World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:5, No:11, 2011

# NOHIS-Tree: High-Dimensional Index Structure for Similarity Search

Mounira Taileb, Sami Touati

***Abstract*—**In Content-Based Image Retrieval systems it is important to use an efficient indexing technique in order to perform and accelerate the search in huge databases. The used indexing technique should also support the high dimensions of image features. In this paper we present the hierarchical index NOHIS-tree (Non Overlapping Hierarchical Index Structure) when we scale up to very large databases. We also present a study of the influence of clustering on search time. The performance test results show that NOHIS-tree performs better than SR-tree. Tests also show that NOHIS-tree keeps its performances in high dimensional spaces. We include the performance test that try to determine the number of clusters in NOHIS-tree to have the best search time.

***Keywords*—**High-dimensional indexing, k-nearest neighbors search.



## I. Introduction

THERE is a need for content-based image retrieval systems in many fields such as geography, security and criminal picture identification system. To implement a CBIR system two areas are involved which are image processing and similarity search. 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 a query image, a similarity search such k- nearest neighbors is made for each descriptor of the query image. Many similarity searches are carried out: as many searches as descriptors characterizing the image query. The use of a data structure to index the descriptors database is essential.. The need for an efficient indexing technique becomes a major concern for two reasons: the large quantity of images and the high dimensionality of descriptors.Existing indexing techniques work well with low-dimensional descriptors. However, their performance degrades when the dimension of descriptors increases. This phenomenon is known as the *curse of dimensionality* [2].The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 describes the indexing technique NOHIS-tree. Section 4 evaluates performance of NOHIS-tree; it is compared to sequential scan, PDDP- tree and SR-tree. SR-tree has been chosen because it is a multidimensional index proposed recently which still attracts the attention [3]. Finally, section 5 concludes the paper.

## II. Related Work

Obtaining a high-dimensional index can be made by using traditional techniques of indexing such as R-tree [4], 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 [5], DBSCAN use R*-tree [6] and X-tree [7]. Many high-dimensional index structures have been proposed, the most known and used are data-partitioning based index structure such as SS-tree [8], SR-tree [9], X-tree, considered as extensions of R-tree, and space-partitioning based index structure such as k-d-B-tree [10], hB-tree [11], and LSDh-tree [12] derived from kd-tree [13]. The R-tree-based index structures suffer from overlapping between bounding regions and the low fanouts, this 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, the high dimensional index technique called NOHIS was proposed [14]. *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) [15]. 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 cluster center to divide. A binary not balanced tree is obtained at the end of the clustering process. The contribution consists of 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. The search algorithm is adapted to the used MBRs.

## III. The NOHIS-TREE

The existing multi-dimensional indexing techniques 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 bounding forms (spheres or rectangles). In the case of the space-partitioning index; 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:

M. Taileb is with Information Technology department, King Abdul-Aziz University, Jeddah, KSA (e-mail: mtaileb@kau.edu.sa).

S. Touati, is with King Saud University, Riyadh, KSA (e-mail: stouati@ksu.edu.sa).

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 by the hyper-plane orthogonal to the first principal component passing through the cluster center 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 center of data given by (2):

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

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

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

Now let *U* be the first eigenvector of the matrix *COV* and $\delta_{max}$ the associated eigenvalue. Let us consider a normalized vector *X*. The dispersion of the data, included in the matrix *M*, with respect to *X* can be calculated as following:

$$D_X = |(M - we^T)^T X|^2$$
$$= X^T(M - we^T)(M - we^T)^T X$$
$$= X^T COVX$$

Since *COV* is a real symmetric matrix, one has

$$D_X \leq \delta_{max} \text{ and } D_X = \delta_{max} \text{ for } X \sim U$$

The most important dispersion of the data is then according to the first principal component $U$, so dividing accordingly it allows having 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.

