2K-2

索引木の均衡を考慮した類似検索索引手法

Similarity Search Index Based on Both Pruning and Balance

倉沢 央† 深川 大路‡ 高須 淳宏‡ 安達 淳‡

Hisashi Kurasawa Daiji Fukagawa Atsuhiro Takasu Jun Adachi

東京大学 大学院 / The University of Tokyo†

国立情報学研究所 / National Institute of Informatics‡

1. はじめに

類似検索は，膨大なオブジェクトの中からクエリに似たオブジェクトを効率的に探す技術である [1]．メトリック空間を対象とした類似検索索引は，オブジェクト間の距離（類似度）が距離の公理を満たすものであれば，いかなるオブジェクトも扱える．例えば，ベクトルや集合データ，文字列は，それぞれユークリッド距離や Jaccard 係数距離，編集距離が適用でき，すべてこの索引で扱える．我々は類似検索の問い合わせ処理コストの削減を目標に，類似検索索引構造について研究している．

類似検索索引は，問い合わせオブジェクトから距離の遠いオブジェクトを，三角不等式などの距離の公理を利用して判別し，枝刈りするのに利用される．最も使われる索引付け手法は *Pivot partitioning* である．これは，*pivot* と呼ばれる参照オブジェクトからの距離でオブジェクトセットを複数の部分空間に分割する手法である．多くの索引は Pivot partitioning で再帰的に空間を分割し，木構造を作る．Pivot partitioning による枝刈り性能は，どのオブジェクトを pivot に設定するか，そしてどの距離で分割するか，という 2 つの要素に大きく左右される．初期の研究は索引木のバランスを重視して完全二分木となる分割距離を設定し，pivot は空間の端から選ぶという経験則に基づいたものが多かった [2]．これに対して，近年の研究は 1 つの pivot あたり枝刈りできるオブジェクト数を多くすることを重視し，pivot とその分割距離をデータのクラスタ構造に基づいて選ぶものが多い [3, 4]．これらの検索索引の性能を比較すると，優劣がデータの分布に大きく依存することがわかる．つまり，これまで提案された手法だけでは，ユーザは扱うデータの分布を見て，それを得意とする索引を選ばねばならない．

そこで，我々は索引木のバランスと枝刈りの効果の両方を考慮することによって，様々なデータの分布に対処できる検索索引の開発を目指した．我々はまず，pivot と分割距離に対する木のバランスと枝刈りについての評価関数，Pivot capacity を定義した． そして，この評価関数に基づいた検索索引，Pivot Capacity Tree (PCTree)を開発した．評価実験により先行研究よりも高速に検索できることがわかった．また，PCTree はデータの分布によって索引木の均衡を適切に設定できることがわかった．

2. Pivot Capacity Tree

PCTree は索引木のバランスと枝刈りの効果の両方を備えた類似検索索引である．PCTree は，Pivot capacity の値を最大にする pivot と分割距離で空間を再帰的に分割して，索引を構築する．

2.1. Pivot capacity

Pivot Capacity（PC）は，pivot によって分割される部分空間中のオブジェクト数のバランスと，pivot によって枝刈りできるオブジェクト数についての評価関数である．この関数で得られる値は既知のクエリ分布に対して期待できる索引性能を表す．本稿ではクエリはデータと同じ分布で生成されると仮定する．

まず，2 つの確率変数を定義する．メトリック空間 *M=(D, d)* において，pivot p とその分割距離 r_p が D を 2 分割しているとする：

$$R_{1,p,r_p} = \{o \in D \mid d(o,p) \le r_p\},$$

$$R_{2,p,r_p} = \{o \in D \mid d(o,p) > r_p\}.$$

$R_{1,p,rp}$，$R_{2,p,rp}$ に属するクエリを空間分割についての確率変数 X でそれぞれ $X_{1,p,rp}$，$X_{2,p,rp}$ と定義すると，クエリが部分空間のどちらに含まれるかを示す確率は，オブジェクト集合 S を用いて表せる：

$$P(X = X_{1,p,r_p}) = \frac{|\{o \in S \mid o \in R_{1,p,r_p}\}|}{|S|},$$

$$P(X = X_{2,p,r_p}) = \frac{|\{o \in S \mid o \in R_{2,p,r_p}\}|}{|S|}.$$

同様に，枝刈りについての確率変数を求める．オブジェクト集合 S におけるオブジェクト o から，空間 D において S の中の k 近傍オブジェクトまでの距離を $r_{o,k,S}$ とおく．すると，D を以下のように 3 分割できる：

