IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4547

K-NN Search in Non-Clustered Case Using K-P-tree

Yang Zhi-rong and Li Lei
Software Institute, Sun Yat-Sen University, Guangzhou, 510275, China
Tel: 8620-83836474, or 8620-84110658-100, Fax: 8620-84037785
roz@163.net

ABSTRACT

Although it has been shown that all current indexing techniques degrade to linear search for sufficiently high dimensions, exact answers are still essential for many applications. A concept of "non-clustered" case is proposed in this paper. And aiming at the characteristics of such case, a new index structure named K-P-tree and a K-NN searching algorithm based on it is presented. By starting with the Self-Organized Mapping method, the partition procedure of K-P-tree does not depend on the dimensionality. Furthermore, the partition locating and pruning also benefit from the fact that only the inner borders between the partitions are recorded merely via some hyper planes. The query experiments indicate that K-P-tree outperforms many current k-NN searching approaches.

1. INTRODUCTION

In many multimedia applications, the multimedia objects are usually mapped to feature vectors in high-dimensional spaces and queries are made on those features vectors. In these cases, one of the most frequently used yet expensive operations is to find k-nearest neighbors to a given query object.

Two surveys in [WSB98, GG98] provide background and analysis on the index structures. These index structures are divided into two major groups. One group is the space-partitioning methods like k-d tree [Ben75, FBF77], K-D-B tree [Bob81] or hB-tree [LS90] dividing the data space along predefined hyper planes in different dimensions on respective levels. The major problem of space-partitioning methods is that their consumption in memory is exponential to the dimensionality and many of the sub partitions are empty and gratuitous. The other group is the data-partitioning methods such as R*-tree [BKS90], X-tree [BKK96], M-tree [PMP97], SR-tree [KS97], A-tree [SYU00] and iDistance [CBK01] using some simple shapes like hyper rectangles or hyper spheres to bound the pattern clusters. Unfortunately such methods have to confront the complicated problem of handling overlaps and indexing the edges.

Some data clustering approaches, such as LVQ [Koh01], CURE [GRS98], and MACT/SACT [QQZ01] have been introduced into this area. But they mainly aim at solving the data mining problem and identify the clusters, while the retrieval acceleration is usually under minor consideration.

In [WSB98], with several observations and analyses, it shows that all current indexing techniques degrade to linear search for sufficiently high dimensions. Hence in [WSB98] and [GIM99], they resort to the solution without exact answers, and some techniques are employed for approximated k-NN search.

However, since index is solely an auxiliary structure to accelerate the information retrieval, it is somewhat irrational to confine the host applications to the error-tolerant. Although the optimizing problem of similarity search is invincible in theoretically high dimensionality, exact answers are still essential for many applications.

The research in this paper is to advance the solution of k-NN searching problem in a distribution case where the patterns cannot be grouped by distinct borders.

2. NON-CLUSTERED CASE

Most current approaches attempt to develop a universally applicable system and they overlook the peculiarity of the pattern distribution. Actually, to differentiate the density level is very important in this process.

A critical concept to describe distribution density is *cluster* which usually refers to a subset of the patterns, where the distances between its members are far less than those between its members and other patterns.

If the patterns are distributed densely in some "natural" clusters with distinct borders, the similarity searching problem is well solved.

(1) If the distances between clusters are far larger than the cluster sizes. The patterns are densely distributed in some clusters, and the sizes and the shapes of those clusters are neglectable during the search process. It is enough to represent the clusters only with their centers.

There are many approaches to find the centers, e.g. Self-Organized Map [Koh01] proposed by Kohonen. And after fast centers finding, one can locate any of these clusters only by their centers. Such simple representation and locating method are enough and efficient for this case.

Even this assumption does not hold if in some local area, the strategies in [PMP97, YL02] have been propsed to stipulate that the clusters locating is still strictly accurate.

