま、検索質問ベクトル $\mathbf{q}$ と検索対象ベクトル $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ が以下のように与えられているとする。

$$\mathbf{q} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

$$\mathbf{x}_3 = \begin{bmatrix} 8 \\ 1 \\ 2 \end{bmatrix}, \quad \mathbf{x}_4 = \begin{bmatrix} 3 \\ 5 \\ 3 \end{bmatrix}$$

ここで、検索質問ベクトル $\mathbf{q}$ に距離的に近い上位 2 件の検索対象ベクトルを探すという問題を考える。検索質問ベクトル $\mathbf{q}$ と最初の 2 つの検索対象ベクトル $\mathbf{x}_1$ および $\mathbf{x}_2$ との間の距離を計算すると以下のようになる[1] 。

$$\begin{aligned} d(\mathbf{q}, \mathbf{x}_1) &= 5 \\ d(\mathbf{q}, \mathbf{x}_2) &= 2 \end{aligned}$$

とりあえず $\mathbf{x}_1$ および $\mathbf{x}_2$ を検索結果の候補とする。この 2 件の候補に対する距離の最大値は 5 である。

3 番目の検索対象ベクトルに対する距離計算は、

$$d(\mathbf{q}, \mathbf{x}_3) = (1-8)^2 + (2-1)^2 + (3-2)^2$$

となるが、右辺の第 1 項 $(1-8)^2$ を計算した時点で、現在までの検索結果候補の距離の最大値 5 を越えるので、右辺第 2 項および第 3 項を計算するまでもなく、この検索対象ベクトルが検索結果となりえないことは明らかである。同様に、4 番目の検索対象ベクトルに対する距離計算は、

$$d(\mathbf{q}, \mathbf{x}_4) = (1-3)^2 + (2-5)^2 + (3-3)^2$$

となるが、右辺第 2 項までの和は、現在までの検索結果候補の距離の最大値 5 を越えるので、右辺第 3 項の計算は不要となる。

以上のように、検索質問ベクトルと検索対象ベクトルとの間の距離計算における演算を省くことにより、最近傍検索を効率的に行うことができるようになる。

2.3 高速最近傍検索アルゴリズム

上記の例が示すように、検索質問ベクトルと距離的に近い k 個の検索対象ベクトルを求める k 最近傍検索では、常に k 個の検索結果候補を保持しておき、検索結果候補中の距離の最大値を用いることにより、距離計算における不要な計算を途中で打ち切ることができる。これにより、計算の省力化を行うことができるため、線形探索を高速に実行することが可能となる。

もし新たな検索対象ベクトルとの距離計算の結果、新しく得られた距離値が現在の検索結果候補中の距離の最大値よりも小さな場合は、検索結果候補を更新する。検索結果候補の更新処理では、候補リストから最大の距離値を持つものを削除し、新たな候補を挿入するという操作が必要となるが、このような操作を効率的に行うことができるデータ構造に順位キュー (priority queue) がある [21]。

図 1 は、以上述べた考えを疑似コードで表現したものである。図の疑似コードでは、順位キューをヒープ (heap) により実現しており、ヒープには、検索質問ベクトルと検索対象ベクトルとの間の距離が既に計算されたもののうち、検索質問ベクトルと距離的に近い k 個の距離が保持されている。

以下、コードについて、簡単に説明する。8〜14 行目では、検索質問ベクトルと最初の k 件の検索対象ベクトルとの間の距離を逐次的に計算し、得られた距離値を配列 `heap` に順次代入する。18 行目で、配列 `heap` を降順に (距離の大きい順に) ソートし、ヒープ条件を満たすようにする。これにより、ヒープ先頭 (`heap[1]`) に、最初の k 件の検索対象ベクトルに対する距離のうち、最大値が置かれることになる。21〜35 行目では、検索質問ベクトルと $k+1$ 番目以降の検索対象ベクトルとの間の距離計算を行う。26 行目でベクトルの j 次元目までの累積距離 `dist` を求めるが、この結果がヒープの先頭値よりも大きい場合には、検索結果となりえないことが明らかなため、28 行目で距離計算を中止する。また、距離計算の結果がヒープ先頭値よりも小さくなる場合には、32〜33 行目でヒープを更新する。33 行目の `downheap` 関数は、ヒープ条件を満たすようにヒープを修復する処理を行う [21]。最後に、38 行目でヒープを昇順に (距離の小さい順に) にソートし、検索結果とする。