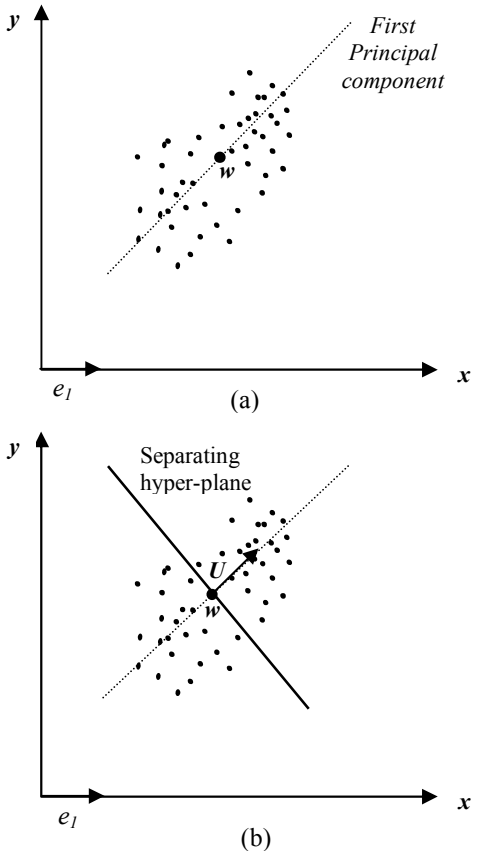
### *A.1. Data Partition*

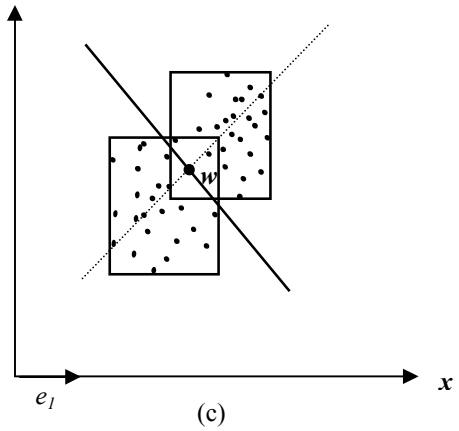
Figure 1 illustrates the example of data partition, in 2D, into two clusters. The whole of the vectors of data constitutes the matrix *M* (figure 1.a). Let us consider the matrix of covariance *COV* and its first principal component *U*. As explained previously, data is divided into two parts which are included in two rectangles having as axis the line engendered by the first principal component *U*. the two rectangles should not overlap. Division is made with the separating plan (hyperplane 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* for left) according to the following rule:

$$g(v_i) = U^T(v_i - w) \geq 0 \implies v_i \in P_R$$

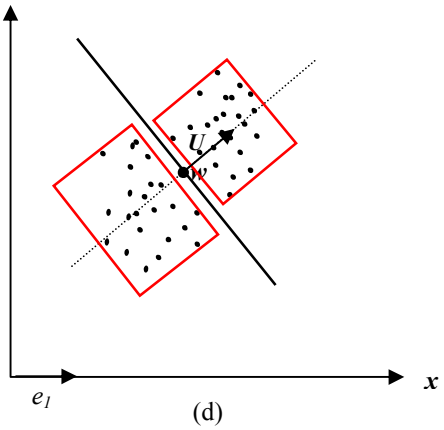
$$g(v_i) = U^T(v_i - w) < 0 \implies v_i \in P_L$$

$v_i$ is a multidimensional vector from the considered data.






(c)




(d)

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 the two parts. Clustering algorithm [16] 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 ScatterValue
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 dataset

Fig. 2 PDDP Algorithm

### *A.2. The change of reference mark*

The figure 1.c represents the case when the MBRs are 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 $R^n$, $e_1=(1,0,0\ldots,0)$, $e_2=(0,1,0\ldots,0)$, $e_3=(0,0,1\ldots,0)$, ….

The goal is to build an orthonormal base $B'=\{u_1,u_2,\ldots,u_n\}$ where a vector is equal to $U$ ($u_1 = U$), such a 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, $u_1 = S(e_1) = U$.

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

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

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

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

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

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

### *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 information 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 $N_1$, $N_2$, $N_3$ 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.

