World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:1, 2008

Non-Overlapping Hierarchical Index Structure for Similarity Search

Mounira Taileb, Sid Lamrous, and Sami Touati

***Abstract*—**In order to accelerate the similarity search in high-dimensional database, we propose a new hierarchical indexing method. It is composed of offline and online phases. Our contribution concerns both phases. In the offline phase, after gathering the whole of the data in clusters and constructing a hierarchical index, the main originality of our contribution consists to develop a method to construct bounding forms of clusters to avoid overlapping. For the online phase, our idea improves considerably performances of similarity search. However, for this second phase, we have also developed an adapted search algorithm.

Our method baptized NOHIS (Non-Overlapping Hierarchical Index Structure) use the Principal Direction Divisive Partitioning (PDDP) as algorithm of clustering. The principle of the PDDP is to divide data recursively into two sub-clusters; division is done by using the hyper-plane orthogonal to the principal direction derived from the covariance matrix and passing through the centroid of the cluster to divide. Data of each two sub-clusters obtained are including by a minimum bounding rectangle (MBR). The two MBRs are directed according to the principal direction. Consequently, the non-overlapping between the two forms is assured.

Experiments use databases containing image descriptors. Results show that the proposed method outperforms sequential scan and SR-tree in processing k-nearest neighbors.

***Keywords*—**K-Nearest Neighbor Search, Multidimensional Indexing, Multimedia Databases, Similarity Search.

I. Introduction

A content-based image retrieval system (CBIR) offers the possibility to manage totally a large images collection. Indeed, it must be able to make an update, to describe the images automatically, and must also allow an example search based on the visual similarity, in other words, to find for an image given in an example images considered similar. To implement such a system, two fields, to which the system appeals, are made complementary. They are the image processing (automatic description of the images) and the data bases.

Image processing is the automatic description of the images; it consists of extracting from an image its visual properties (form, color, texture…). These properties are represented as multidimensional vectors called *descriptors* [1]. To find the images similar to an image query, a similarity search (example: nearest neighbors) is made for each descriptor of the image query. Considering that several similarity searches are carried out: as many searches as descriptors characterizing the image query. Using data structure to index the descriptors base proves to be essential. Several index structures were proposed, and the high-dimensional index structures are adapted to descriptors with large dimension. The objective of proposed high-dimensional indexes is to structure descriptors data base with an aim of accelerating the search process.

Obtaining a high-dimensional index can be made by using traditional techniques of indexing such as R-tree [2], or by using a clustering algorithm to form clusters or groups of descriptors, and the clusters are supported by a hierarchical structure, as an example BIRCH use CF-tree [3], DBSCAN use R*-tree [4] and X-tree [5]. Many high-dimensional index structures have been proposed, the most known and used are data-partitioning based index structure such as SS-tree [6], SR-tree [7], X-tree, considered as extensions of R-tree, and space-partitioning based index structure such as k-d-B-tree [8], hB-tree [9], and LSDh-tree [10] derived from kd-tree [11]. The R-tree-based index structures suffer from overlapping between bounding regions and the low fanouts, these influence negatively on the results of query processing. The kd-tree-based index structures drawbacks are essentially the no guarantee of using allocated space; this led to the consultation of few populated or empty clusters.

Taking into account drawbacks cited above, and with an aim to accelerate nearest neighbors search, we propose a new high-dimensional index technique called NOHIS. It is composed of two phases:

- The first offline phase consists in gathering descriptors in clusters; the clustering algorithm used is the Principal Direction Divisive Partitioning (PDDP) [12]. It's one of the divisive hierarchical clustering algorithms; it divides data recursively into two sub-clusters by using the hyper-plane orthogonal to the principal direction derived from the covariance matrix and passing through the centroid of the cluster to divide. A binary not balanced tree is obtained at the end of the clustering process. Our contribution consists in using minimum bounding rectangle (MBR) avoiding overlap, MBRs are directed according to the principal direction (principal component) used in the clustering algorithm to divide a cluster into two sub-clusters. We call NOHIS-tree, the tree obtained by using PDDP, in which we use non-overlapping rectangles.

Manuscript received December 31, 2007.

M. Taileb is with the Department of Computer Science, University of Paris-Sud, Orsay 91405 France (e-mail: mounira.taileb@u-psud.fr).

S. Lamrous, is with SeT Laboratory, University of Technology of Belfort Montbeliard, Belfort, 90010 France (e-mail: sid.lamrous@utbm.fr).

