データベースシステム 124－19
情 報 学 基 礎 62－19
（2001．5．22）

距離索引 MetricMatrix と距離索引の効率について

石川 雅弘 † 陳 漢雄 ‡ 古瀬 一隆 ‡ 大保 信夫 ‡
農業生物資源研究所 † 筑波大学 電子・情報工学系 ‡

距離の定義されたデータ空間における効率的な近傍検索を支援するためのデータ構造、MetricMatrix を提案する。MetricMatrix は静的に構築されるメモリ格納型の距離索引であり、検索時の距離計算回数の削減を目的としている。その構造は非常に単純であり、応用に適している。本稿では MetricMatrix の構造と構築法、近傍検索手順を示し、実験によりその効率を測定する。最後に距離索引の有効性についての考察を述べる。

A metric index, MetricMatrix, and effectiveness of metric indices

Masahiro Ishikawa † Hanxiong Chen ‡ Kazutaka Furuse ‡ Nobuo Ohbo ‡
National Institute of Agrobiological Sciences † Institute of Information Sciences and Electronics, University of Tsukuba ‡

We propose a metric index, named MetricMatrix, which supports efficient near neighbour searches in an arbitrary metric space. MetricMatrix is a memory-resident, static index. Its distinct feature is the simple structure, which is profitable for applications. In this paper, we present its structure and procedures for construction and near neighbour searches. Experimental results are also presented. Finally, we discuss the effectiveness of metric indices.

1 はじめに

主に事務処理などに利用される伝統的なデータベース環境においては、数値や比較的短かい文字列が典型的データ種であった。そのような環境における典型的なデータベース問合せは、具体的な数値や文字列などを指定した完全一致検索である。しかし近年、画像、音声、全文文献あるいはゲノムデータなど多様なデータが電子的に記録・蓄積されるようになってきた。このようなデータ種に対しては従来のような完全一致検索は適さない。重要となるのは与えられたデータに似たデータを検索する類似検索である。

画像や文書データにおいては、対象データを特徴ベクトル空間に写像し、その空間上で類似度を定義する事が一般的である。そのため効率的な類似検索実現のためには R-tree などの多次元ベクトルのための索引を利用する事も考えられる。しかし文字列間距離のように類似度が元のデータ空間上で直接定義されている場合には適用できない。多次元ベクトル表現されたとしても各次元の相関を考慮する必要がある場合には二次形式距離などが用いられる事が多いが、このような非ユークリッド距離を用いた場合にも多次元索引の利用は困難である。

このような背景から、距離公理の成立のみを前提とし、データ間の相対距離のみに基づいた距離索引の研究が行なわれてきた [3] [2] [4] [6]。しかし既存の距離索引はいずれも木構造を取り、構造が複雑で応用に適さない。例えば単発の検索ではなく、検索半径を漸増させなが

ら同一データを中心とした検索を多数回発行するような状況では必ずしも効率的ではない。

本稿では単純な構造を持ち応用にも適した静的距離索引 MetricMatrix を提案し、その構築法および近傍検索法について述べ、実験結果から距離索引の効率についての考察を述べる。

2 準備

データ集合 O 上に以下の条件 (距離公理) を満たす全関数 $\phi : O \times O \to \mathcal{R}$ が定義されているとき、ϕ を O 上の距離関数、$\langle O, \phi \rangle$ を距離空間 (metric space) と呼ぶ:

$$\forall x, y \in O; \quad d(x,y) = d(y,x) \quad (1)$$

$$\forall x \in O; \quad d(x,x) = 0 \quad (2)$$

$$\forall x, y \in O; \quad x \neq y \to 0 < d(x,y) < \infty \quad (3)$$

$$\forall x, y, z \in O; \quad d(x,z) \leq d(x,y) + d(y,z) \quad (4)$$

式 (4) は三角不等式である。

距離索引 (*metric index*) とは、距離公理の成立のみを前提として距離空間中のデータに対する距離に基づく検索を支援する索引である。

高次元ベクトルや文字列間距離、二次形式距離などが利用される最近のアプリケーションにおいては距離計算コストも大きくなる傾向にある。そのため距離索引の主要な目的は検索時に必要な距離計算回数を削減する事にある。また、距離計算回数がデータ数 N に対し充分小さければ距離計算回数のオーダが $O(N)$ であっても距離索引のメリットは大きい。