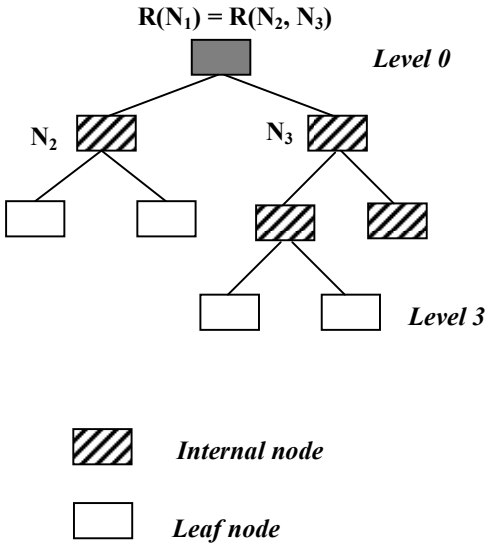



Fig. 3 Example of NOHIS-tree

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

$$\langle q, u_i \rangle = \langle q, e_i \rangle - 2\langle e_i, V \rangle . \langle q, V \rangle$$

$$q' = [\langle q, u_1 \rangle, \langle q, u_2 \rangle, \ldots\ldots, \langle q, u_n \rangle]^T$$

$$q' = q - 2\langle q, V \rangle . V \quad (6)$$

Distance separating $q$ from a rectangle $R$ is calculated as given in [17] using $q'$, $S$ and $T$. *MINDIST* is the distance between the query vector and an MBR.

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

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

And when $q'$ is inside the rectangle $R$, $MINDIST(q', R) = 0$

### *B.1. 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 index in the database, the index 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 computed. Only vectors having a distance lower than LN.dist[k] are inserted in LN by an insertion sort algorithm.

**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 sort of current vector in list_Neighbors
7.       end if
8.     end For
9.   Else
10.    For j : = 1 to 2 (the two child nodes)
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
24. 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 cannot 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 *father* node, the bounding forms of its children are contained in its bounding form and consequently, the distance from a query vector to the *father* node 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 *father* node R, as shown in the figure 4.
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.

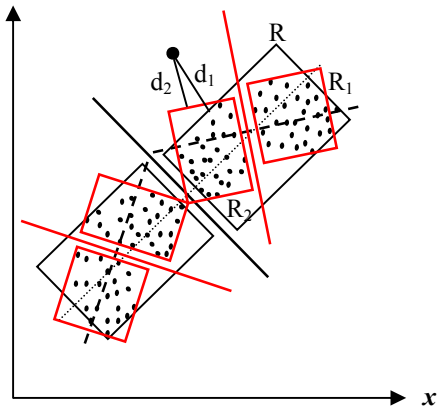



Fig. 4 Example of children MBRs

## IV. Experiments

Clustering algorithm and the search algorithm are implemented in C++. Algorithms run on a PC with Intel processor, its CPU is 1.8 GHz and 2 Go of RAM. To evaluate performances of similarity search using NOHIS-tree we performed various experiments. We use a dataset of 5,481,487 descriptors of dimension 30. Descriptors are points of interest derived from 22,991 images. In all experiments, we consider the cumulated response time when searching in datasets 20 nearest neighbors (NNs) for 200 query vectors.

In experiment 1, we study the impact of clustering on the search time. We are trying to determine the number of clusters to obtain the best search time or at least to define a range of minimum and maximum number of clusters giving the best search time.

To carry out this study we used two groups of datasets and several NOHIS-trees with different numbers of clusters are generated for both groups dataset. Indexes and datasets of the first group are loaded completely in main memory, and for the second group, indexes and datasets are stored on hard disk and loaded in main memory when necessary. Results of this experiment are given in figures 5 and 6. We present results when using the dataset of 969 729 descriptors, 30-d, from the first group of datasets and the dataset of 3,079, 771 descriptors, 30-d, from the second group of datasets. We found that, for the first group of datasets, search time decreases and becomes monotonous in the interval $[2\sqrt{n}, 4\sqrt{n}]$, where $n$ is the dataset size. Even if the monotony is verified beyond $4\sqrt{n}$, we suggest setting the number of clusters closer to the upper bound of the interval. And for the second group of datasets, values of number of clusters minimizing the search time are within the interval $\left[\frac{1}{5}\sqrt{n}, \sqrt{n}\right]$.