S. Touati was with DRT Laboratory in the CEA/SACLAY France. He is now with the Company Aviasys R&D, Evry, 91000 France (e-mail: sami.touati@aviasys.com).

- The second online phase is the step during witch the interrogation of the obtained index is made by a nearest neighbors search carried out on the NOHIS-tree. We propose a search algorithm adapted to the used MBRs.

The rest of the paper is organized as follows: in section II we present our proposed hierarchical indexing method by detailing its two phases. Section III presents experiments when comparing our proposed method with sequential scan, PDDP-tree and SR-tree. We have chosed SR-tree for several reasons, it is a multidimensional index proposed recently which still attracts the attention [13], and for the availability of its source code. Finally, section IV concludes the paper.

II. The NOHIS Methods

The proposed high-dimensional index can be divided in two groups according to the partitioning strategy, the data-partitioning and the space-partitioning based index structure. When the nearest neighbors search is applied on a data-partitioning index, additional clusters are visited due to the overlapping between the MBRs. In the case of the space-partitioning; consultation of few populated or empty clusters is extremely probable.

By using NOHIS, the overlapping is avoided and the quality of clusters is preserved. This can be explained by the following facts:

1- The clustering algorithm forms clusters by using data dispersion, by guaranteeing the possibility to avoid empty and few populated clusters by fixing a minimal threshold for the cluster size (number of vectors contained in the node).

2- The direction of the two MBRs according to the principal direction ensures that there is no overlapping between them.

Before detailing the suggested method, we indicate by data the notaly of multidimensional descriptors.

A. Offline phase

The phase offline of the suggested method can be represented in three principal stages:

1 - Data constituting the initial cluster is divided into two sub-clusters using the hierarchical clustering algorithm PDDP [12]. Division is made by the hyper-plane orthogonal to the first principal component passing through the centroid of the cluster to divide. The principal component corresponds to the first principal direction carried by the first eigenvector of the matrix COV given by (1) associated to its largest eigenvalue.

M(n x m) represents the matrix of data to be clustered, m is the size of data, n their dimension and w the centroid of data given by (2):

$$COV = (M - we^T)(M - we^T)^T \quad (1)$$

$$w = \frac{v_1 + v_2 + \cdots + v_m}{m} = M.e.\frac{1}{m} \quad (2)$$

$$e = (1,1,\ldots,1)^T$$

The most important dispersion of the data is according to this component, so dividing accordingly it allows to have dense clusters.

2- Data of each obtained sub-clusters is gathered by bounding form. The bounding forms of both clusters do not overlap because overlapping degrades considerably the performances of the similar search.

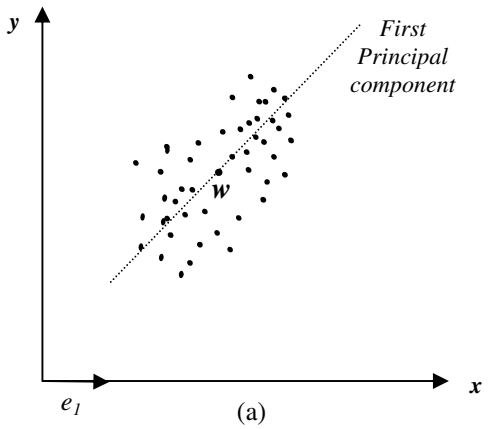
3- Hyper-rectangles are the bounding forms used; they are directed according to the first principal component considered. The direction of the bounding forms according to the first principal component ensures the non-overlapping between the two forms.

A.1. Data Partition

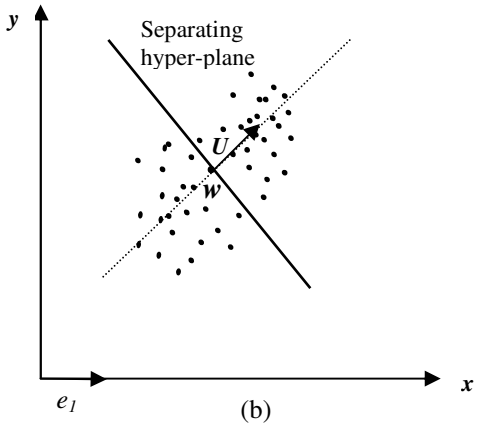
Figure 1 illustrates the example of data partition, in 2D, into two clusters. The whole of the vectors of data constitutes the matrix *M* (figure 1.a). Let us consider the matrix of covariance of *M*, note by *U* the first principal component. Data is divides into two parts which are included in two rectangles having as axis the line engendered by *U*. the two rectangles should not overlap. Division is made with the separating plan (hyper-plane starting from 3D) passing by the center of data *w* and perpendicular to the first principal component (figure 1.b). Data is divided recursively into two parts P_R and P_L (R for right and L left) according to the following rule:

$$g(x_i) = U^T(x_i - w) \geq 0 \quad \Rightarrow \quad x_i \in P_R$$
$$g(x_i) = U^T(x_i - w) < 0 \quad \Rightarrow \quad x_i \in P_L$$

x_i is a multidimensional vector from the considered data.