2.1 距離空間における類似検索

距離索引により支援される類似検索問合せの代表的なものに r-近傍検索と k-最近傍検索がある。それぞれ次のように定義される。

定義 2.1 (r-近傍検索)
問合せオブジェクト q と検索半径 $r(> 0)$ が与えられた時、q の r-近傍に含まれるデータの集合

$$RN(q,r) = \{o \mid o \in \mathcal{O} \wedge \phi(q,o) \leq r\}$$

を求める検索を r-近傍検索 (r-neighbour search) と呼ぶ。□

定義 2.2 (k-最近傍検索)
問合せオブジェクト q と正整数 k が与えられた時、q からの距離が最も小さい k 個のデータの集合

$$\begin{aligned} KNN(q,k) \quad = \quad & \{o \mid o \in \mathcal{O} \\ & \wedge (\exists r; o \in RN(q,r) \wedge k \leq |RN(q,r)| \\ & \wedge (\forall r' < r; |RN(q,r')| < k))\} \end{aligned}$$

を求める検索を k-最近傍検索 (k-nearest neighbour search) と呼ぶ。□

2.2 空間分割と木構造

一般的に距離索引は対象データ集合を再帰的に分割管理し、それに対応した木構造を取る。空間の分割法は

1. 超球による分割
2. 超平面による分割
3. 超双曲面による分割

に分けられる。

超球による分割では、距離空間は一つの参照オブジェクトを中心とした超球によりその内部と外部に分割される。この方法を利用した距離索引には MVP-tree [2] や M-tree [4] がある。超球の半径はデータ分布を考慮して任意に選択できるため、バランスした木が得られる。

超平面による分割では、空間は二つの参照オブジェクトから等距離にある平面により分割される。GNAT [3] はこの方法を採っている。分割平面はデータ分布に依存せず決定されるため、必ずしもバランスした木とはならない。

超双曲面による分割では、二つの参照オブジェクトからの距離の差の等しい超曲面 (超双曲面) により分割される。この方法は MI-tree [6] により導入された。距離の差は任意の値を選択できるため、バランスした木が得られる。

2.3 R-近傍検索と解候補選択

全ての距離索引において、少なくとも概念レベルでは、r-近傍検索は次の二つのステップで実行される:

1. **候補選択**
   すべての解を含むオブジェクトの集合である**候補集合**を構成する。この要素を候補オブジェクト、その内実際には解でないものをフォールスドロップと呼ぶ。

2. **リファインメント**
   候補集合からフォールスドロップを取り除く。

リファインメントステップでは全ての候補オブジェクトと問合せオブジェクト間の距離が計算される。そのため候補集合のサイズをいかに小さく抑えるかが距離索引にとって重要である。

2.3.1 三角不等式に基づくフィルタリング

木構造と検索手順に加えて全ての既存距離索引で共通するのが、三角不等式 (4) から導かれる候補フィルタリングの原理の利用であり、距離索引の特徴ともなっている。図 1 はこの原理を二次元ユークリッド空間において示した例である。

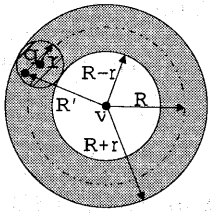

図 1: 三角不等式に基づくフィルタリング

R を参照オブジェクト v から問合せオブジェクト q への距離、R' を v からある解オブジェクト x への距離とする。この時次の不等式が成立する [2]:

$$R - r \leq R' \leq R + r.$$

図中灰色で示した円環はこの不等式が成立する領域である。この領域を r-近傍問合せの v に対する**候補領域**と呼ぶ。候補領域外のオブジェクトは本不等式を満たさないため候補選択のステップで取り除かれる。