なお、k 個の検索検索質問ベクトル $\mathbf{q}$ との間の距離が ε よりも小さな検索対象ベクトル $\mathbf{x}_i$ を求める ε 最近傍検索問題では、$d(\mathbf{q}, \mathbf{x}_i) < \varepsilon$ となるような距離をヒープ上に保持するようにすればよい。

3 データ変換による不要演算の早期検出

2.3 で述べた最近傍検索手法では、ベクトルの各次元ごとの累積距離計算中に不要な演算であることを早期に検出することができれば、それだけ多くの演算を削減できることになり、より高速な検

[1] ここでは、簡単のため、ユークリッド距離の 2 乗 (要素の 2 乗和) を距離としている。

```
n  ←  次元数
N  ←  データ数
k  ←  検索件数
q  ←  検索質問ベクトル
x[i]  ←  i 番目の検索対象ベクトル

// 最初の k 件の検索対象ベクトルに対する距離計算
for(i = 0; i < k; i++)
{
    dist = 0.0;
    for(j = 0; j < n; j++)
        dist += (q[j] - x[i][j])*(q[j] - x[i][j]);
    heap[i+1] = dist;
}

// ヒープを降順にソートする。
// ヒープ先頭 (heap[1]) に現在までの検索結果候補の距離の最大値を置く。
descending_sort(heap);

// k + 1 番目以降の検索対象ベクトルに対する距離計算
for( ; i < N; i++)
{
    dist = 0.0;
    for(j = 0; j < n; j++)
    {
        dist += (q[j] - x[i][j])*(q[j] - x[i][j]);
        if(dist > heap[1])
            break;
    }
    if(dist < heap[1])
    {
        heap[1] = dist;
        downheap();  // ヒープの修復
    }
}

// ヒープを昇順にソートする。
ascending_sort(heap);

// 検索結果をヒープ先頭から順番に出力する。
```

図 1: 高速最近傍検索の疑似コード

索を行うことが可能となる。不要な演算を早期検出するための前処理として、以下の 2 つの方法を提案する。

3.1 分散値による次元ソート

検索対象ベクトルの各次元ごとに要素の分散値を求め、分散値の大きい次元が最初に来るように要素の入れ替えを行う。これにより、分散値の大きい次元から順に累積距離の計算が行われることになるので、累積距離が早い段階で大きくなることが期待でき、不要な演算を早期検出できる可能性が高くなる。

3.2 主成分分析によるデータ変換

分散値による次元ソートでは、ベクトルデータ中の要素を入れ替えるだけであったが、より効果的な方法として、線形変換によりデータそのものを変換する方法が考えられる。この際、変換後のベクトル間距離が保存されるためには、直交変換を用いる必要がある。

さまざまな直交変換があるが、このうち、主成分分析 (KL 変換) は、多次元ベクトルデータの変動を最もよく表すような基底を求めることができる。主成分分析では、多次元ベクトルデータの共分散行列を固有値分解することにより、固有ベクトルを新しい基底とするが、この際、大きな固有値に対する固有ベクトルほど、要素の分散が大きくなっている。固有値の大きい固有ベクトルから順番に第 1 主成分、第 2 主成分というように呼ぶが、まず第 1 主成分に対する座標値、次に第 2 主成分に対する座標値というようにして、あらかじめデータを変換しておくことにより、距離計算の際に不要な演算を早期に検出できる可能性を高くすることができる。