(a)



(b)

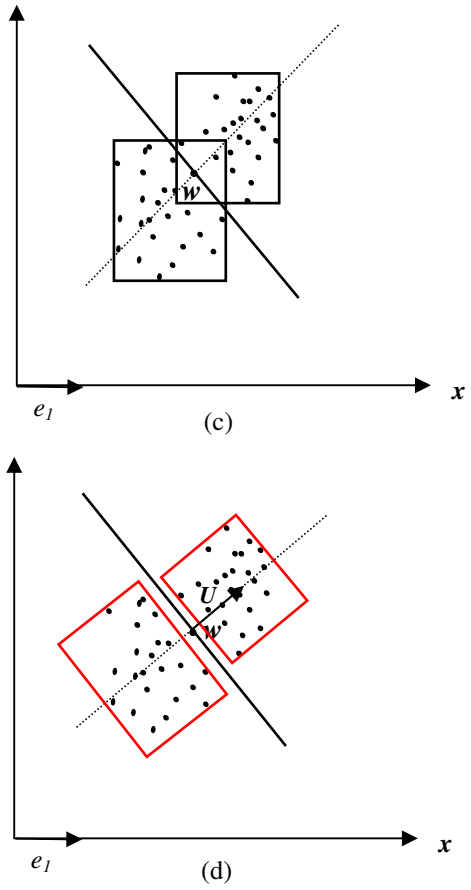


Fig. 1 Example, in 2D, of data clustering and the use of the Minimum Bounding Rectangles in direction of the first principal component

Let us note M_R and M_L the two corresponding matrices of he two parts. Clustering algorithm [14] is given in figure 2 below.

0. **Start** with the matrix of vectors *M(n x m)*, and a desired number of clusters c_{max}.
1. **Initialize** Binary tree with a single Root node
2. **For** c = 2,3,…., c_{max} **do**
3. **Select** node C with largest Scatter Value
4. **Create** L & R : = left & right children of C
5. **For** i = 1 to C_{size}
   Compute $g(x_i)$, if $g(x_i) \leq 0$ assign x_i to L
   else assign it to R
6. **Result:** A binary tree with c_{max} leaf nodes forming a partitioning of the entire data set

Fig. 2 PDDP Algorithm

A.2. The change of reference mark

The figure 1.c represents the case of use of MBRs in the origin reference mark; in which the coordinates of the vectors are expressed. It is clear that having MBRs in this way creates an overlapping, and consequently, in a nearest neighbors search, additional clusters will be consulted without improving the results. As solution, to avoid the overlapping, we propose to direct MBRs according to the first principal component (figure 1.d). In this case, a change of reference mark is essential.

Let $B=\{e_1,e_2,\ldots,e_n\}$ be the canonical base of R^n, $e_1=(1,0,0\ldots,0)$, $e_2=(0,1,0\ldots,0)$, $e_2=(0,0,1\ldots,0)$, ….

The goal is to build an orthonormal base $B'=\{u_1,u_2,\ldots,u_n\}$ where a vector is equal to U (e.g. $u_1 = U$), a such base can be obtained by transforming B by an orthogonal isomorphism, for example by an orthogonal symmetry S. We must have $B'=(S(e_1),S(e_2),\ldots,S(e_n))$ and in particular $S(e_1)= u_1=U$.

$$\text{let} \quad V = \frac{(U - e_1)}{\| U - e_1 \|} \tag{3}$$

and H be the hyper-plane orthogonal to V, $H=V^{\perp}$, so that H be the mediator hyper-plane of e_1 and U.
We define S as the orthogonal symmetry with respect to H.

The image of the vector x by S is: $S(x) = x - 2\langle x,V\rangle.V$, where $\langle x,V\rangle$ is the scalar product of x and V.

In particular, we have: $u_i = e_i - 2\langle e_i,V\rangle.V \qquad 1\leq i\leq n$

With this definition of S when has, in fact, $u_1=S(u_1)= U$.