参照オブジェクトと他の全てのオブジェクトへの距離が予め計算されていれば、候補領域内の全てのオブジェクトを選択するのに必要な距離計算は問合せオブジェクトと参照オブジェクト間距離のみである。

3 MetricMatrix

MetricMatrix はメモリ格納型で静的な距離索引であるが、その構造は非常に単純であり応用に適している。

2.3.1 で述べたフィルタリング原理は MetricMatrix でも利用しているが、より効率的にこの原理を利用するため、MetricMatrix では参照オブジェクトからの距離に基づいて全オブジェクトの並び換えを行なう。この点については以下で説明する。

3.1 構造

図 2 に N 個のオブジェクトのための MetricMatrix の構造を示す。構築に先立って各オブジェクトには 0 から $N-1$ の整数値が ID として付与されるものとする。

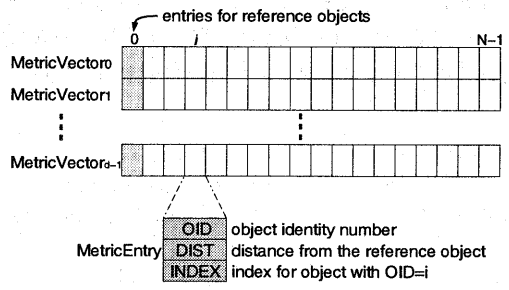

図 2: MetricMatrix の構造

図に示した通り、MetricMatrix は複数の MetricVector と呼ぶ MetricEntry の配列から構成される。一つの MetricMatrix 中の MetricVector の数をその MetricMatrix の**次数**

(*degree*) と呼ぶ。

一つの MetricEntry は一つのオブジェクトに対応しており、図 2 に示すように複数のスロットから成る。各 MetricVector は一つの参照オブジェクトに対応付けられており、MetricEntry の DIST スロットには参照オブジェクトから当該オブジェクトへの距離が格納される。OID スロットには構築に先立って与えらえた各オブジェクトの ID が格納される。他のスロットが当該オブジェクトに関する情報を格納するのに対し、INDEX スロットは若干異なった役割を持つ。すなわち、i 番目のエントリの INDEX 値 k は、OID 値が i であるオブジェクトのエントリ位置が k である事を示す。例えば 6 番目のエントリの INDEX スロットの値が 10 である時、OID 値が 6 のオブジェクトのためのエントリは 10 番目のエントリである。

MetricVector 中の MetricEntry は DIST 値によって昇順に並び換えられる。したがって参照オブジェクトに対応するエントリは常に左端に位置する。

DIST 値で並び換える事により、r-近傍問合せの候補領域中のオブジェクトは MetricVector 上で連続した範囲を構成する。この範囲を問合せの MetricVector における**候補範囲**と呼ぶ。候補領域と候補範囲の関係を図 3 に示す。候補範囲の DIST 値の上限と下限は容易に求まるため、MetricVector を参照する事で容易に r-近傍問合せのための候補集合を選択できる。

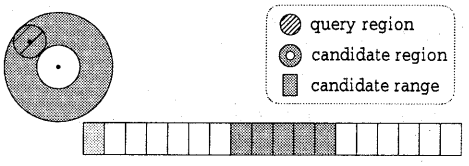

図 3: 候補領域と候補範囲

さらに、複数の MetricVector を利用する事でフォールスドロップを更に削減する事ができる。この様子を図 4 に示す。

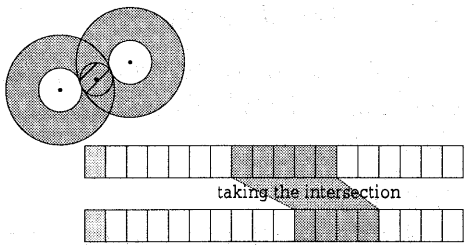

図 4: 複数の MetricVector による絞り込み

3.2 構築法

MetricMatrix の構築は次の手順で行なわれる:

