多次元空間における疎なオブジェクト群を提示する手法について*

5T－7

朝倉 浩志　吉川 正俊†
奈良先端科学技術大学院大学　情報科学研究科‡

1. はじめに

大量のマルチメディアデータから利用者の要求に適合したデータを高速に検索する技術は重要であり，活発に研究が行われている．一般的に，このような検索ではマルチメディアデータが特徴量からなる多次元ベクトルで表現される．従来の研究はこのようなベクトル集合に対し，与えられた問合せ点からの近傍点の検索や範囲検索の高速化が主な研究課題であった．

しかし，探索型のデータベースにおける利用者とデータベースとの対話を考えた場合，単なる近傍点検索や範囲検索だけでは不十分である．デジタル博物館やオンラインショッピングなど，探索型のデータベースでは，サンプルデータ提示と近傍データ検索が，繰り返し行なわれる．サンプルの提示は，デジタル博物館における知的好奇心の喚起，またオンラインショッピングでは購買意欲の喚起につながる重要な機能である．本論文では，サンプルデータ提示のための一手法として大量の多次元ベクトルデータから，相互になるべく離れた複数個のベクトルを木索引を用いて検索するアルゴリズムを与える．

2. 対象とする問合せ

本稿で対象とする問合せは「互いの類似度がなるべく低いデータをk個提示せよ」というものである．サンプルデータの提示は，類似しているデータではなく，類似していないものを複数提示する方が良い．本問合せは，ベクトル空間内において「互いの距離がより離れたベクトル(以下オブジェクトという)をk個選択する」ことであり，疎なオブジェクト群を提示することである．望まれる結果は，問合せ条件を満足するオブジェクト集合を少なくとも一つ返すことである．従来のデータベースのように，条件を満足するデータを全て返すのとは異なる．

3. 木索引による解決

R*-tree[1]やX-treeなどの木索引を用いてこの問合せの処理を考える．議論している問合せは格納されているベクトルの相互距離を利用する必要がある．空間を分割し，索引付けする木索引はこのような点から有効である．この他にも静的なベクトルの相互距離をあらかじめ計算し格納しておく方法が考えられるが，更新や検索するデータサイズなどの問題があり，現実的ではない．

4. 提案するアルゴリズム

4.1 基本的な考え方

木索引を利用し，根から葉に向かって以下に説明する選択アルゴリズムを適用する．このとき問合せで与えられたk個を子に振り分けて行く．振り分けの際には，これから述べる距離を利用することによりなるべく離れたベクトルを選択する．

*Retrieving Sparse Objects in High Dimensional Spaces
†Hiroshi ASAKURA and Masatoshi YOSHIKAWA
‡Graduate School of Information Science, Nara Institute of Science and Technology (NAIST)

4.2 準備

定義 1: 矩形領域

n次元ユークリッド空間の矩形領域Aは次のような対角をなす2つの端点$\boldsymbol{a}$と$\boldsymbol{a}'$によって表現することができる．　$A = (\boldsymbol{a}, \boldsymbol{a}')$

$$\boldsymbol{a} = [a_1, a_2, \ldots, a_n],\ \boldsymbol{a}' = [a'_1, a'_2, \ldots, a'_n]$$
$$(a_i \leq a'_i; i \in \{1, 2, \ldots, n\}).$$

□

以降，矩形領域のことを単に領域と呼ぶ．

定義 2: 定義1のような矩形領域Aに対し，Aの境界を構成する$2n$個の$n-1$次元矩形のことをAの境界矩形(border rectangle)と呼ぶ．□

4.3 いくつかの距離の定義

定義 3: MINDIST

n次元空間内において領域$A = (\boldsymbol{a}, \boldsymbol{a}')$, $B = (\boldsymbol{b}, \boldsymbol{b}')$とその座標値$\boldsymbol{a}$,$\boldsymbol{a}'$,$\boldsymbol{b}$,$\boldsymbol{b}'$が与えられているとすると，領域$A$と領域$B$の最短距離$MINDIST(A, B)$は以下の式で定義される．

$$MINDIST(A, B) = \sqrt{\sum_{i=1}^{n} f_i^2(A, B)}$$

$$f_i(A, B) = \begin{cases} b_i - a'_i & (if\ a'_i < b_i) \\ a_i - b'_i & (if\ b'_i < a_i) \\ 0 & (otherwise) \end{cases}$$