In fact: $u_1 = e_1 - 2\alpha\langle e_1,U-e_1\rangle.(U-e_1)$

With: $$\alpha = \frac{1}{\|U - e_1\|^2} = \frac{1}{\|U\|^2 + \|e_1\|^2 - 2\langle e_1,U\rangle}$$

$$\|U\|^2 = \|e\|^2 = 1 \Rightarrow \alpha = \frac{1}{2(1 - \langle e_1,U\rangle)}$$

$$u_1 = e_1 - \frac{2}{2(1 - \langle e_1,U\rangle)}(\langle e_1,U\rangle - 1)(U - e_1)$$

$$u_1 = e_1 + \frac{1}{(\langle e_1,U\rangle - 1)}(\langle e_1,U\rangle - 1)(U - e_1)$$

$$u_1 = e_1 + (U-e_1) = U$$

A.3. MBR's construction

Let be N_R(resp. N_L) the matrix containing vectors of P_R (resp. P_L) in the base B'.

We have: $N_R = M_R - 2.V^T.V.\ M_R = (I - 2V^T.V).\ M_R$ (4)

$N_L = M_L - 2.V^T.V.\ M_L = (I - 2V^T.V).\ M_L$ (5)

Vectors of N_R (resp. N_L) are included in a MBR R_R (resp. R_L). A property of the MBR is that each of his face passes by a vector at least. MBRs are characterized by vectors S and T, where:

$S_R = \min(N_R)$ (resp. $S_L = \min(N_L)$) and
$T_R = \max(N_R)$ (resp. $T_L = \max(N_L)$)

Note that the minimum and the maximum of this formula are taken line by line, so that $S=(s_1,..,s_i,...s_n)$ et $T=(t_1,..,t_i,...t_n)$ where s_i (resp. t_i) is the minimum (resp. the maximum) of the i^{th} component of the considered vectors.

In an internal node (not a leaf) of the NOHIS-tree, following informations are stored: S_R, T_R, S_L, T_L and the common vector V given by (3). A leaf node contains vectors, it is called data node. Leaves represent the obtained clusters.

B. On-line phase

As a result from the off-line phase, a not balanced binary tree is obtained. Let's called *father*, a cluster having two sub-clusters obtaining after division (for example, the nodes N1, N2, N3 in figure 3), and *child*, a sub-cluster. Leaves are the data nodes which contain the vectors. For a query vector *q*, before searching its nearest neighbors, coordinates in the new reference mark (i.e. the new base *B'*) must be calculated, in order to calculate the distance to the MBR. The computing of new coordinates is done in each level in the NOHIS-tree until a leaf node. The passage of the *q* from a father node to its *child* requires the computing of its new coordinates because a change of the reference mark has occurred. Two children of the same father have a common reference mark.

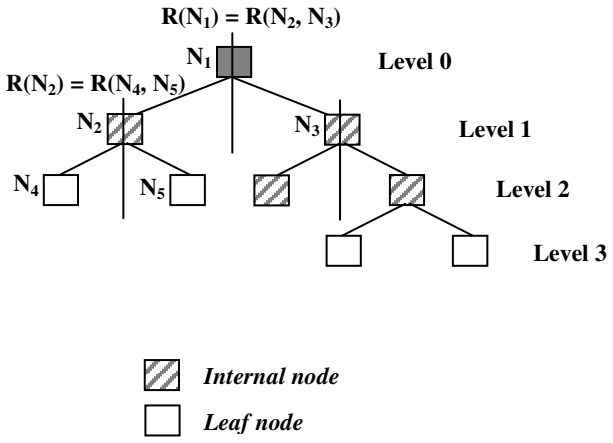


Fig. 3 Example of NOHIS-tree

B' is orthonormal, so coordinates of *q* in *B'* (*q'*) are given by the products scalar:

$$<q,u_i> = <q,e_i> - 2<e_i,V>.<q,V>$$

$$q' = [<q,u_1>,<q,u_2>,\ldots..,<q,u_n>]^T$$

$$q' = q - 2<q,V>V \qquad (6)$$

Distance separating *q* from a rectangle *R* is calculated as given in [15] using *q'*, *S* and *T*. *MINDIST* is the distance between the query vector and an MBR,

$$MINDIST(q',R) = \sum_{i=1}^{n} \left| q'_i - r_i \right|^2 \qquad (7)$$

with: $$r_i = \begin{cases} s_i & if\ q'_i < s_i \\ t_i & if\ q'_i > t_i \\ q'_i & else \end{cases}$$

when q' is in R, *MINDIST*(q',R) = *0*.