4 評価実験

提案した最近傍検索手法の有効性を調べるために、類似画像検索実験を行なった。以下で実験の概要および実験結果について述べる。

4.1 実験の概要

類似画像検索実験では、Corel 画像データベースから抽出した 51,138 件のカラー写真画像を用いた。また、このうち、1,000 件の画像データをランダムに抽出し、検索質問画像とした。これらの画像データから、表 1 に示すような、次元数の異なる 5 種類の特徴量ベクトルを作成した。

評価実験で用いた最近傍検索手法は、以下に示す 6 種類である。

- SR-tree [17] [2]
- VP-tree [24]
- 線形探索 (Linear)
- 提案手法 (Fast)
- 提案手法と次元ソート法を組み合わせた手法 (Fast-DSORT)
- 提案手法と主成分分析によるデータ変換を組み合わせた手法 (Fast-PCA)

なお、いずれの手法も 2 次記憶上に置かれたデータを検索するように実装されている。

4.2 実験結果

図 2 に実験結果を示す。図の棒グラフは、上記の各検索手法を用いて 1,000 件の検索質問画像を検索した際に、1 件当たりを検索するのに要した CPU 時間を示している。なお、検索実験は 5 回行い、平均時間を求めた。また、検索実験に使用した計算機は、CPU に 2.4GHz の Pentium-4、主メモリ 1,024k バイトのパソコンである。

図 2 の実験結果から分かるように、いずれの特徴量に対しても提案手法は従来の手法よりも高速であり、その差は高次元になるほど顕著となっている。提案手法に主成分分析によるデータ変換を組み合わせた手法 (Fast-PCA) は最も高速であり、高次元の場合にも性能の劣化はほとんどみられない。

また、今回の実験によると、SR-tree や VP-tree 等の空間分割型手法よりも、通常の線形探索のほうが高速な検索を行うことができた。高次元の場合には、複雑な空間分割型手法よりも、本提案手法のような線形探索の改良型手法が適していると考えることができる。

[2] http://research.nii.ac.jp/~katayama/homepage/research/srtree/ で公開されているプログラムを用いた。

表 1: 評価実験に用いた特徴量

特徴量	次元数	特徴量の概略
HSI-48	48	画像全体から 256 階調の色相 (hue)、彩度 (saturation)、輝度 (intensity) のヒストグラムを求め、各特徴量を 16 次元 (計 48 次元) に圧縮した特徴量
HSI-192	192	画像全体から 256 階調の色相 (hue)、彩度 (saturation)、輝度 (intensity) に関する HSI 特徴量を求め、各特徴量を 64 次元 (計 192 次元) に圧縮した特徴量
HSI-384	384	画像全体から 256 階調の色相 (hue)、彩度 (saturation)、輝度 (intensity) に関する HSI 特徴量を求め、各特徴量を 128 次元 (計 384 次元) に圧縮した特徴量
HSI-432	432	画像全体を 3×3 の部分画像に均等分割し、各部分画像に対して HSI 特徴量を求め、各部分画像の HSI 特徴量を 48 次元 (計 432 次元) に圧縮した特徴量
Lab-cube-576	576	画像全体を 3×3 の部分画像に均等分割し、各部分画像の色情報を Lab 表色系に変換後、各部分画像ごとに Lab 空間を $4 \times 4 \times 4 = 64$ 個の部分空間に分割し、それぞれの部分空間に属する画素の頻度値から得られる特徴量。画像全体としては、$64 \times 9 = 576$ 次元