$$R'_{1,p,r_p,k} = \{o \in D \mid d(o,p) + r_{o,k,S} \le r_p\},$$

$$R'_{2,p,r_p,k} = \{o \in D \mid d(o,p) + r_{o,k,S} > r_p\},$$

$$R'_{3,p,r_p,k} = D - R'_{1,p,r_p,k} - R'_{2,p,r_p,k}.$$

$R'_{1,p,rp}$，$R'_{2,p,rp}$，$R'_{3,p,rp}$ に属するクエリを空間分割についての確率変数 Y でそれぞれ $Y_{1,p,rp}$，$Y_{2,p,rp}$，$Y_{3,p,rp}$ と定義すると，

$$P(Y = Y_{1,p,r_p,k}) = \frac{|o \in S \mid o \in R'_{1,p,r_p,k}|}{|S|},$$

$$P(Y = Y_{2,p,r_p,k}) = \frac{|o \in S \mid o \in R'_{2,p,r_p,k}|}{|S|},$$

$$P(Y = Y_{3,p,r_p,k}) = \frac{|o \in S \mid o \in R'_{3,p,r_p,k}|}{|S|},$$

で表せる．これら 2 つの確率変数を用いて，pivot p の PC は以下のように定義される．

$$PC_k(p) \equiv \max_{r_p} I(X;Y),$$

$$= \max_{r_p}\left(\sum_i \sum_j \left(P\left(X_{i,p,r_p}, Y_{i,p,r_p}\right) \log\left(\frac{P\left(X_{i,p,r_p}, Y_{i,p,r_p}\right)}{P\left(X_{i,p,r_p}\right) \cdot P\left(Y_{i,p,r_p}\right)}\right)\right)\right),$$

ここで，*I(•;•)*は相互情報量を表す．この式が示すように，PC では pivot 候補に対して最適な分割距離を自動的に決める．また，相互情報量の定義から，PC は 2 つそれぞれの確率変数の情報量以下の値となる．

2.2. 索引付け

PCTree の索引付けは，まず，pivot 候補集合の生成からはじめる．索引対象のオブジェクト集合 S から過去にサンプリングした pivot 候補よりも距離が s 以上離れたオブジェクトで最近傍のものを再帰的にサンプリングし，S_p とする．そして，分割対象の空間 D から m 個のオブジェクトをランダムサンプリングして S_m とする．次に，S_m 中のオブジェクト間で k 近傍オブジェクトまでの距離を事前に求める．そして，S_p の pivot 候補それぞれについて S_m に対しての PC を計算し，最も値が大きいオブジェクトを pivot p とする．p によって D を 2 分割し，それぞれの部分集合に対して同様の処理を繰り返す．空間の分割は pivot の効果が低くなるまで行う．具体的には，pivot 1 つ用いてもオブジェクト集合から 1 つのオブジェクトしか分類できないときの PC の値を基準とする．分割を停止する際には，ランダムに選んだオブジェクトを pivot として，その pivot からの距離とともにオブジェクトを整理する．

索引は木構造で表される．internal ノードは pivot と分割距離，子ノードへのポインタ を，leaf ノードは pivot とその pivot からの距離とオブジェクトのペアの集合を保持する．索引付けコストはデータの分布に依存するが，最悪 $O(N^2)$必要となる．

2. 3. 近傍検索

PCTree は近傍検索と範囲検索を扱う．ここでは近傍検索のみを図 1 に擬似コードで示す．PCTree は，三角不等式で internal ノードを枝刈りするとともに，leaf ノードでも各オブジェクトを pivot からの距離で枝刈りする．

```
Function kNNSearchSub(Node node, Query q, Neighbor k,
Result R)
if node.flag = 1 then
  R.obj.add(SearchBucketNN(node.bucket,
  d(node.query,q)),R.dist)
  if |R.obj| > k then
    SizeRefine(R.obj,k)
  end if
  if |R.obj| = k then
    R.dist←MaxDist(R.obj)
  end if
else
  dist ← d(node.query,q)
  if dist ≤node.dist then
    kNNSearchSub(node.in,q,k,R))
    if Size(R.obj)<k or dist+R.dist≥node.dist then
      kNNSearchSub(node.out,q,k,R))
    end if
  else
    kNNSearchSub(node.out,q,k,R))
    if Size(R.obj)<k or dist−R.dist<node.dist then
      kNNSearchSub(node.in,q,k,R))
    end if
  end if
end if
return

Function kNNSearch(Node node, Query q, Neighbor k, Re-
sult r)
Result.obj ← empty
Result.dist ← ∞
return kNNSearchSub(node,q,k,Result)
```