(2) If the distances between clusters are close to the cluster sizes, the cluster sizes and shapes are not neglectable. The data clustering techniques in [GRS98, QQZ01, Koh01] must be employed to recognize the clusters and to delineate the cluster borders. And the problem of indexing each cluster recursively reduces to original one at a smaller scale.

Nevertheless, if the distances between clusters are far less than the cluster sizes, it is probable that there are no "distinct" borders between the "clusters". This case is very prevalent, e.g. the uniform distribution data. What is more, even if the original data can be grouped into clusters, it is still probable that there is no distinct border within individual groups. In fact, under the premise of uniform distribution, it has been shown that most of the current index structures and similarity search algorithms, including the latest ones like A-tree and iDistance, deteriorate dramatically when dimensionality increases.

Hence it leads us to focus on the last case of distribution. Before presenting the definition of "non-clustered", we shall introduce some related notations.

Given a set of patterns denoted *Patterns* and a division of denoted π. π is a class of pattern sub sets and each member of π is called a :

Definition 2.1 , π={C_i, i= 1,2,..., k} where $Patterns = \bigcup_{i=1}^{k} C_i$ and $C_i \cap C_j = \varnothing$. Suppose the center of a class is denoted by $\overline{C}_i$, then we use the cluster radius to describe the size of C_i, which is defined as $r_i = \max\{d(c_i, x) \mid x \in C_i\}$.

And the minimum distance between the respective members from two classes and ($i \neq j$) is used to describe their cluster distance:

Definition 2.2 $d(C_i, C_j) = \min\{d(x,y) \mid x \in C_i, y \in C_j\}$

Definition 2.3 Given a threshold δ (e.g. δ=10 or δ=100), two classes C_i and C_j ($i \neq j$) are δ-*mergable iff* $d(C_i, C_j) > \delta \times \max\{r_i, r_j\}$.

For convenience, we will use the term *mergable* short for δ-mergable unless otherwise stated.

When the *mergable* predicative holds for C_i and C_j, it means C_i and C_j are so close in some direction(s) that they can be merged into one cluster.

For convenient expression, we define the following boolean function for the *mergable* predicative: $CM(C_i, C_j) = 1$ if C_i and C_j are mergable; Otherwise $CM(C_i, C_j) = 0$. It is obvious that $CM(C_i, C_j) = CM(C_j, C_i)$

Definition 2.4 $Degree(C_i) = \sum_{\substack{j=1 \\ j \neq i}}^{k} CM(C_i, C_j)$

The *degree* function of a class returns the connectivity between other classes.

Definition 2.5 A division $\pi = \{C_i, i = 1,2,...,k\}$ is $\theta - non - clustered$ *iff* $E(\pi)/k > \theta$.

Here *E(.)* is the mathematical expectation and θ is a given threshold, say, θ=0.5 or θ=0.9. Likewise, in contexts without confusion, we use the term *non-clustered* short for θ–*non-clustered* .

Definition 2.6 A pattern set *Patterns* is *non-clustered iff* all divisions of *Patterns* are *non-clustered*.

When the *non-clustered* predicative holds for a pattern set *Patterns*, it means by any way we divide *Patterns*, each class can always be merged with most of the other classes. The whole pattern set behaves in integrity and there is no way to find some "natural" borders to separate its members. The problem can be simplified as dividing a high-dimensional sub space with a special shape, as well as locating and pruning the partitions.

In this paper, a new approach is proposed to solve the problem in the last case. By starting with the SOM method, our partitioning method does not depend on the dimensionality. Furthermore, the partition locating and pruning also benefit from that only the inner borders between the partitions are recorded merely via some hyper planes.

3. DATA STRUCTURE

In order to construct a balanced index structure, before dividing the pattern set, we should get its compact form.

Definition 3.1 Suppose $X \in R^n$ is the distribution of a stochastic variable *x*, and $C_X \in R^n$ is the distribution of another stochastic variable c_x. C_X is the set of reference vectors of *X iff* $p(x) = p(c_x)$. Here $p(\cdot)$ is the probability density function.

