DEIM Forum 2009 D8-4

k-NN 問題に対する効率的な関数索引手法

劉　健全† 陳　漢雄† 古瀬　一隆† 大保　信夫†

† 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻
〒 305–8577 茨城県つくば市天王台 1–1–1
E-mail: †{ljq,chx,furuse,ohbo}@dblab.is.tsukuba.ac.jp

あらまし　本論文では，高次元データの問い合わせ処理における近似検索の k-NN 問題を解決するための新たな索引手法を提案する。$0 < p < 1$ であるような p を用いた L_p 距離による k-NN 検索では，p の冪乗の計算コストが性能低下の大きな原因となる。この問題を避けるため，提案手法は関数索引と上界・下界の概念を組み合わせた手法を採っている。関数索引により検索時に p の冪乗計算をすることなく上界・下界の値を求めることが可能となり，この上界・下界を用いることにより検索対象のデータ集合から k-NN 検索の解となり得ない多くのデータを排除することが可能となる。これにより冪乗計算の回数を大幅に減らすことができるため，検索時間の向上が実現できる。提案手法の有効性については，合成データと実データを用いた評価実験の結果によって確認している。

キーワード　k-NN，問合せ処理，関数索引，近似検索

1. はじめに

近年，高次元データの問合せ処理に対する効率的な近似検索手法の確立がデータ工学研究領域での 1 つ重要な研究課題となっている。この研究課題についてこれまでに提案されたものとしては，例えば, [5] [7] [9] などが挙げられる。高次元空間を対象として扱う場合，一般に「次元の呪い (dimensionality curse)」と呼ばれる高次元ならではの性質により，検索性能が低下するという問題が起こる。この性質のため，簡単な連続スキャン手法と比較して，空間分割やデータ分割などの通常の索引技術ではむしろ効果が悪くなることが指摘されている [3]。この問題を避けるため，高次元空間索引としては，連続スキャン手法をベースにその高速化を目指したものが提案されている。こういった手法には，例えば，VA-file やそれを改良したものなどがある [2] [6] [7] [9]。

高次元空間データに対する近似検索については，どのような距離に基づく距離空間を使うのが効果的であるかという点に関する議論もある [1] [4] [8] [10]。ベクトルデータに対する検索では，L_p 距離がよく使われている。p は任意の値をとることができるが，一般に使われているのは $p = 1$ と $p = 2$ である。これに関して [1] では，高次元データにおける近似検索では p 値の変化によって結果が大きく変わることが示した。特に，p が 1 未満の小数の場合に高次元データに対する近似検索の結果がよくなるとの指摘がなされている。しかし，p が整数の場合と比較して p が整数でない場合の L_p 距離の計算量は極めて大きくなるため，検索処理が低下するという問題が生じる。

このような問題を解決するため，本論文では，p が小数の場合の L_p 距離計算のコストを削減する新たな関数索引手法を提案する。本論文の構成は以下の通りである。第 2 章で本研究における問題設定に関する定義や条件を示す。第 3 章では，本研究の提案手法を詳しく説明する。続いて，第 4 章で関数索引に基づく k-NN アルゴリズムを提案し，第 5 章で評価実験によって本提案手法が効率向上であることを示す。最後に，本論文での内容を簡単にまとめ，今後の課題を述べる。

2. 問題設定

本論文では，高次元近似検索における k-NN(k Nearest Neighbors) 問題を主に対象とする。本章では，k-NN 問題に対する近似検索の L_p 距離や，上界，下界等の定義を示す。

2.1 L_p 距離

本論文では高次元空間でのベクトル距離を計算するために，L_p 距離を用いる。以下に L_p 距離の定義を示す。

定義：高次元空間の点 X_i を，

$$X_i = (x_{i1}, x_{i2}, \cdots, x_{iD})^T, \quad i = 1, 2, \cdots, n;$$

とする。高次元空間上の 2 点 X_i と X_k との L_p 距離を，次元数 D とパラメータ p を用いて以下のように表す。

$$L_p(X_i, X_k) = \left(\sum_{j=1}^{D} |x_{ij} - x_{kj}|^p \right)^{\frac{1}{p}}$$