B.d. Search algorithm

Nearest neighbor search improves clustering efficiency. In algorithm 1, we present our k-nearest neighbors search (k-NN) adapted to the obtained NOHIS-tree. We note that NOHIS-tree support also a range query search. For a vector query *q*, the k nearest vectors must be returned. *list_Neighbors* (LN) is the table containing k-nearest neighbors. For each nearest vector, LN must contain: its indice in the database, the indice of the cluster to which it belongs, and its distance from *q*. Distances of the returned nearest neighbors are initialized in the infinite value. Returned LN is sorted according to the distances. This algorithm is recursive; the first call is done with the root of the NOHIS-tree and a distance called maxDist initialized at 0. If the node is not a leaf then, first *q'* is calculated by (6) and then distances MINDIST given by (7) are computed between *q'* and the two children's MBRs of the node. A first recursive call in the algorithm 1 can be made with the child node having smallest distance MINDIST, let M[j] be this smallest distance. We attribute to M[j] the maximum between M[j] itself and maxDist. The condition of this recursive call is that M[j] must be lower than the biggest distance contained in LN (LN.dist[k]). A second call can be made with the second child if the same condition is satisfied. Else if the considered node is a leaf, Euclidian distances between *q'* and all the vectors of the node are calculated. Only vectors having a distance lower than LN.dist[k] are inserted by sorting in LN.

Algorithm 1 : *K-NN Search*

```
1. Begin
2.   If the node is a leaf
3.     For i : = 1 to node.size
4.       Compute distance between vectQuestion and
         node.vect[i], let be dist;
5.       if (dist < list_Neighbors.dist[k])
6.         Insertion of current vector in list_Neighbors by
           sorting
7.       end if
8.     end For
9.   Else
10.    For j : = 1 to 2 (the two node's chikdren)
11.      Compute coordinates of vectQuestion in the new
         reference mark, let be vectQuestion'
12.      M[j] = MINDIST (vectQuestion', MBR of node.child j);
13.    end For
14.    Take the child node having the smallest distance M[j];
15.      M[j] = max (maxDist, M[j])
16.    if (M[j] < list_Neighbors.dist[k])
17.      Recursive call of K-NN Search passing the child node
         and M[j] as maxDist
18.    end if
19.      let MS the MINDIST of the second child
20.      MS = max(maxDist, MS)
21.    if (MS <list_Neighbors.dist[k])
22.       Recursive call of K-NN Search with the second child
          and MS as maxDist
23.    end if
24.  end Else
25. End
```

The condition of the recursive call in algorithm (M[j] < LN.dist[k]) is necessary because distances of vectors included in a MBR from a query vector can be only higher or equal to M[j], and as LN is sorted in the ascending order, therefore LN.dist[k] is the biggest distance contained in LN, and consequently if M[j] is not lower than LN.dist[k], the MBR can not contains closer vectors to the query vector that those already found.

In a hierarchical index the bounding forms of a level are contained in that of the inferior level. Taking the example of a node *father*, the bounding forms of its children are contained in its bounding form and consequently, the distance from a query vector to the node *father* is lower or equal to its distances to the children. This gives a property to the hierarchical index that the distance of a query vector q to the bounding forms increases from a level to that highest.

In our index structure NOHIS-tree, bounding rectangles of children (R_1, R_2) are not included completely in the bounding rectangle of their node *father* R, as shown in the figure 3.

The instruction M[j] = max (maxDist, M[j]) in the line 15 of algorithm 1, (resp. MS = max (maxDist, MS) in the line 20), preserve the property that the distance increases from a level to that highest in the search tree. M[j] expresses the distance to their intersection.

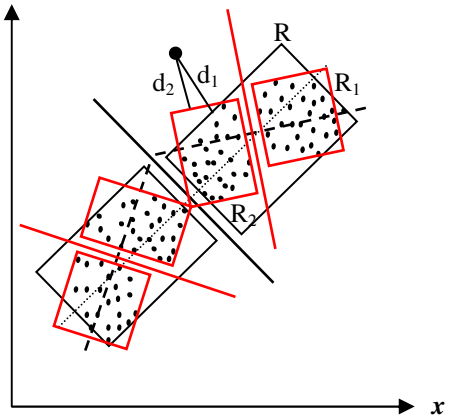


Fig. 4 Example of children's MBRs

III. Experiments

Clustering algorithm and the search algorithm are mplemented in C++. Algorithms run on a PC with Intel processor, its CPU is 1.7 GHz and 512 Mo of RAM. To evaluate performances of similarity search on our proposed index we performed various experiments. We use a database containing 1,193,647 vectors of dimension 30. Vectors are local descriptors based on points of interest derived from 4,996 images. In the first and second experiment, during the search process index and data of considered methods are loaded completely in main memory. In the third experience, the index and data of NOHIS-tree and SR-tree are stored on the disk and loaded in main memory when necessary.