1. 参照オブジェクトの選択。
2. 参照オブジェクトからの距離計算と並び換え。(MetricVector の構築)

```
 O    -  対象データセット
 deg  -  MetricMatrix の次数
 φ    -  距離関数
 M    -  MetricMatrix
---------------------------------------------
def construct(O, deg, φ):
    M.DEG ← deg
    ## 参照オブジェクトの選択
    dummy ← random_select(O)
    for i in range(0, O.SIZE−1):
        min_dists[i] ← φ(dummy, O[i])
    for i in range(0, M.DEG−1):
        M[i].REF ← 最大の min_dists[k] 値
                   を持つ O[k]
        for k in range(0, O.SIZE−1):
            M[i][k].DIST ← φ(O[k], M[i].REF)
            if i = 0
              or M[i][k].DIST < min_dists[k]:
                min_dists[k] ← M[i][k].DIST
    ## 各 MetricVector の構築
    for i in range(0, M.DEG−1):
        for k in range(0, O.SIZE−1):
            M[i][k].OID ← k
        M[i].sortByDIST()
        for k in range(0, O.SIZE−1):
            oid ← M[k].OID
            M[i][oid].INDEX ← k
    return M
```

図 5: 構築手続き

参照オブジェクトのデータ空間における分布は検索効率に影響を与えると考えらえるが、ここでは文献 [1] において提案されているヒューリスティクスを採用する。すなわち、互いに最も遠方に位置する k 個のオブジェクトを選択する。詳細は図 5 に示した構築手続きを参照されたい。図中 $\mathcal{M}[i]$ は第 i MetricVector を、$\mathcal{M}[i][k]$ は第 i MetricVector 中の第 k MetricEntry を表わしている。

3.3 R-近傍検索

前述したように、ある r-近傍問合せの候補集合は MetricVector を参照する事で容易に選択でき、複数の MetricVector を用いる事でその数を更に削減する事ができる。MetricMatrix における r-近傍検索手続きはこの方針をそのまま具体化する事で得られる。

1. まず各 MetricVector において参照オブジェクトから問合せオブジェクトへの距離を計算し、候補範囲の左端と右端を求める。

2. 次いで候補範囲をサイズの小さい順に並びかえる。
   共通集合を得るのが目的であるので、ある一つの MetricVector における候補が他の全ての MetricVector でも候補範囲に含まれている事を調べれば十分である。そこでサイズの最も小さい候補範囲内の各候補についてチェックを行なうためである。

3. 各オブジェクトについて、他の MetricVector 上でも候補範囲に入っているか否かをチェックする。チェックの順序は候補範囲のサイズの小さい順とする。これはなるべく早期にフォールスドロップを排除するためである。

この手続きの距離計算を除いた計算量は $O(dN)$ である。ここで d は MetricMatrix の次数、N はデータ総数である。N は MetricVector における候補オブジェクト数の最小数の最も悲観的な上限であるが、仮にそれが k で与えられるならば計算量は $O(dk)$ である。また、通常 d は比較的小さな数であり定数とみなす事もできるため、その場合計算量は $O(N)$ あるいは $O(k)$ である。この時距離計算回数は k である。

3.4 K-最近傍検索

K-最近傍検索のための手続きは r-近傍検索に比べ幾分複雑であり、五つ組のプライオリティキューを用いる。五つ組は MetricVector 上の MetricEntry を示し、以下の要素から構成される:

- DIM — MetricVector 番号。DIM 値が k の時、この五つ組は k 番目の MetricVector 中のエントリを表している事を意味する。

- INDEX — 当該エントリの MetricVector 上での位置。

- OID — 当該エントリの OID 値。

- DIFF — 当該エントリと問合せオブジェクトの DIST 値の差の絶対値。

- DIR — 当該エントリが問合せエントリの左右どちらに位置するかを表し、`left` もしくは `right` の値を取る。DIR 値が `left` の時、この五つ組は問合せオブジェクトよりも左側に位置する事を意味する。