本研究では，p を区間 $(0, 1)$ 内の値とする。また，全ての点に対して，$x_{ij} \in [0, 1)$ とする。

2.2 上界と下界

ある順序集合 X の部分集合 $A \subseteqq X$ について考える。X の要素 x_0 が A の上界であるとは A の任意の α に対して $\alpha \leqq x_0$ であることを言う。

定義：x_0 が A の上界である $\Leftrightarrow \forall \alpha \in A$ について $\alpha \leqq x_0$ が成り立つ。

同様に，順序集合 X の要素 y_0 が A の下界であるとは A の任意の α に対して $y_0 \leqq \alpha$ であうことを言う。

定義：y_0 が A の下界である $\Leftrightarrow \forall \alpha \in A$ について $y_0 \leqq \alpha$ が成り立つ。

2.3 *k*-NN 問題

高次元検索では，空間上にある高次元データに対して，問合せキーとの距離を比較することで最近接点などの一定の条件を満たすものを検索する。高次元に対する近似検索の代表なものに，k-NN 検索がある。

V をベクトルデータの集合とし，q を問合せベクトルとする。ここで，D は次元数である。

$$V \subseteqq [0,1)^D,\ u, v \in V,\ q \in [0,1)^D$$

この時，k-NN 検索は以下のように定義される。

定義：k-NN 検索の問合せベクトル q と検索個数 k (k は自然数) が与えられた時，q との距離が最も短い k 個のベクトルの集合 ($V_k \subset V$) を検索する。

$$\forall v \in V_k \forall u \in (V - V_k), L_p(q,v) < L_p(q,u) \text{ かつ}$$
$$|V_k| \geqq k \wedge |V_k - arg\ max_{v \in V_k}\{L_p(q,v)\}| < k$$

p が非整数の L_p 距離によってこの定義のように k-NN 検索を行う場合，ベクトル間の距離計算のたびに p の冪乗計算を D 回行うのは非常に大きな計算量となる。このことに伴う検索性能の低下を回避するため，本論文では関数索引を用いる手法を提案する。

3. 提 案 手 法

本章では実例を用い，本研究でのモチベーションとアプローチを説明する。

まず，上界・下界の概念を用いた関数索引の手法を図 1 を用いて述べる。ここでは，正規化された 2 次元空間を仮定し，距離計算は L_p 距離を使うものとする。p は非整数の $(0 < p < 1)$ の範囲をとるものと仮定する。このとき，k-NN 検索のための $(\cdot)^p$ の冪乗の計算量は非常に大きいので，検索処理時にこのような計算を行うのをなるべく避けたいという考え方が本提案手法のモチベーションとなっている。

図 1(b) のように座標の区間 $[0,1)$ を 10 ブロックに等分割すると，それぞれの分割点は $t/10$ となる。ここで，配列 $c[t] = (t/10)^p \quad (t = 0, 1, \ldots, 10)$ を用意する。このときに必要となる p の冪乗の計算回数は (自明な $c[0] = 0$ および $c[10] = 1$ の 2 つの計算を除くとすると)9 回である。この配列 $c[t]$ の 11 個の値と上界・下界の概念を組み合わせることにより，効率的な近似検索が可能となる。以下に詳細に説明する。