図 1. PCTree の近傍検索

3. 評価実験

我々は，PCTree の評価を行った．データセットには，人工的に生成したベクトルと，flickr から収集した 200 万件の画像から 960 次元の gist 特徴量を抽出したベクトルを使用した．人工ベクトルは 32 次元，分散値が 0.02，クラスタ数 100 の混合正規分布 10 万件を基準に，次元数を 2 から 64，クラスタ数を 10 から 200，データサイズを 1000 から 50 万件に変化させたものを生成した．クエリは索引付け対象のデータセットと同じ分布で重複のないベクトル 1,000 件を用いた．比較手法には，Metric Space Library [5]にて実装されている GHT [6]，MVP[7]，LC [8]，そして SAT [9]を用いた． 手法の評価には，[8]と同様に，Naïve なシークエンシャルスキャンで検索に必要な距離計算回数を 1 としたときの比を用いた．また，索引木の均衡を知るため，索引木の平均長と最大長も出力した．

データサイズを変化させたときの結果が図 2，次元数を変えたときの結果が図 3，クラスタ数を変えたときの結果が図 4，そして flickr のデータに対する結果が図 5 である．提案手法は低次元でクラスタ化したデータに対して効果的なことがわかった．また，また，図 3(b)から低次元で枝刈りが比較的容易な分布のデータに対してはバランスに重きが置かれた索引木を，高次元なデータに対しては枝刈りに重きが置かれた索引木を構築することがわかった．一方で，クラスタ数が極度に多くなりデータの分布の偏りが小さいときの性能は，LC とほぼ変わらないか若干劣ることもわかった．

4. おわりに

本稿では類似検索索引 PCTree について述べた．PCTree はバランスと枝刈りの両方を考慮した類似検索索引である．評価実験から多様な分布のデータに対して有効であることを確認した．今後は PCTree の索引づけコストを削減する改善を行いたい．

参考文献

[1] P. Zezula, et al. *Similarity search: The Metric Space Approach*. Springer-Verlag, 2005.
[2] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, 1993.
[3] H. V. Jagadish, et al. iDistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. On Database Systems*, 30(2):364-397, 2003
[4] H. Kurasawa, et al. Maximal metric margin partitioning for similarity search indexes. In *CIKM*, 2009.
[5] Metric spaces library, http://www.sisap.org/metric_space_library.html.
[6] J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175-179, 1991.
[7] T. Bozkaya, et al. Indexing large metric spaces for similarity search queries. *ACM Trans. On Database Systems*, 24(3):361-404, 1999.
[8] E. Chevez et al. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 24(9):1363-1376, 2005.
[9] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28-46, 2002.

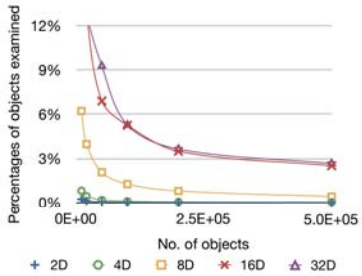

図 2. データサイズに対する距離計算回数

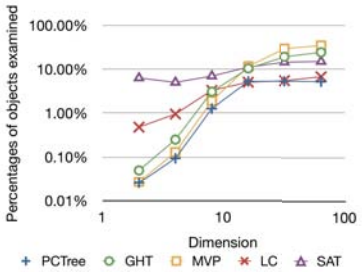

図 3(a). 次元数に対する距離計算回数

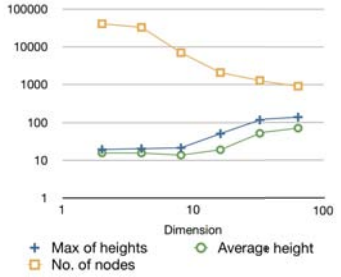

図 3(b). 次元数に対する索引木の高さ

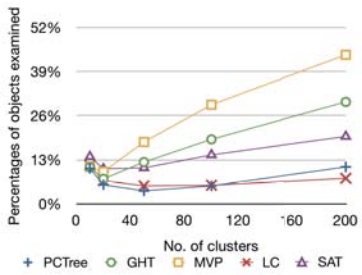

図 4(a). クラスタ数に対する距離計算回数

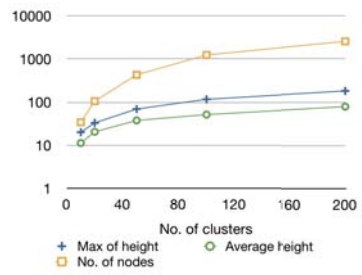

図 4(b). クラスタ数に対する索引木の高さ

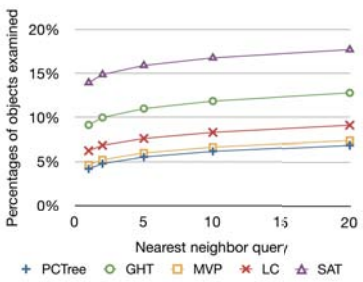

図 5. flickr データに対する距離計算回数