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 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. Thekd-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.
The second online phase is the step during which 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 chosen 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 totality 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 with 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.

\[ COV = (M-w\cdot e)^T(M-w\cdot e) \] (1)

\[ w = \frac{v_1 + v_2 + \cdots + v_m}{m} = M.e \cdot \frac{1}{m} \] (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:

\[ g(x_i) = U^T(x_i - w) \geq 0 \implies x_i \in P_R \]

\[ g(x_i) = U^T(x_i - w) < 0 \implies x_i \in P_L \]

\( x_i \) is a multidimensional vector from the considered data.
Let us note $\text{MR}$ and $\text{ML}$ the two corresponding matrices of the two parts. Clustering algorithm [14] is given in figure 2 below.

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 $\mathbb{R}^n$, $e_1=(1,0,0,\ldots,0)$, $e_2=(0,1,0,\ldots,0)$, $e_3=(0,0,1,\ldots,0)$, …

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

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 $S(e_i)=u_i=U$.

Let $V = \frac{U - e_i}{\|U - e_i\|}$

and $H$ be the hyper-plane orthogonal to $V$, $H=V^\perp$, so that $H$ be the mediator hyper-plane of $e_i$ 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: $u_i = e_i - 2 \langle e_i,V \rangle V$, $1 \leq i \leq n$

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

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

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: $N_R = M_R - 2.V.T.V$, $M_R = (I - 2V^T.V)$, $M_R$

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 $S=(s_1,\ldots,s_p)$ et $T=(t_1,\ldots,t_p)$ where $s_i$ (resp. $t_i$) is the minimum (resp. the maximum) of the $i^{th}$ component of the considered vectors.

0. Start with the matrix of vectors $M(n \times m)$, and a desired number of clusters $c_{\text{max}}$.
1. Initialize Binary tree with a single Root node
2. For $c=2,3,\ldots, c_{\text{max}}$ do
3. Select node $C$ with largest Scatter Value
4. Create $L \& R := \text{left \& right children of } C$
5. For $i=1$ to $c_{\text{size}}$
   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 $c_{\text{max}}$ leaf nodes forming a partitioning of the entire data set.

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

Fig. 2 PDDP Algorithm
In an internal node (not a leaf) of the NOHIS-tree, following informations are stored: \(S_6, T_6, S_7, T_7\) 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.

\[B’\] is orthonormal, so coordinates of \(q\) in \(B’\) (\(q’\)) are given by the products scalar:

\[ q’ = [<q,u_1>,<q,u_2>,.....,<q,u_n>]^T \]

(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,

\[ MINDIST(q’,R) = \sum_{i=1}^{n} |q’_i - r_i|^2 \]

(7)

with:

\[ r_i = \begin{cases} s_i & \text{if } q’_i < s_i \\ t_i & \text{if } q’_i > t_i \\ q’_i & \text{else} \end{cases} \]

when \(q’\) is in \(R\), \(MINDIST(q’,R) = 0\).

**Algorithm 1 : K-NN Search**

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

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>
<thead>
<tr>
<th>Dimension</th>
<th>Number of clusters</th>
<th>NOHIS-tree</th>
<th>PDDP-tree</th>
<th>Seq. scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>600</td>
<td>0.210</td>
<td>2.083</td>
<td>3.525</td>
</tr>
<tr>
<td>40</td>
<td>600</td>
<td>1.151</td>
<td>4.977</td>
<td>5.037</td>
</tr>
<tr>
<td>80</td>
<td>600</td>
<td>1.261</td>
<td>7.551</td>
<td>8.882</td>
</tr>
<tr>
<td>100</td>
<td>700</td>
<td>1.853</td>
<td>9.414</td>
<td>10.866</td>
</tr>
<tr>
<td>150</td>
<td>600</td>
<td>2.494</td>
<td>11.456</td>
<td>15.802</td>
</tr>
</tbody>
</table>

![Fig. 4 Example of children’s MBRs](image-url)

**III. EXPERIMENTS**

Clustering algorithm and the search algorithm are implemented 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 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**

<table>
<thead>
<tr>
<th>Number of vectors</th>
<th>Number of clusters</th>
<th>Total search time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>600</td>
<td>0.210</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.083</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.525</td>
</tr>
<tr>
<td>100,000</td>
<td>1200</td>
<td>0.280</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.356</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.040</td>
</tr>
<tr>
<td>250,000</td>
<td>2000</td>
<td>0.621</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17.816</td>
</tr>
<tr>
<td>350,000</td>
<td>2400</td>
<td>0.791</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14.201</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24.876</td>
</tr>
<tr>
<td>500,000</td>
<td>3000</td>
<td>0.892</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14.792</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35.361</td>
</tr>
</tbody>
</table>

**TABLE III**

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

<table>
<thead>
<tr>
<th>Number of vectors</th>
<th>Number of clusters</th>
<th>Number of consulted leaves (clusters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>600</td>
<td>20,380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>273,850</td>
</tr>
<tr>
<td></td>
<td></td>
<td>600</td>
</tr>
<tr>
<td>100,000</td>
<td>1200</td>
<td>24,655</td>
</tr>
<tr>
<td></td>
<td></td>
<td>529,900</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1200</td>
</tr>
<tr>
<td>250,000</td>
<td>2000</td>
<td>37,215</td>
</tr>
<tr>
<td></td>
<td></td>
<td>937,255</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>350,000</td>
<td>2400</td>
<td>41,335</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1081,835</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2400</td>
</tr>
<tr>
<td>500,000</td>
<td>3000</td>
<td>43,385</td>
</tr>
<tr>
<td></td>
<td></td>
<td>882,295</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3000</td>
</tr>
</tbody>
</table>

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.

**TABLE IV**

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

<table>
<thead>
<tr>
<th>Number of vectors</th>
<th>Total search time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>0.621</td>
</tr>
<tr>
<td>100000</td>
<td>0.781</td>
</tr>
<tr>
<td>1193647</td>
<td>2.202</td>
</tr>
<tr>
<td></td>
<td>5.097</td>
</tr>
<tr>
<td></td>
<td>11,396</td>
</tr>
<tr>
<td></td>
<td>30,974</td>
</tr>
</tbody>
</table>

![Fig. 6.b Exp. 2 Leaves accessed, 20 NNs for 200 query vectors, increasing size](image)

![Fig. 6.a Exp. 2 Response time, 20 NNs for 200 query vectors, increasing size](image)

![Fig. 7 Exp. 3 Total search time depending on the databases size](image)
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