例えば五つ組 $\langle dim, idx, oid, diff, \texttt{left} \rangle$ はエントリ $\mathcal{M}[dim][idx]$ を表し、その位置は問合せエントリよりも左に位置し、参照オブジェクトからの距離の差は $diff$ である。また対応するオブジェクトの OID は oid である。
キューは DIFF 値により昇順に順序付けられる。

K-最近傍検索手続きを図 6 に示す。

for ループ **a:** では五つ組のキューを初期化する。**while** ループ **b:** では五つ組が一つずつチェックされる。キューは DIFF 値で順序付けられているため、チェックされるのは各時点で DIFF 値が最小のものである。**c:** で更新

される $border[dim][dir]$ 変数は各ベクトル上で既にチェックされた範囲を記録する。

```
M       - MetricMatrix
q       - 問合せオブジェクト
nnk     - 検索オブジェクト数
φ       - 距離関数
entQue  - 五つ組
          ⟨DIM, INDEX, OID, DIFF, DIR⟩
          を要素とし DIFF 値により
          順序付けられたキュー
A       - 解集合
```

```
def knnsearch(M, q, nnk, φ):
    ## キューの初期化
a:  for i in range(0, M.DEG−1):
        qdist[i] ← φ(q, M[i].REF)
        qpos[i] ← M[i].binSearchDIST(qdist[i])
        ⟨l, r⟩ ← ⟨qpos[i], qpos[i] + 1⟩
        if 0 ≤ l :
            diff ← qdist[i] − M[i][l].DIST
            e ← ⟨i, l, M[i][l].OID, diff, left⟩
            entQue.push(e)
        if r < N :
            diff ← M[i][r].DIST −qdist[i]
            e ← ⟨i, r, M[i][r].OID, diff, right⟩
            entQue.push(e)
b:  while not entQue.empty():
        ⟨d, pos, oid, diff, dir⟩ ← entQue.pop()
c:      border[dim][dir] ← pos
        isCandidate ← true
d:      for i in range(0, M.DEG−1):
            if not border[i][left]
                    ≤ M[i][oid].INDEX
                    ≤ border[i][right]:
                isCandidate ← false
                break for
e:      if isCandidate:
            dist ← φ(q, O[oid])
            if A.SIZE < nnk
              or dist = A.farthestDistance():
                A.push(⟨O[oid], dist⟩)
            else if dist < A.farthestDistance():
                A.removeFarthestItem()
                A.push(⟨O[oid], dist⟩)
        pos′ ← (dir is left)
                ? (pos − 1) : (pos + 1)
f:      if 0 ≤ pos′ < N:
            diff′ ← | qdist[d] − M[d][pos′].DIST |
            if A.SIZE < nnk
              or diff′ ≤ A.farthest_distance():
                e ← ⟨d, pos′, oid, diff′, dir⟩
                entQue.push(e)
    return A
```

図 6: K-最近傍検索手続き

for ループ **d:** では、五つ組に対応するオブジェクトが全ての MetricVector 上で半径を *diff* とする候補範囲内にあるか否か判定する。*diff* は各時点で最小の値が選択されるため、もしある MetricVector においてそのオブジェクトが半径 *diff* の範囲内にあるならば、すでにそのオブジェクトの五つ組はチェックされているはずである。よって $border[][]$ を参照することで判定できる。

if 文 **e:** は解集合を更新する。**if** 文 **f:** では、チェックした五つ組を DIR 値に従って前方あるいは後方に進める。もしもこの時点で既に指定されただけの解候補が得られている場合、その中で問合せオブジェクトからの距離が最大のものが、実解における最大距離の上限を与える。よって *diff* がその値を超える時には五つ組は更新されず、破棄される。