□

定義 4: MAXMINDIST

n次元空間内において領域Aと領域Bの境界矩形がそれぞれ$A_p(1 \leq p \leq 2n)$とB_q $(1 \leq q \leq 2n)$で与えられているとすると，MAXMINDIST(A, B)は以下の式で定義される．

$$MAXMINDIST(A, B) = \max_{A_i \in A, B_j \in B}(MINDIST(A_i, B_j))$$

□

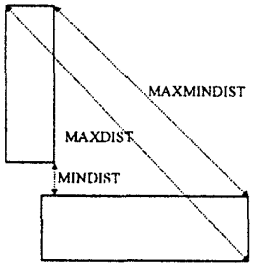

図1 2次元における距離の例

補題 1: n次元空間内の二つの包囲矩形(bounding rectangle) A, Bが与えられた時，

$$\|a, b\| \geq MAXMINDIST(A, B)$$

なるオブジェクト$a(\in A)$と$b(\in B)$が存在する．

略証: 上記のMAXMINDISTの定義において，

$MAXMINDIST(A, B) = MINDIST(A_p, B_q)$

が成立するとする．Aは包囲矩形であるため，任意のA_i $(1 \leq i \leq 2n)$は，オブジェクトを少なくとも一つ包含する(Bについても同様)．MINDISTの定義より，$\forall a \in A_p$と$\forall b \in B_q$に対して$MINDIST(A_p, B_q) \leq \|a, b\|$.

□補題1は，A内のオブジェクトとB内のオブジェクトの対のうち，距離が$MAXMINDIST(A, B)$以上離れたものが必ず存在することを保証する．

4.4 選択アルゴリズム

節 Nにおいて，k個の領域を選択することを目的とする．

4.4.1 準備

木索引中の節 Nが子として持つ全ての領域の組合せについて，MINDIST，MAXMINDIST それぞれを計算する．結果は後で述べるアルゴリズムで利用するためにバケツと呼ぶデータ構造に保持する．バケツ Bは，大きさ $k(k-1)/2$ の距離配列 $D[i,j](1 \leq i < j \leq k)$ と制約付領域集合 Rの対から成る．ある二つの領域 R_a, R_b(一般性を失うことなく $a < b$) に対して MINDIST, MAXDIST ごとに次のようなバケツを作成する．

- MAXMINDIST(R_a, R_b) の場合，R_{a_p},B_{b_q}をそれぞれ MAXMINDIST(R_a,R_b) = MINDIST(R_{a_p},B_{b_q}) が成立する R_a, R_bの境界矩形であるとすると，
  $$D[a,b] = MAXMINDIST(R_a, R_b)$$
  $$D[i,j] = -1 \quad ((i \neq a) \text{ or } (j \neq b))$$
  $$R = \{(R_a, R_{a_p}), (R_b, R_{b_q})\}$$
  とする．
- MINDIST(R_a, R_b) の場合には，
  $$D[a,b] = MINDIST(R_a, R_b)$$
  $$D[i,j] = -1 \quad ((i \neq a) \text{ or } (j \neq b))$$
  $$R = \{(R_a, *), (R_b, *)\}$$
  とする．$MINDIST(R_a, R_b)$ は領域 R_aと領域 R_bの任意の 2 点間の距離が MINDIST 以上であることから$*$を持つ．

このようにして得られたバケツを初期バケツと呼ぶことにする．初期バケツはその作成方法から，ある一つの距離 $D[a,b]$ と大きさが 2 の制約付領域集合 $\{(R_a, c_a), (R_b, c_b)\}$ で表現できる．

次に，$D[a,b]$ の値をキーに降順にソートする．MINDIST，MAXMINDIST は種類の区別なく，両方混在してソートされる．

例として 2 次元空間において領域 A～Eを子として持つ節 Nを考える (図 2)．領域 A～Eはそれぞれ x 軸から見て原点に近い辺から半時計周りに 1～4 の番号を振ってあり，領域と組にして制約付領域集合を表現する．準備を行ったデータは表 1 のようになる．