There are many methods to get the reference vectors, one of which is the Self-Organizing Map (SOM) [Koh01]. With proper configuration, the output of SOM approximates the original distribution with a much smaller set of reference vectors. And each reference vector is the centroid (mean value) of its corresponding pattern set.

In our approach, some sub spaces are defined to enclose each pattern sub set and the inner borders between these sub sets.

Definition 3.2 The partition defined on a given set of hyper planes *B* is a subspace of R^n, denoted *P(B)*, where $P(B) = \{x \mid x \in R^n \wedge x \cdot q - \alpha > 0$, for all $b = H(q,\alpha) \in B\}$.

Here *B* is called the border of *P*, each $b \in B$ is called a border segment of *P(B)*.

Definition 3.3 *P(B)* is the enclosed partition of vector set . (In the pseudo code listed below, without confusion, the term "enclosed" will be omitted.)

Definition 3.4 $P(B_1)$ is called child partition of $P(B_0)$ and $P(B_0)$ is called parent partition of $P(B_1)$. It is easy to see that $P(B_1) \subseteq P(B_0)$.

A child partition inherits all border segments from its parent. And usually with some additional ones, the child partition represents a smaller region.

For neighborhood preservation, it is advantageous to designate each pattern to its nearest reference vector. Under such premise, we can easily compute the enclosed partitions with the reference vectors. Because it is not difficult to prove that the border segments between two pattern sub sets are a pair of hyper planes with negative directions to each other, which locate in the middle of the respective reference vectors. And the pair of hyper planes is orthogonal to the line through the two reference vectors.

Given two pattern sub sets X_i and X_j (i≠j), whose reference vectors are c_i and c_j, and whose enclosed partitions are $P(B_i)$ and $P(B_j)$, respectively. The border segments between X_i and X_j are $b = H(q,\alpha)$ and $b' = H(q',\alpha')$, $l = 1,...,k$, where

$-q = q' = \frac{c_j - c_i}{\| c_j - c_i \|}$, and $-\alpha = \alpha' = \frac{c_i + c_j}{2} \cdot q' = \frac{\| c_j \|^2 - \| c_i \|^2}{2 \| c_j - c_i \|}$.

Obviously it is profitable to compute only one border segment in each pair and to store them together in the upper level. In each node of K-P-tree, a bitwise mask indicates the valid border segments for the corresponding partition. Likewise, another bitwise string indicates the directions for those valid border segments.

As shown in Figure 4.1, a node of the K-P-tree has the following structure:

TPartition : (*border_mask*, *border_directions*, *segments*, *patterns*, *children*), where

border_mask and *border_directions* are BITWISE elements mentioned above.

segments refers to the border segments in the next level, which is an array of hyper planes with size of NUM_SEGMENTS, where

NUM _SEGMENTS := $\frac{k*(k-1)}{2}$; //*k* is the partition count in each level

As we know, if $q \in R^n - \{0\}$, $\alpha \in R^1$, then the (n-1)-dimension set $H(q,\alpha) = \{x \in R^n \mid x \cdot q = \alpha\}$ defines a hyper plane in R_n. Hence we use the pair (q, α) to represent a hyper plane.

patterns is the entry for the set of vectors within the enclosed partition defined by *segments*. It is a null pointer for intermediate nodes,

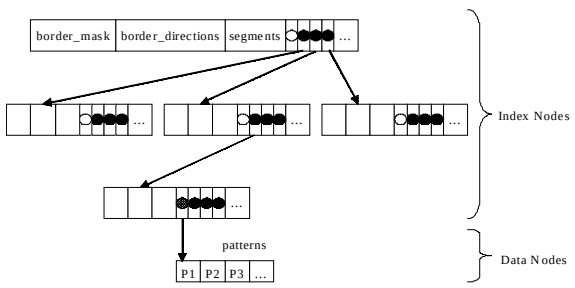
children are the entries to the child nodes.