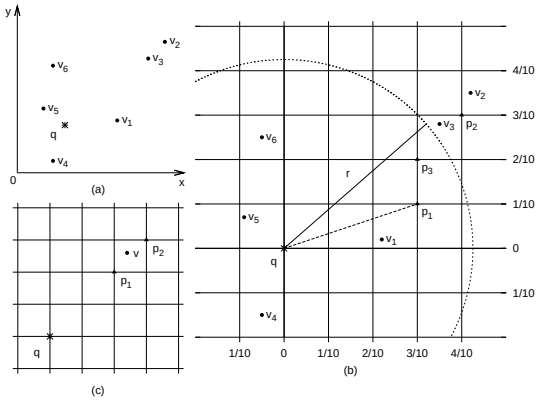

図 1　関数索引の 2 次元例

あらかじめ算出した配列 $c[\cdot]$ をグリッド網の “knots” と言う。knots の交点に位置する 2 点間の L_p 距離の p 乗 L_p^p は，配列 c を使って簡単に求めることができる。例えば，図 1(b) の点 q と $p1$ の場合には，$L_p^p(q, p1) = c[3] + c[1]$ となる。

グリッド網上では，任意の 1 つ 2 次元ベクトルは，必ずいずれか 1 つの四辺形セルに所属する。四辺形セルの角の knots は 4 個であるが。そのうちの 2 つが問い合わせベクトル q と検索対象ベクトル v の距離の上界・下界となる。例えば，図 1 の (c) では，ベクトル q に対するベクトル v の上界は p_2 で，下界は p_1 であるから，$L_p(q, p1) \leqq L_p(q, v) < L_p(q, p2)$ となり，したがって距離 $|q - v|$ の上界は $L_p(q, p_2)$，下界は $L_p(q, p_1)$ となる。ここで，図 1 は提案手法の考え方を説明するために用いたものであり，本論文の提案手法で実際に検索する際にベクトル空間を分割する必要があるわけではなく，また，問い合わせベクトルを原点に置く必要もないことに注意されたい。

図 1 の (c) のような場合，あるベクトルデータを $v = (v.x, v.y)$ に対して問い合わせ q との距離が $L_p(q, v) \leqq r$ であるか否かを以下のように確認することができる。q と v の距離は

$$L_p(q, v) = \sqrt[p]{|q.x - v.x|^p + |q.y - v.y|^p}$$

であるが，これを直接計算せずにこれまでに述べた考え方に基づいてより少ない計算量で確認することができる。まず，$|q.x - v.x|$ を δ_x，$|q.y - v.y|$ を δ_y と表記することとする。$0 \leqq \delta_x < 1$ が成り立つことは明らかであるから，$\frac{t_x}{10} \leqq \delta_x < \frac{t_x+1}{10}$ を満たすような $t_x \in \{0, 1, \ldots, 9\}$ が存在する。このような t_x については，$c[t_x] \leqq \delta_x^p < c[t_x + 1]$ が成り立つ。同様に，$c[t_y] \leqq \delta_y^p < c[t_y + 1]$. を満たすような t_y も必ず存在する。以上より，

$$l_b \equiv c[t_x] + c[t_y] \leqq L_p^p(q, v) = \delta_x^p + \delta_y^p$$
$$< c[t_x + 1] + c[t_y + 1] \equiv u_b$$

となる。

ここで，l_b を距離 $|q-v|$ の下界，u_b を距離 $|q-v|$ 上界となる。したがって，$l_b > r^p$ を満たすなら，q と v の実距離は必ず r^p より小さいので，v は答えではないとして安全に排除することができる。逆に，$u_b \leqq r^p$ を満たすなら，v は解の 1 つであることが確実であるので，直ちに集合に追加することができる。以上の処理を行うことにより，実際の L_p 距離を計算することなく，あらかじめ算出した配列 $c[\cdot]$ のうちで $O(1)$ の計算量で走査するだけで距離の上界と下界は簡単に判断できる。この手法を用いれば，計算量の大きい p の冪乗計算回数を大幅に削減することが可能となる。