最終的にキューが空になった時点で手続きは終了する。

K-最近傍検索の距離計算を除いた計算量は $O(d\log N)+O((dN)(\log(dN)+d))$、すなわち $O(dN(d+\log(dN)))$ である。d を定数と見做せば $O(N+N\log N)$、すなわち $O(N\log N)$ であり、候補となるオブジェクトの数の上限が k で与えられるならば、$O(k\log k)$ となる。

4 実験

MetricMatrix による近傍検索実験を Debian GNU/Linux 2.2 (Pentium-III, 700MHz, 1GB メモリ) 上で行なった。実装には C++ 言語を用いた。実験には合成クラスタデータを用いた。距離は L_2 距離 (ユークリッド距離) である。即ち d 次元ベクトル x, y 間の距離は

$$\phi(x,y) = (\sum_{i=0}^{d-1} |x_i - y_i|^2)^{\frac{1}{2}}$$

で与えられる。クラスタデータセットは各次元の幅を $[0,1)$ とするユークリッド空間中の点とし、文献 [5] の Appendix H の方法で生成した。これは一点を中心とした正規分布に従うデータ群をクラスタとし、各クラスタの中心はランダムに配置したものである。デー

タ生成のパラメータである分散は $\sigma^2 = 0.1$ とし、クラスタ数は 5 に固定、各クラスタは均等サイズとした。

4.1 実験結果

図 7 に 8 次元 16000 データにおいて r-近傍検索に必要とされた距離計算回数を示す。用いた MetricMatrix の次数は 30 である。図には比較のため GNAT および MVP-tree による結果も示した。GNAT は次数 50、MVP-tree は次数 2 分割数 2 のものを用いた。図から明らかなように、MetricMatrix は他の索引と同程度、あるいはより良い結果を示している。

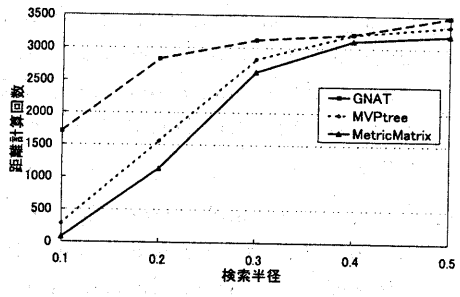

図 7: r-近傍検索: 8 次元 16000 データ

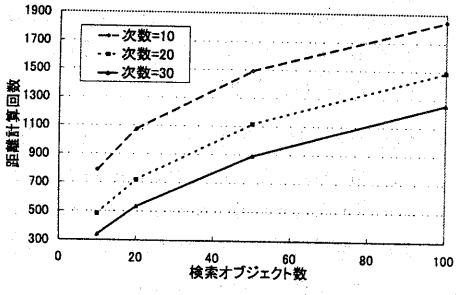

図 8: k-最近傍検索: 8 次元 16000 データ

図 8 は k-最近傍検索に必要とされた距離計算回数である。次数 10、20、30 の各 MetricMatrix を比較した。

図 9 はデータの次元が変化した時の様子である。クラスタデータとともに一様分布データセットにおける結果も示した。図から 16 次元以上になると効率が著しく悪化している事が分かる。

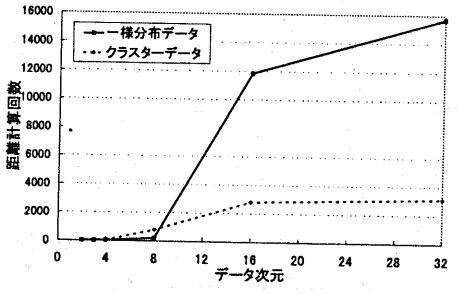

図 9: K-最近傍検索: 次元-距離計算回数

効率についての考察

実験により MetricMatrix が MVP-tree, GNAT と同程度、あるいはより良い効率を示す事が明らかになった。

この事実は距離索引における三角不等式によるフィルタリングの重要性を示している。すなわち、距離索引では空間の分割に基づく木構造の構築とそれを利用した検索時の枝刈りによる探索空間の縮小法は、少なくとも距離計算回数という観点からは、重要ではない。重要なのは三角不等式によるフィルタリングであり、それだけで十分でもある。