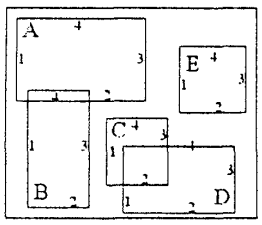

図 2 節 N

$D[a,b]$	制約付領域集合 R
0.83	{(B,1), (E,3)}
0.74	{(A,4), (D,2)}
0.72	{(A,4), (B,2)}
0.67	{(D,2), (E,4)}
0.62	{(A,1), (E,2)}
0.59	{(B,4), (D,3)}
0.53	{(C,1), (E,3)}
0.51	{(C,2), (E,4)}
0.46	{(C,1), (D,3)}
0.34	{(A,1), (C,2)}
0.32	{(B,*), (E,*)}
0.13	{(B,4), (C,3)}
⋮	⋮

表 1 ソートされた距離

4.4.2 選択

選択は図 3 のようなアルゴリズムに従う．準備で作成したリストの上位，即ち $D[a,b]$ の値が大きい順に評価を行う．下記アルゴリズム中の size(B') が $\frac{1}{2}k(k-1)$ になれば，B'が選択され，アルゴリズムは終了する．初期バケツ $B_I = (D_I, R_I)$ とバケツ $B = (D, R)$ が与えられた時，関数merge(B_I, B) は，次のように定義される．

```
ChooseChildNodes
 入力: ソート済み初期バケツリスト L = [L[1], L[2], ...]
       選択するオブジェクトの個数 k
 出力: バケツ B'
 処理:
{
   B = {⊥};
   i = 1;
   for L[i] ∈ L {
      foreach B ∈ B {
         B' = merge(L[i], B)
         if size(B') = k(k-1)/2 return B'
         else B = B ∪ B'
      }
      B = B ∪ {L[i]}
      i = i +1
   }
}
```
図 3 選択アルゴリズム

1. $D[a,b] \neq -1$ の場合，merge(B_I, B) = $\{\perp\}$ とする．
2. 上記以外の場合は以下を実行する．
   まず，二つの制約付領域 (R_a, c_a), (R_b, c_b) に対し，merge($(R_a, c_a), (R_b, c_b)$) を以下のように定義する．

   $$\text{merge}((R_a, c_a), (R_b, c_b)) = \begin{cases} \{\} & (R_a \neq R_b \text{のとき}) \\ \{\perp\} & (R_a = R_b \text{かつ } c_a \text{と } c_b \text{が異なる矩形領域のとき}) \\ \{(R_a, c_a)\} & (R_a = R_b \text{かつ } c_b \text{が*のとき}) \\ \{(R_b, c_b)\} & (R_a = R_b \text{かつ } c_a \text{が*のとき}) \\ \{(R_a, c_a)\} & (R_a = R_b \text{かつ } c_a = c_b \text{のとき}) \end{cases}$$

3. 次に，二つの制約付領域集合 R, Sに対して，merge(R, S) を次のように定義する．

   $$\text{merge}(R, S) = \begin{cases} \{\perp\} & (if\ \exists r \in R, \exists s \in S \text{ s.t. } \text{merge}(r,s) = \{\perp\}) \\ \bigcup_{r \in R, s \in S} \text{merge}(r,s) & (otherwise) \end{cases}$$

4. merge(B_I, B) = (merge(D_I, D), merge(R_I, R)) ただし，

   $$\text{merge}(D_I, D) = [max(D_I[i,j], D[i,j])] \quad (1 \leq i < j < m)$$

   また，merge(R_I, R) = $\{\perp\}$ の場合は，merge(B_I, B) = $\{\perp\}$ とする．

また，バケツ $B = (D, R)$ に対してsize(B) は，$D(i,j) \neq -1$ なる要素 $D(i,j)$ の数とする．

上記のアルゴリズムを図 2 で挙げた節 Nに適用すると，$\{(B,*), (D,*)\}$ を評価した時点で終了し，答えの 3 領域は $(A,4), (B,2), (D,2)$ となる．

5. まとめ

本稿では，多次元空間における疎なオブジェクト群を選択する手法について述べた．現在，R*-tree を利用して本アルゴリズムを実装中である．今後，実験を通じて本アルゴリズムの有効性を検証する予定である．

参 考 文 献

1) Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 322–331, May 1990.