次に，2-NN 検索を例として，提案する関数索引手法を説明する。図 2 では，データベース V が 2 次元ベクトルデータ $v_1, v_2 \ldots, v_8$ の 8 個で構成されている。ここで，V から問合せベクトル q との 2-NN 検索を行う。本提案手法では，k-NN 検索の処理を 2 段階に分ける。第一段階では，V から解にならないベクトルを除く処理を行う。このフィルタリング処理は上下界のみを使って，ベクトルデータと q の距離は 2-NN の解になり得るか否かを判断する。第二段階では，第一段階で排除できなかったベクトルデータに対して，L_p 距離を実際に計算し，最終的な k-NN の解を求める。

この例での 2-NN を検索は，以下のように処理される。まず，先に述べたようにあらかじめ算出した配列 $c[\cdot]$ から，v_1, v_2 の下界を得る。これらはそれぞれ，$l_b(v_1)$ と $l_b(v_2)$ と表すこととする。この段階で，v_1, v_2 を候補として解候補集合へ追加する。更に，それらの上界値と下界値を昇順でそれぞれソートする。ここまでの処理により，$k(=2)$ 個の解候補が既に存在する。そこから先は，データベースで残りのベクトルを順次調べて，既存の解候補とするか否かを判断する。

v_1, v_2 の処理が終わったら，次に v_3 を見る。下界 $l_b(v_3)(\equiv L_p^p(q, p1))$ はこの時点での解候補集合での最大の上界値 (v_1 の上界)$u_b(v_1)(\equiv L_p^p(q, p4))$ よりも小さいので，v_3 は解になる可能性がある。よって，v_3 を解候補集合へ追加する。次に，v_4 を見てみると，$l_b(v_4) \geqq max\{u_b(v_2), u_b(v_3)\}$ を満たすことから v_4 は解になり得ないので，解候補から排除する。これは，v_4 と q の実距離 $L_p(q, v_4)$ は v_2 または v_3 との距離より小さい可能性がないので，q との 2-NN になる可能性がないからである。

同様に，v_5 や v_7，v_8 は解候補から除外することになる。しかし，v_6 の下界は v_5 の下界より小さいので，v_6 は解候補集合に残す。

以上のフィルタリング段階での処理が終わると，v_1, v_2, v_3, v_6 が候補解として残っている。第二段階では，それぞれと q の L_p 距離を実際計算しなければならない。この結果，v_3 と v_6 が 2-NN の最終的な解として得られることとなる。

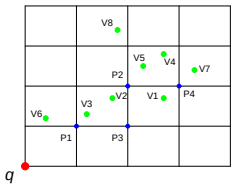

図 2　2-NN 検索の 2 次元例

4. アルゴリズム

本章では，提案手法に基づく汎用的な k-NN アルゴリズムの詳細を説明する。前章で述べたように，k-NN アルゴリズムの処理は 2 段階に分けられる。それぞれを**フィルタリング**と**リファインメント**と言う。

フィルタリング段階では，$\forall v_1, v_2 \in V, l_b(v_1) \geqq u_b(v_2) \Rightarrow L_p(q, v_1) \geqq L_p(q, v_2)$ という性質を利用してフィルタリングを行う。ベクトル v を見るときに，既存の k 個候補解の最大の上界値 $u_b(v_k)$ に対して，$l_b(v) \geqq u_b(v_k)$ を満たすなら，v を安全に排除することができる。一方，もしある既存候補解の下界が v の上界より大きければ，これを捨てて v を入れ替える。このような処理の流れを詳細に記述するのがアルゴリズム 1 である。このアルゴリズムでは，まず，最初の k 個ベクトルをそれらの下界と共にヒープ $Cand$ に入れる。同時に，それらの k 個の上界もヒープ H_u に挿入する。H_u はソートされ，サイズが k に固定される。3 行目から 7 行目までのように k 個の候補解を確保することができる。次のベクトルを見ていくと，最大の上界値 $H_u[k]$ と比べて (10 行目)，このベクトルの下界が $H_u[k]$ を超えなければ，$Cand$ へ追加する。同時に，$u_b(v)$ を H_u へ入れて (12 行目)，ヒープを保持するため，調整しながら要素の交換することも発生する。