The index construction for K-P-tree is simple: (1) Get the reference vectors by means like SOM; (2) Divide the pattern set by designating each pattern to it nearest reference vector; (3) Compute the border segments, afterwards log the indicator of validity and directions (border_mask and border_directions); (4) Recursively apply the procedure in steps (1)-(3) on each sub pattern set until the size of which is less than a given leaf threshold.

4. SEARCH ALGORITHM

The search algorithm comprises a main procedure and two subroutines to locate a leaf partition and to judge the intersection.

Figure 4.1 The K-P-tree structure



The first subroutine follows the definition of half-space: First compute the location of input pattern, which is stored in the bits of location_code. Afterwards, by masking the invalid bits and comparing with the border segment directions indicator, only one child partition is located. Then by recursively applying LocatePartition procedure, we get the process in pseudo C codes to locate the leaf partition which the input pattern belongs to.

```
#define Positive( x , h ) ( x · h.q – h.α > 0 ) ? 1 : 0;
function LocatePartition(VECTOR input , TPartition parent ) : TPartition
{
    BITWISE location_code;
    for(i=0;i<NUM_SEGMENTS;i++)
        location_code[i] = Positive( input , parent .segments[i]);
    for each child of parent do
        if (location_code & child.border_mask == child.border_directions)
            return child;
}

function LocateLeaf(VECTOR input , TPartition root ) : TPartition
{
    TPartition child := root ;
    while( child is not a leaf)
        child := LocatePartition( input , child );
    return child ;
}
```

The second subroutine, *Intersectant*, and the main procedure are list below.

```
function Intersectant (VECTOR input , REAL r̂min k max , TPartition child ) : BOOL
{
    for j:=1 to NUM_ SEGMENTS do
        if child .border_mask[j] then
        {
            y := x − ((x · q − α) · q) / ‖q‖² ;
            if d(input, y) < r̂min k max then
                return true;
        }
    return false;
}

function k_NN_Search(VECTOR input , int k , TPartition root ) : Pattern_Set
{
    leaf := LocateLeaf( input , root );
    result0:= k_NN_Search0( input , k , leaf .patterns );
    r̂min k max := max{ d(input, y) }, where y ∈ result0;
    Queue := { root }; Possible := ∅ ;
    while( Queue ≠ ∅ )
    {
        node :=pop( Queue );
        if node is not a leaf then
            for each child of node do
                if Intersectant( input , r̂min k max , child ) then
                    Append child to Queue ;
        else
            Possible := Possible ∪ node.patterns ;
    }
    return k_NN_Search0( input , k , Possible );
}
```

The main function, k_NN_Search, consists of three steps: (1) an exhaustive search procedure, k_NN_Search0, is employed on the leaf partition which the input pattern belongs to; (2) a pruning process is employed to find all possible partitions partitions with $\hat{r}_{\min k \max}$; (3) the exhaustive search will employed again on the possible pattern set.

The partition locating in (1) has been presented in the preceding section and the exhaustive search in (1) and (3) is simple. To determine whether a given partition contains one or more k-nearest neighbors in step (2), we make use of the property of intersection.

Consider $r_{\min k \max} = \max\{d(input, y) \mid y \in KNN\}$, where KNN is the set of k-nearest neighbors. We denote $S(input, r_{\min k \max}) = \{x \mid d(input, x) < r_{\min k \max}\}$. It is easy to see that $S(input, r_{\min k \max})$ is a hyper sphere and $KNN \subseteq S(input, r_{\min k \max})$. Hence it is easy to see that given an enclosed partition $P(B) \neq leaf$ and its pattern set $PS \subseteq P(B)$, there will be $PS \cap KNN \neq \varnothing$ *iff* $\exists b \in B$, *b* intersects the hyper sphere $S(input, r_{\min k \max})$.