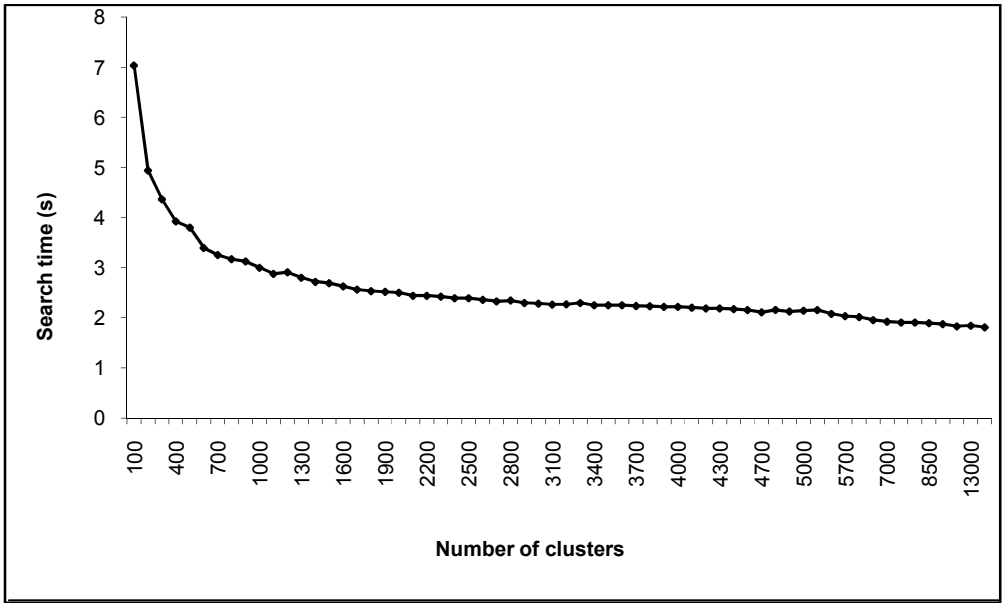



Fig. 5 Exp.1 Search time when varying number of clusters, 20 NNs, 200 query vectors, dataset of 969,729 vectors, dimension 30

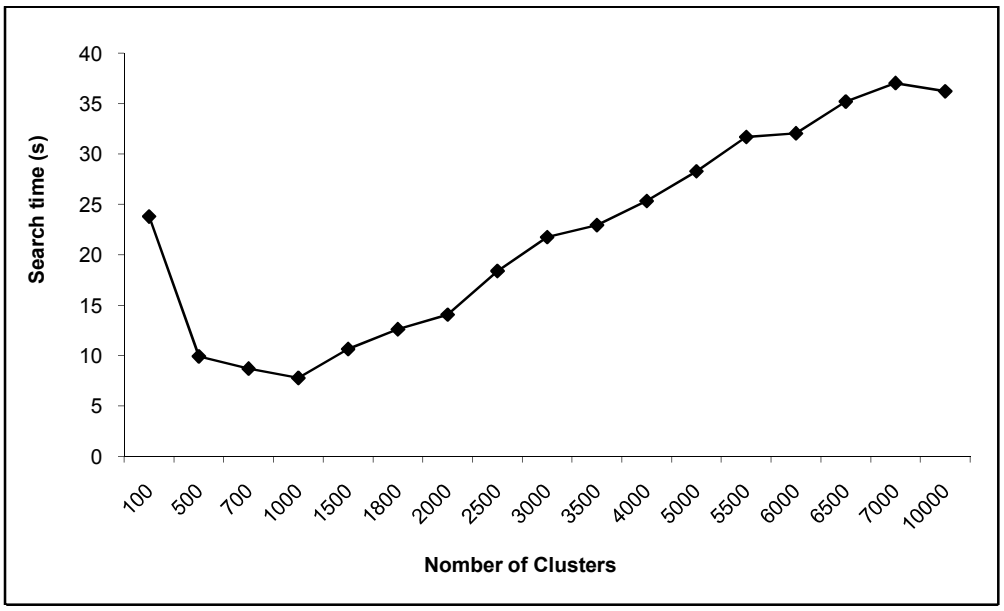



Fig. 6 Exp.1 Search time when varying number of clusters, 20 NNs, 200 query vectors, dataset of 3,079, 771 vectors, dimension 30

In experiment 2, 6 datasets of 50,000 vectors 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.