フィルタリング段階で得られる解候補ベクトルは k 個より多いので，最終的な解を得るためにリファインメント段階で q との実距離を検査しなければならない。この処理を効率良く行うのがアルゴリズム 2 である。このアルゴリズムではまず，k 個の解を確保するため，$Cand$ から最初の k 個を解として，解集合 V_k に入れる (3 行目から 7 行目まで)。V_k は $L_p^p(q, v_i)$ によってソートされる。解候補集合 $Cand$ の残りのベクトルと V_k の距離をを比較する代わりに，8 行目から 12 行目までのような処理手順を用いることで，解集合に含まれるベクトルの入れ替えを効率良く判断する。ここでは，

$$L_p^p(q, v_j) > l_b(v_j) \geqq l_b(v_i) > L_p^p(q, v_k)$$

を満たす v_j は解になり得ないという性質を利用している。

5. 実験と評価

本論文の提案手法の有効性を確認するため，実験を行った。実験環境は以下の通りである。

- CPU: Intel(R) Xeon (TM) 2.80 GHz

Algorithm kNN-filtering
Input: Vector set V and query vector q, B and k.
Output: Candidate set and bounds $Cand$

begin
1: create heaps $Cand$ and H_u
2: $c[t] \leftarrow (t/B)^p$ for $t = 1, 2, \ldots, B-1$.
3: **foreach** $v \in \{v_1, v_2, \ldots, v_k\}$
4: $(l_b(v), u_b(v)) \leftarrow$ get-bound(v, q, c)
5: insert $(v, l_b(v))$ into $Cand$ // $l_b(v)$ as sorting key
6: insert $u_b(v)$ into H_u // H_u is sorted
7: **end foreach**
8: **foreach** $v \in V \backslash \{v_1, v_2, \ldots, v_k\}$
9: $(l_b(v), u_b(v)) \leftarrow$ get-bound$(v, q, c[B])$
10: **if** $l_b(v) \leqq H_u[k]$ **then**
11: insert $(v, l_b(v))$ into $Cand$
12: insert $u_b(v)$ into H_u
13: **end if**
14: **end foreach**
15: **return** $Cand$
end

Algorithm 1: k-NN のフィルタリング処理

Algorithm kNN-refinement
Input: V, q, B, k.
Output: Answer set V_k (in heap structure).

begin
1: $Cand \leftarrow$ **Call** kNN-filtering
2: $V_k \leftarrow \emptyset$
3: **for** $i = 1, 2, \ldots, k$
4: $(v, l_b(v)) \leftarrow Cand[i]$
5: compute $L_p^p(q, v)$ //the real distance
6: insert $(v, L_p^p(q, v))$ into V_k
7: **end for**
8: **for** $(i = k+1; i \leqq$ size-of$(Cand)$; i++)
9: $(v, l_b(v)) \leftarrow Cand[i]$
10: **if** $l_b(v) > L_p^p(q, v_k)$ of $V_k[k]$ **then break**
11: **else** insert $(v, L_p^p(q, v))$ into V_k
12: **end for**
13: **return** $V_k[i], i = 1, 2 \ldots, k$
end

Algorithm 2: k-NN のリファインメント処理

- 主記憶: 3.2 GB
- 実装言語: C++

実験データには一様分布の合成データと Corel Image Features from UCI KDD Archive(注1) の実データを用いた。

上記の環境で合成データと実データのそれぞれについて，k-NN 検索の評価実験を行った。今回，本研究での問題設定に基づき，通常の k-NN 検索における計算方法と，本論文で提案する関数索引手法との計算コストを比較した。以下にその結果を示す。

図 3 は，合成データに対する検索の計算コストの比較である。パラメータ B は関数索引を構成するための座標の等分割数である。このグラフの横軸はデータセットのサイズ (ベクトルの数) であり，縦軸は計算の実行時間である。提案手法は通常の手法より高速であることがわかる。