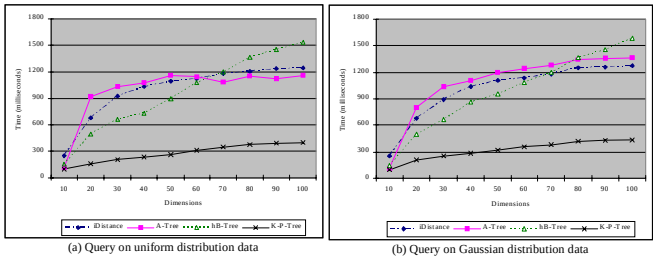
During the pruning process, since $\hat{r}_{\min k \max} \geq r_{\min k \max}$, if *b* does not intersect the hyper sphere $S(input, \hat{r}_{\min k \max})$ for any $b \in B$, then the partition *P(B)* and its pattern set *PS* can be safely pruned.

A static pruning is applied in the main procedure as presented above, where $S(input, \hat{r}_{\min k \max})$ is invariable during the pruning process. Some dynamic strategies can be employed. For example, when a leaf *node* is popped from *Queue*, the exhaustive search, k_NN_Search0, is applied to $Possible \cup node.patterns$ to get more accurate $\hat{r}_{\min k \max}$ for further pruning. Moreover, some heuristic information, say $d(input, c_i)$, can be used as the key to sort the *Queue* in addition to dynamic adjusting $\hat{r}_{\min k \max}$. With these dynamic strategies, the pruning effect can be enhanced so that step (3) benefits, though the step (2) suffers from the cost of additional operations.

5. QUERY EXPERIMENTS

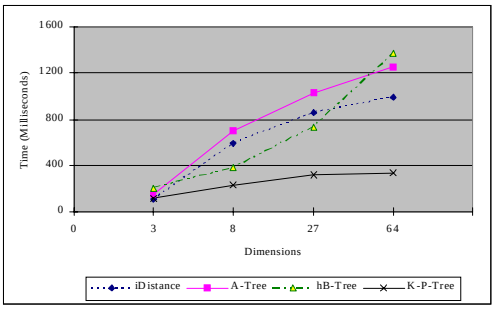
The purpose of the experiments is to test the query performance of our method and several current multi-dimensional index structures with varying dimensionality. The selected approaches in comparison were iDistance [CBK01] on behalf of data partitioning with hyper spheres, A-tree [SYU00] on behalf of data partitioning with rectangles and hB-tree [LS90] on behalf of space partitioning.

Figure 5.1 Performance of query on synthetic data with varying dimensionality



(a) Query on uniform distribution data (b) Query on Gaussian distribution data

Figure 5.2 Performance of query on real data with varying dimensionality



All the experiments were performed on a PC with AMD Athlon 850 CPU, 512MB RAM and several GB hard disk space. To avoid the interference of I/O cost, we configured all the index structures and all vector data stored in the main memory.

We evaluated the structures on synthetic databases of two typical kinds of non-clustered data sets by searching the nearest neighbor of a randomly generated vector. The first was uniform distribution and the second was Gaussian distribution with some randomly chosen peaks. The size for all data sets was 10,000, and the dimensionality ranged from 5 to 100 with step size of 5. For K-P-tree, we used k=6 for each level. For convenient observation, the query process was repeated by 1,000 times and the query time consumed is shown in Figure 5.1(a) and 5.1(b).

In our experiment, we also used real data, which are color histogram vectors extracted from a clipart database with size of 7,600. Since there was no predefined relationship between these selected pictures, they can be deemed as non-clustered data. The dimensions selected in this experiment are 3, 8, 27, and 64. The input vector was also extracted from a picture which is randomly selected from a picture set SS. We configured half pictures were contained in DB while the other half was not. Likewise, the query was also repeated by 1,000 times and the query time consumed is shown in Figure 5.2.