The goal is to study the impact of the dimensionality on the three search methods. Figure 7 and table I show the total search time for three search methods. NOHIS-tree significantly outperforms the PDDP-tree and sequential scan. In 25-dimensional space, the NOHIS-tree performs the queries 19.78 times faster than the PDDP-tree and 29.63 times faster than the sequential scan. Even in 150-dimension space, the NOHIS-tree is 5.91 times faster than the PDDP-tree and 8.146 times faster than the sequential scan. NOHIS-tree keeps its performances even in high dimensional space.

TABLE I
EXP.2 IMPACT OF DATASET DIMENSIONALITY ON THE SEARCH PERFORMANCE

| Dimension | Number of clusters | Serch time (s) | | |
|---|---|---|---|---|
| | | NOHIS-tree | PDDP-tree | Seq.scan |
| 25 | 499 | 0,156 | 3,087 | 4,623 |
| 40 | 546 | 0,906 | 6,173 | 6,187 |
| 80 | 571 | 1,062 | 8,317 | 10,202 |
| 100 | 574 | 1,562 | 9,987 | 12,297 |
| 150 | 589 | 2,125 | 12,561 | 17,312 |

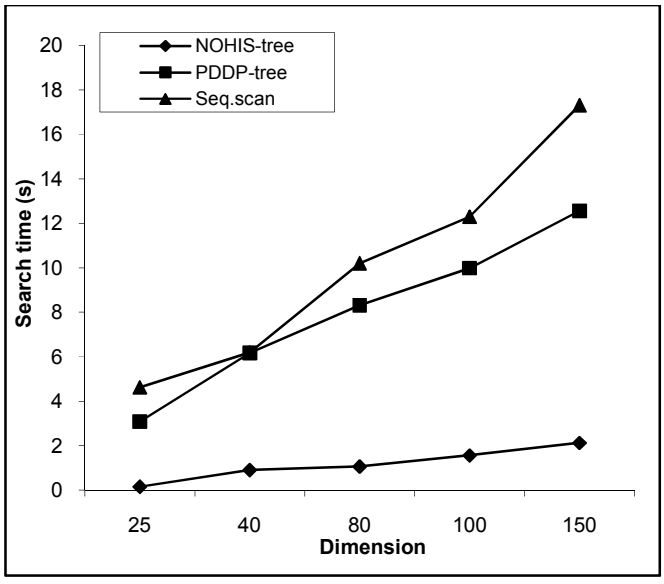



Fig. 7 Exp.2 Search time, 20 NNs, 200 query vectors, increasing dimension

TABLE II
EXP.3 COMPARISON BETWEEN NOHIS-TREE, SR-TREE AND SEQ.SCAN

| Size | Search time (s) | | |
|---|---|---|---|
| | NOHIS-tree | SR-tree | Seq.scan |
| 100 000 | 0,266 | 4,358 | 8,125 |
| 500 000 | 0,968 | 10,013 | 55,000 |
| 969 729 | 2,984 | 20,875 | 100,703 |
| 2 110 042 | 4,219 | 26,155 | 227,626 |
| 3 079 771 | 5,687 | 33,920 | 329,989 |
| 5 481 487 | 6,484 | 47,859 | 595,21 |