図 4 は，実データに対する検索の計算コストの比較である。この実データのベクトルの次元数は $D = 64$ となっている。提案手法では，高次元空間を $B = 256$ によって等分割して構成した関数索引を用いている。この実験においても，提案手法は従来の手法と比較して数倍程度高速であるとの結果となった。

以上のように，合成データと実データのいずれに対しても，提案手法は従来手法よりも計算効率がよいことが確認された。

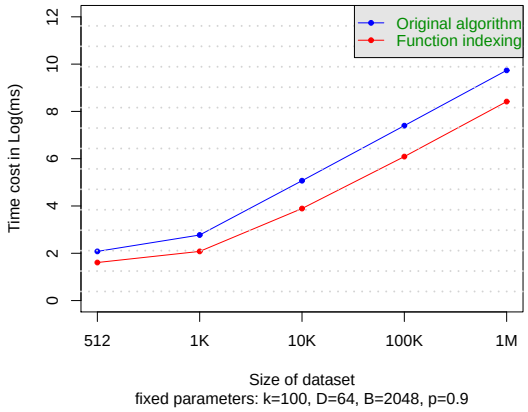

図 3　合成データに関する計算コストの比較

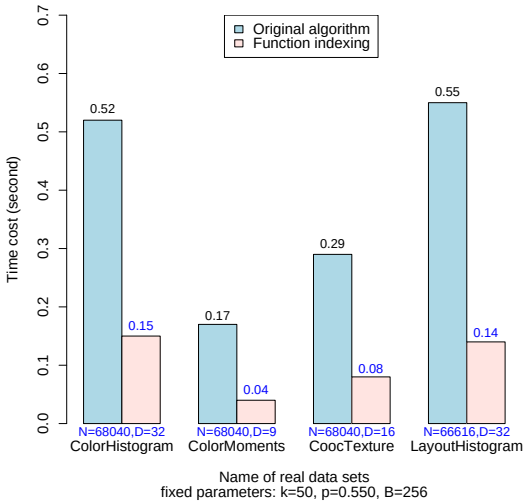

図 4　実データに関する計算コストの比較

(注1)：http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html

6. ま と め

本論文では，k-NN 検索を効率的に行うための関数索引手法を提案した。この手法はあらかじめ計算した値を索引として利用することによって k-NN 検索の際に必要となる冪乗処理の回数を大幅に減らすものであり，これによって検索効率の向上を実現している。提案手法の有効性については，合成データと実データの 2 種類データでの評価実験によって確認した。L_p 距離に基づく高次元データにおける k-NN 検索の計算コストの比較により，提案手法は従来の手法より顕著に速くなることを示した。

今後の課題として，本提案手法を更に汎用的なものとするため，I/O コストの考慮も含めた手法の改良や拡張などを行うとともに，より詳細な評価実験の実施を行うことを検討している。

文　献

[1] C. Aggarwal, A. Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th Int. Conf. on Database Theory*, pages 420–434, 2001.

[2] J. An, H. Chen, K. Furuse, and N. Ohbo. Cva-file: An index structure for high-dimensional datasets. *the Knowledge and Information Systems Journal*, 7(3):337–357, 2005.

[3] S. Berchtold, C. Böhm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *ACM PODS Symposium on Principles of Database Systems*, pages 78–86, 1997.

[4] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is “nearest neighbor” meaningful. In *Proceedings of the 7th Int. Conf. on Database Theory*, pages 217–235, 1999.

[5] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[6] H. Chen, J. An, K. Furuse, and N. Ohbo. C^{2}VA:trim high dimensional indexes. In *Proc. WAIM2002*, pages 303–315, 2002.

[7] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 202–209, 2000.

[8] A. Hinneburg, D. Agrawal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th VLDB Conference*, pages 506–515, 2000.

[9] R. Weber, H. J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 194–205, 1998.

[10] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary L_p norms. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 385–394, 2000.