In experiment 1, 6 databases of size 50,000 and different dimensions (25 to 150) are used and three search methods are considered, sequential scan and k-NN search carried out the PDDP-tree and NOHIS-tree. Sequential scan remains competitive in high dimensional spaces. The PDDP-tree is obtained when applying PDDP clustering algorithm and using MBRs oriented according the original reference mark with overlap as shown in Fig.1.c. 200 query vectors are randomly selected from data bases. In this experience and the two following we take for NOHIS-tree and PDDP-tree the good number of clusters which gives the best performance of search. We create for each base multiple partitions that differ by the number of clusters; we take the partition having the number of clusters that minimizes the search time as shown in the column "number of clusters" in tables I, II and III.

The experiment consider the cumulated response time for searching in databases 20 nearest neighbors (NNs) of the 200 query vectors. The goal is to study the impact of the dimensionality on the three search methods. Fig. 5 shows the total search time for three search methods. The search time increase with growing dimension, the NOHIS-tree significantly outperforms the PDDP-tree and sequential scan. In 25-dimensional space, the NOHIS-tree performs the queries 9.91 times faster than the PDDP-tree and 16.785 times faster than the sequential scan. Even in 150-dimension space, the NOHIS-tree performs the queries 4.593 times faster than the PDDP-tree and 6.336 times faster than the sequential scan. NOHIS-tree keeps its performances even in high dimensional space, when performances of other index structure decrease in high dimensional space when comparing to the sequential scan.

TABLE I
EXP. 1 IMPACT OF DATA DIMENSIONALITY ON THE SEARCH PERFORMANCE

Dimension	Number of clusters	Total search time (s)		
		NOHIS-tree	PDDP-tree	Seq. scan
25	600	0,210	2,083	3,525
40	600	1,151	4,977	5,037
80	600	1,261	7,551	8,882
100	700	1,853	9,414	10,866
150	600	2,494	11,456	15,802

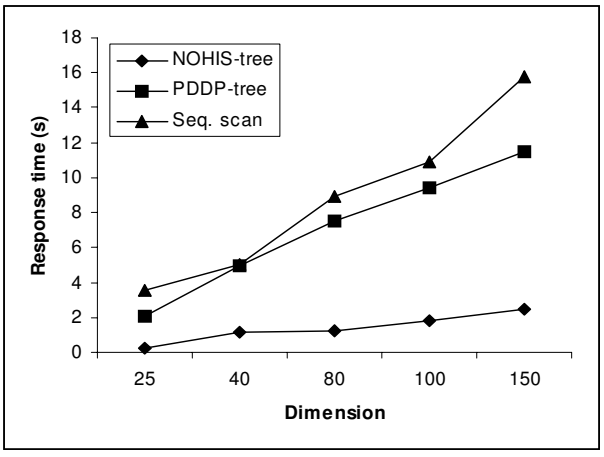


Fig. 5 Exp. 1 Response time, 20 NNs for 200 query vectors, increasing dimension

In the second experiment we evaluated the performance behavior while varying the database size. We measured total search time and the consulted leaves when using the same search methods applied on 5 bases of dimensionality 25 and different sizes (50,000 to 500,000), these databases have been generated from the database of 1,193,647 descriptors. Time in fig. 6.a represents the cumulated response time when searching in databases 20 NNs of each of the 200 query vectors. We can observe that NOHIS-tree gives the best times, the speed-up of NOHIS-tree in the total search time is ranges between 9.91 and 18.06 over PDDP-tree, and between 16.785 and 39.642 over the sequential scan. In fig. 6.b we show results of another comparison which is the number of the consulted leaves, we consider the mean of the leaves consulted when processing queries since leaves are the data nodes (clusters). This comparison explains the rapidity of the NOHIS-tree and shows the number of the consulted leaves for our proposed index which is fewer than the number of consulted leaves of the other compared methods. The speed-up with respect to the number of the consulted leaves in NOHIS-tree is ranges between 13.437 and 20.336 over PDDP-tree, and ranges between 29.440 and 69.148 over the sequential scan. The orientation of MBRs in NOHIS-tree avoiding overlapping explains the obtained results.