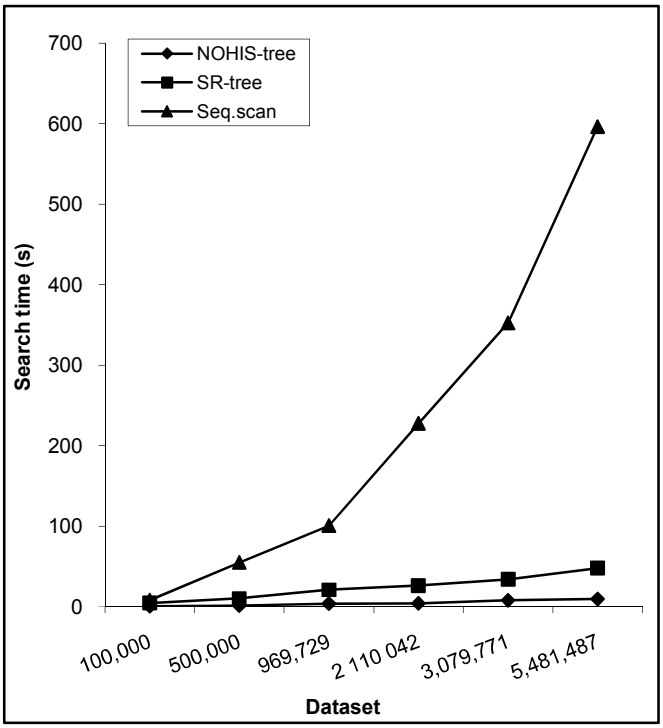



Fig. 8 Exp.3 Search time, 20 NNs, 200 query vectors, increasing dataset size

In the experiment 3, NOHIS-tree is compared to SR-tree and sequential search. 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 datasets of NOHIS-tree and SR-tree are stored on the disk and loaded in main memory when necessary. We use datasets of dimension 30 and varying from 50,000 to 5,481,487 vectors. Results are given in table II and figure 8. NOHIS-tree outperforms SR-tree; it is 5.96 times faster than SR-tree when using the dataset of 3,079, 771 vectors and 16.38 times faster than the SR-tree when using the dataset of 100,000 vectors. Also, NOHIS-tree is 30.54 times faster than sequential scan with the dataset of 100 000 vectors and 91.79 times faster than sequential scan with the dataset of 5,481,487 vectors.

## V. Conclusion

In this paper we evaluated performances of the hierarchical index NOHIS-tree when we scale up to very large databases. Results show that NOHIS-tree performs better than SR-tree and sequential scan search when the search is carried out on huge datasets. We also presented a study of the influence of clustering on search time. We included the performance test that tries to determine the number of clusters in NOHIS-tree to have the best search time; intervals were given to choose the appropriate number of clusters when building NOHIS-tree. Tests also show that NOHIS-tree keeps its performances in high dimensional spaces. We plan to further improve NOHIS-tree by including outliers detection.

## Acknowledgment

The authors would like to thank research center in the college of computer and information sciences, King Saud University, and the Information Technology department of Faculty of Computing and Information Technology, King Abdul-Aziz University, Jeddah, KSA, for their financial support to complete this study.

## References

[1] C. Faloutsos, "Searching Multimedia Databases by Content". *Kluwer Academic Publishers*, 1996.

[2] R. Bellman, "Adaptive Control Process: A Guided Tour". Princeton University Press, 1961.

[3] N. Bouteldja, V. Gouet-Brunet and M. Scholl, "Evaluation of strategies for multiple sphere queries with local image descriptors". *IS&T/SPIE Conference on Multimedia Content Analysis, Management and Retrieval*, San Jose CA, USA, 2006.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] A. Henrich, "The LSDh-Tree: An Access Structure for Feature Vectors*". Proc. 14th Int'l Conf. Data Eng.,* pp. 362-369, 1998.

[13] J. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching*," Comm. ACM*, vol. 18, no. 9, pp. 509- 517, 1975.

[14] M. Taileb, S. Lamrous and S. Touati, "Non Overlapping Hierarchical Index Struture", *International Journal of Computer Science*, vol. 3 no. 1, pp. 29-35, 2008.

[15] D. L. Boley, "Principal Direction Divisive Partitioning"**,** *Data Mining and Knowledge Discovery* 2(4):325-344, 1998.

[16] 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.

[17] N. Roussopoulos, S. Kelly, F. Vincent. "Nearest Neighbor Queries". *In Proceeding of ACM SIGMOD,* May 1995.