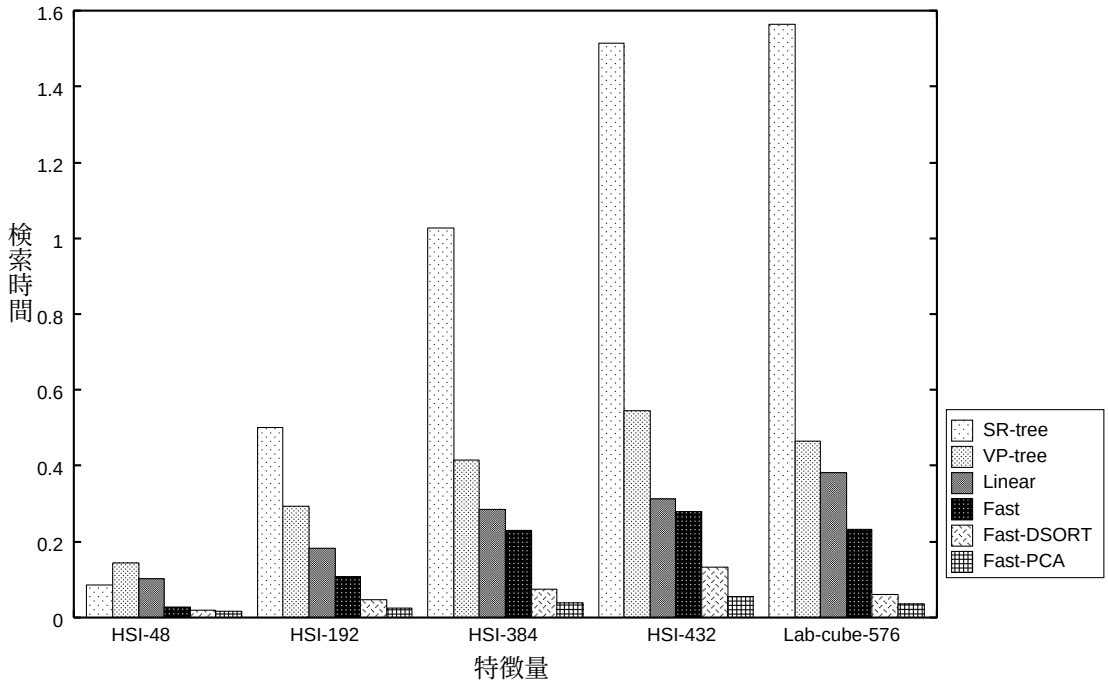

図 2: 1 検索質問当たりの CPU 時間の比較

5 おわりに

本稿では、線形探索アルゴリズムにおける距離計算中の不要な演算を削減することにより、高速な最近傍検索アルゴリズムを提案した。さらに、不要な演算を早期検出するために、要素の分散値を用いた次元ソート法、ならびに主成分分析に基づくデータ変換法を提案した。特に、提案した高速検索アルゴリズムに主成分分析によるデータ変換を組み合わせた手法はきわめて高速であり、高次元の場合にも性能の劣化はほとんどみられなかった。

今後は、我々が現在研究開発を進めているマルチメディア情報検索システム、クロスメディア情報検索システムに本提案手法の成果を取り入れ、高速・高精度なシステムを開発したい [2, 3, 4, 5]。

謝辞： 本研究に協力頂いた株式会社シンフォーム IT ソリューション事業本部ならびに株式会社ソフテックに感謝する。また、本研究の一部は、財団法人 放送文化基金、財団法人 電気通信普及財団の援助によった。

参考文献

[1] 片山 紀生，佐藤 真一：“類似検索のための索引技術”，情報処理，Vol. 42, No. 10, pp. 958–964, 2001.

[2] 小泉 大地，柘植 覚，獅々堀 正幹，北 研二：“色情報分布に基づく類似画像検索システムの開発”，情報処理学会四国支部研究シンポジューム論文集，pp. 35–38, 2002.

[3] 小泉 大地，柘植 覚，獅々堀 正幹，北 研二：“テキストと画像のクロスメディア情報検索に向けた画像キーワード登録システムの開発”，情報処理学会情報学基礎研究会，pp. 105–112, 2002.