TABLE II
EXP. 2.A IMPACT OF DATA SIZE ON THE SEARCH TIME

Number of vectors	Number of clusters	Total search time (s)		
		NOHIS-tree	PDDP-tree	Seq. scan
50.000	600	0,210	2,083	3,525
100.000	1200	0,280	4,356	7,040
250.000	2000	0,621	11,216	17,616
350.000	2400	0,791	14,201	24,676
500.000	3000	0,892	14,792	35,361

TABLE III
EXP. 2 .B LEAVES ACCESSED WHEN COMPARING NOHIS-TREE WITH PDDP-TREE AND SEQ.SCAN

Number of vectors	Number of clusters	Number of consulted leaves (clusters)		
		NOHIS-tree	PDDP-tree	Seq. scan
50.000	600	20,380	273,850	600
100.000	1200	24,655	529,900	1200
250.000	2000	37,215	937,255	2000
350.000	2400	41,335	1081,835	2400
500.000	3000	43,385	882,295	3000

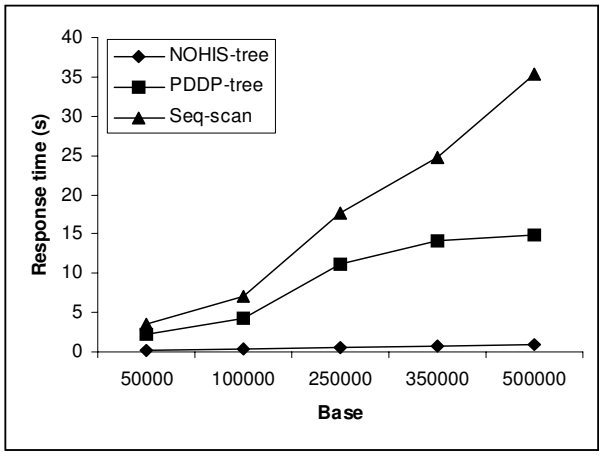


Fig. 6.a Exp. 2 Response time, 20 NNs for 200 query vectors, increasing size

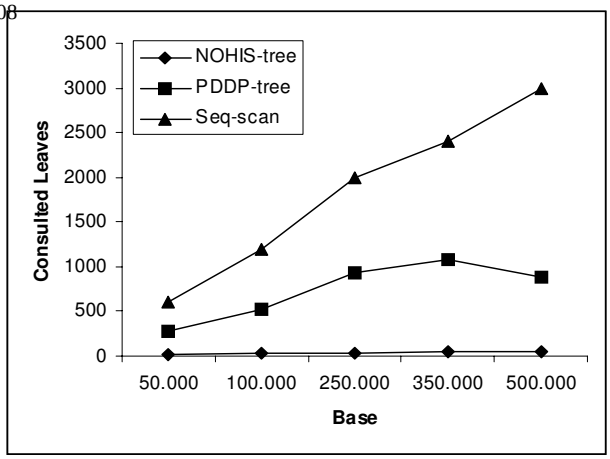


Fig. 6.b Exp. 2 Leaves accessed, 20 NNs for 200 query vectors, increasing size

TABLE IV
EXP. 3 COMPARISON OF OUR PROPOSED INDEX WITH SR-TREE

Number of vectors	Total search time (s)	
	NOHIS-tree	SR-tree
50000	0,621	3,515
100000	0,781	5,097
500000	2,202	11,396
1193647	6,399	30,974