6. CONCLUSION

In this paper we have pointed out the importance of discriminating the distribution density, based on which we proposed the concept of "non-clustered case". Aiming at the characteristics of such distribution we have presented a hierarchical index structure named K-P-tree and a k-NN searching algorithm based on it. By partitioning the patterns based on SOM and only storing the inner borders via the simplest shape, hyper planes, our method achieve very good searching efficiency. The query experiments indicate that K-P-tree outperforms many current k-NN searching approaches such as iDistance, A-tree and hB-tree. Furthermore, the structure of K-P-tree is so flexible that it can be applied to various applications.

REFERENCE

[Ben75] Jon Louis Bentley: 'Multidimensional Binary Search Trees Used for Associative Searching', ACM Vol.18, No.9, 1975, pp.509-517.

[BKK96] Berchtold, S., Keim, D., Keriegel, H.-P: 'The X-tree: An index structure for high-dimensional data', Proceedings of the 22nd International Conference on Very Large Data Bases, (Bombay), 1996, pp.28-39.

[BKS90] Beckmann N., Kriegel H.-P, Schneider R., Seeger B.: 'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles', Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[CBK01] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, H. V. Jagadish: 'Indexing the Distance: An Efficient Method to KNN Processing', Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, Raphael Ari Finkel: 'An Algorithm for Finding Best Matches in Logarithmic Expected Time', ACM Transactions on Mathematical Software, Vol.3, No. 3, September 1977, pp.209-226.

[GG98] Volker Gaede, Oliver Günther: 'Multidimensional Access Methods', ACM Computing Surveys, Vol.30, No.2, 1998, pp.170-231

[GIM99] Aristides Gionis, Piotr Indyk, Rajeev Motwani: 'Similarity Search in High Dimensions via Hashing', Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[GRS98] Sudipto Guha, Rajeev Rastogi, Kyuseok Shim: 'CURE: An Efficient Clustering Algorithm for Large Databases', Proceedings of ACM SIGMOD'98 Conference, 1998.

[Gut84] Guttman, A: 'R-trees: A dynamic index structure for spatial searching', Proceedings of the ACM SIGMOD International Conference on Management of Data, 1984, pp.47-54.

[Koh01] Kohonen T.: 'Self-Organizing Maps', Third Edition, Springer, 2001

[KS97] N. Katayama and S. Satoh: 'The SR-tree: an Index Structure for High-Dimensional Nearest Neighbor Queries', Proceedings of the 1997 ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 369-380.

[LJF94] Lin, K.-I., Jagadish, H., Faloutsos, C.: 'The TV-tree: An index structure for high-dimensional data'. VLDB J. Vol.3, No.4, 1994, pp.517-543.

[LS90] David 9. Lomet, Betty Salzberg: 'The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance', ACM Transactions on Database Systems, Vol.15, No.4, 1990, pp.625-658.

[PMP97] Paolo Ciaccia, Marco Patella, Pavel Zezula: 'M-tree: An Efficient Access Method for Similarity Search in Metric Spaces', Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.

[QQZ01] Qian Wei Ning, Qian Hai Lei, Zhou Ao Ying: 'Merging Cluster-Trees: Clustering Very Large Databases', Proceedings of The 18th National Conference on Data Bases, China, 2001, pp.128-133.

[Rob81] Robinson, J. T.: 'The K-D-B-tree: A search structure for large multidimensional dynamic indexes', Proceedings of the ACM SIGMOD International Conference on Management of Data, 1981, pp.10-18.

[SYU00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. 'A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation', Proceedings of the 26th VLDB Conference, 2000, pp.516-526.

[WSB98] Roger Weber, Hans -J. Schek, Stephen Blott: 'A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces', Proceedings of the 24th VLDB Conference, New York, USA, 1998.

[YL02] Yang Zhi Rong, Li Lei: 'Hierarchical Index of High-Dimension Point Data Using Self-Organizing Map', Computer Research and Development, Vol. 39, 2002