[4] 小泉 大地，柘植 覚，獅々堀 正幹，北 研二：“画像知識データベースを用いた WWW からの画像検索システムの開発”，FIT2003. (発表予定)

[5] 光宗 朋宏，柘植 覚，黒岩 眞吾，獅々堀 正幹，北 研二：“話者情報を用いた映像検索システム”，FIT2003. (発表予定)

[6] Aggarwal, C.C., Hinneburg, A. and Keim, D. A.: “On the surprising behavior of distance metrics in high dimensional space”, *Proceedings of the 8th International Conference on Database Theory*, pp. 420–434, 2001.

[7] Beckmann, N., Kriegel, H. P., Schneider, R. and Seeger, B.: “The R*-tree: An efficient and robust access method for points and rectangles”, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 322–331, 1990.

[8] Berchtold, S., Keim, D. A. and Kriegel, H. P.: “The X-tree: An index structure for high-dimensional data”, *Proceedings of the 22th International Conference on Very Large Data Bases*, pp. 28–39, 1996.

[9] Beyer, K.S., Goldstein, J., Ramakrishnan, R. and Shaft, U.: “When is “nearest neighbor” meaningful?”, *Proceedings of the 7th International Conference on Database Theory*, pp. 217–235, 1999.

[10] Blott, S. and Weber, R.: “A simple vector-approximation file for similarity in high-dimensional vector spaces”, Technical report Nr. 19, ESPRIT project HERMES (No. 9141), 1997.

[11] Bozkaya, T. and Özsoyoglu, Z. M.: “Indexing large metric spaces for similarity search queries”, *ACM Transactions on Database Systems*, Vol. 24, No. 3, pp. 361–404, 1999.

[12] Ciaccia, P., Patella, M. and Zezula, P.: “M-tree: An efficient access method for similarity search in metric spaces”, *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB’97), pp. 426–435, 1997.

[13] Flickner, M. et al: “Query by image and video content: The QBIC system”, *IEEE Computer*, Vol. 28, No. 9, pp. 23–32, 1995.

[14] Gaede, V. and Günther, O.: “Multidimensional access methods”, *ACM Computing Surveys*, Vol. 30, No. 2, pp. 170–231, 1998.

[15] Guttman, A.: "R-trees: A dynamic index structure for spatial searching", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 47–57, 1984.

[16] Jin, J. S.: "Indexing and retrieving high dimensional visual features", In *Multimedia Information Retrieval and Management*, Feng, D., Siu, W. C. and Zhang, H. J. (Eds.), Springer, pp. 178–203, 2003.

[17] Katayama, N. and Satoh, S.: "The SR-tree: An index structure for high-dimensional nearest neighbor queries", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 369–380, 1997.

[18] 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", *Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases*, pp. 110–120, 1997.

[19] Pentland, A., Picard, R. and Sclaroff, S.: "Photobook: Content-based manipulation of image databases", *International Journal of Computer Vision*, Vol. 18, No. 3, pp. 233–254, 1996.

[20] Salton, G., Wong, A. and Yang, C. S.: "A vector space model for automatic indexing", *Communications of the ACM*, Vol. 18, No. 11, pp. 613–620, 1975.

[21] Sedgewick, R.: *Algorithms*, 2nd edition, Addison-Wesley, 1988.
野下浩平 他 (訳)：「アルゴリズム」，第1巻＝基礎・整列，近代科学社, 1990.

[22] Sellis, T., Roussopoulos, N. and Faloutsos, C.: "The R+-tree: A dynamic index for multi-dimensional objects", *Proceedings of the 12th International Conference on Very Large Data Bases*, pp. 507–518, 1987.

[23] White, D. A. and Jain, R.: "Similarity indexing with the SS-tree", *Proceedings of the 12th IEEE International Conference on Data Engineering*, pp. 516–523, 1996.

[24] Yianilos, P. N.: "Data structures and algorithms for nearest neighbor search in general metric spaces", *Proceedings of the Fourth ACM-SIAM Symposium on Discrete Algorithms*, pp. 311–321, 1993.