したがって距離索引のコストモデルを考えるにあたっても、空間の分割法などを考慮せずとも、三角不等式によるフィルタリング効率のみを考慮すれば良いと考えられる。図 10 に示すのは各次元の一様分布データセットの二点間の距離分布である (全二点対の 1% をサンプリングした)。次元が上がるに従って分布のピークが鋭くなり、同時に Y-軸から遠ざかっている事が分かる。

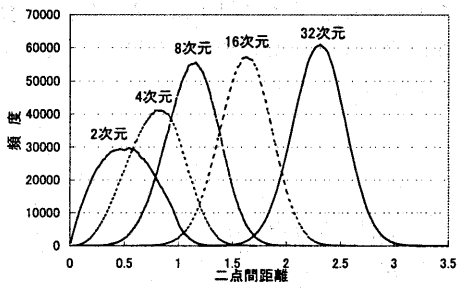

図 10: 一様分布データの二点間距離分布

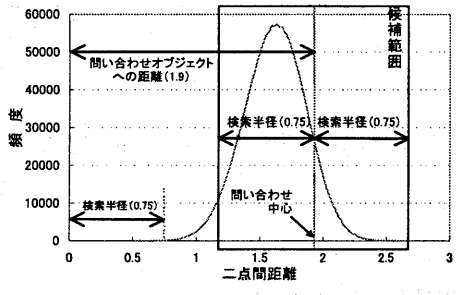

図 11: 16 次元一様分布データにおけるフィルタリング効果

図 11 に示すのは距離分布と候補フィルタリングの関係である。図から、16 次元データにおいて近傍データを検索するためには検索半径を約 0.75 以上にしなければならないことが分かる。距離分布を参照オブジェクトからの距離分布と見做し、問合せオブジェクトの参照オブジェクトからの距離を 1.9、検索半径を 0.75 とすると、中心を 1.9、幅を左右に 0.75 ずつとった範囲が候補範囲となる。このときほとんどのデータが候補となっている。すなわち 16 次元空間では、その距離分布のため三角不等式に基づくフィルタリングは有効でない。これが検索効率が悪化している原因である。

このように距離分布をサンプリングする事で、あるデータセットに対して距離索引が有効か否かを判定する事ができる。

5 まとめと今後の課題

メモリ格納型で静的な距離索引 MetricMatrix を提案した。MetricMatrix が他の距離索引と同程度あるいはより良い効率を示す事を実験により示した。MetricMatrix は構造が単純で応用にも適している。効率に関して、フィルタリング原理に基づいた距離索引の有効性の判断基準を示した。MetricMatrix はフィルタリング原理のみに基づいており、また実装も簡単であるため、距離索引の有効性の判定にも利用できる。

参考文献

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

[2] Bozkaya, T. and Özsoyoglu, Z. M.: Distance-Based Indexing for High-Dimensional Metric Spaces, *Proc. of SIGMOD 1997, May, 1997, Tucson, Arizona, USA*, pp. 357–368 (1997).

[3] Brin, S.: Near Neighbor Search in Large Metric Spaces, *Proc. of VLDB'95, September, 1995, Zurich, Switzerland*, pp. 574–584 (1995).

[4] Ciaccia, P., Patella, M. and Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces, *Proc of VLDB'97, August, 1997, Athens, Greece*, pp. 426–435 (1997).

[5] Jain, A. K. and Dubes, R. C.: *Algorithms for Clustering Data*, Prentice-Hall (1998).

[6] 石川雅弘, 能登谷淳一, 陳漢雄, 大保信夫: 距離索引 MI-tree, 情報処理学会論文誌データベース, Vol. SIG 6(TOD 3), pp. 104–114 (1999).