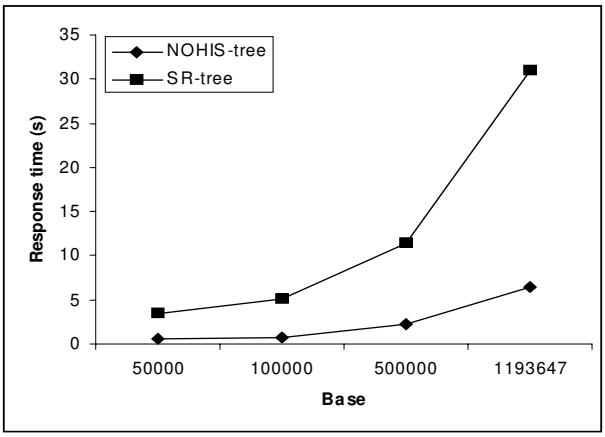


Fig. 7 Exp. 3 Total search time depending on the databases size

In the experiment 3, we compare the NOHIS-tree and the SR-tree, this index was chosen because it is considered one of popular and recent used index. We used the code version 2.0 of SR-tree provided by the authors, we retained the default parameters; SR-tree is build dynamically. In this experiment, the index and data of NOHIS-tree and SR-tree are stored on the disk and loaded in main memory when necessary. We use databases of dimension 30 and varying form 50,000 to 1,193,647 vectors. For the both indexes, we search 20 NNs for 200 query vectors, as in the previous experiments. Fig. 7 shows the total search time, NOHIS-tree outperforms SR-tree; it is 5.660 times faster than SR-tree when using the database of 50,000 and 4.840 times faster than the SR-tree when using the database of 1,193,647 vectors.

Through the experiments, we conclude that NOHIS-tree outperforms PDDP-tree and sequential scan in CPU time and the number of the consulted leaves; it also outperforms the competitive index SR-tree in CPU time.

IV. Conclusion

A new hierarchical indexing method for high dimensional data was proposed in this paper, called NOHIS. The proposed multidimensional index avoiding the overlapping between including forms of clusters presents the principal originality of our contribution. We also introduced the k-nearest neighbors search adapted to the proposed index NOHIS-tree.

The experimental results are conclusive on the corpus of bases chosen, and thus allow to validate the proposed index.

We compared the NOHIS-tree with sequential scan, PDDP-tree and the SR-tree. The NOHIS-tree improves all.

We plan to test consequent bases to measure the performances more, to compare the NOHIS-tree with other index and include outliers detection during clustering.

References

[1] C. Faloutsos, "Searching Multimedia Databases by Content". Kluwer Academic Publishers, 1996.
[2] A. Guttman. "R-trees: A dynamic index structure for spatial searching". In Proceedings of the ACM SIGMOD International Conference on Management of data, Boston, Masachussets, USA, pages 47-57. 1984.
3] T. Zhang, R. Ramakrishnan, M. Linvy, "Birch: An efficient data clustering method for very large databases". In Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada, pp.103-114, 1996.
4] N. Beckman, H.P.Kriegel, R. Schneider, & B. Seeger, (1990). "The R*-tree: An efficient and robust access method for points and rectangles". In Proceeding of the ACM SIGMOD International Conference on Management of Data, Atlantic City, New Jersey, USA, pp. 322-331, 1990.
5] S. Bertchold, D.A. Keim, H.P. Kriegel, "The X-tree: An index structure for hight-dimentional data". In Proceeding of the 22nd Internatioanl Conference on Very Large Data Bases, Mumbai (Bombay), India, pp. 28-39, 1996.
6] D.A. White and R. Jain, "Similarity Indexing with the SS-Tree". In Proceeding of the 12th Int'l Conf Data Eng., pp. 516-523, 1996.
7] N. Katayama, S. Satoh. "The SR-tree : An index structure for high-dimensional nearest neighbor queries". In Proceeding of the ACM SIGMOD, International Conference on Management of Data, Tuscon, Arizona, USA, pages 369-380. 1997.
8] J.T. Robinson, "The k-d-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 10-18, Apr. 1981.
9] D.B. Lomet and B. Salzberg, "The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance," ACM Trans. Database Systems, vol. 15, no. 4, pp. 625-658, 1990.
10] A. Henrich, "The LSDh-Tree: An Access Structure for Feature Vectors". Proc. 14th Int'l Conf. Data Eng., pp. 362-369, 1998.
11] [kd-Tree] J. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," Comm. ACM, vol. 18, no. 9, pp. 509- 517, 1975.
[12] D. L. Boley, "Principal Direction Divisive Partitioning", Data Mining and Knowledge Discovery 2(4):325-344, 1998.
[13] N. Bouteldja, V. Gouet-Brunet et M. Scholl. "Evaluation of strategies for multiple sphere queries with local image descriptors". In IS&T/SPIE Conference on Multimedia Content Analysis, Management, and Retrieval, San Jose CA, USA, 2006.
[14] S. Savaresi, D. L. Boley, S. Bittanti, G. Gazzaniga. "Choosing the cluster to split in bisecting divisive clustering algorithms". CSE Report TR-00-055, University of Minnesota, 2000.
[15] N. Roussopoulos, S. Kelly, F. Vincent. "Nearest Neighbor Queries". In Proceeding of ACM SIGMOD, May 1995.