Cluster-Based Data Oriented Hashing

Sanaa Chafik*/*** Imane Daoudi** Mounim A. El Yacoubi* Hamid El Ouardi***

*Telecom SudParis / Mines Telecom Institute
Paris, France
Email: sanaa.chafik@telecom-sudparis.eu
mounim.el_yacoubi@telecom-sudparis.eu

**Hassan II University Casablanca
ENSEM - GREENTIC/ LISER
Casablanca, Morocco
Email: i.daoudi@ensem.ac.ma

***Hassan II University Casablanca
ENSEM - LISER
Casablanca, Morocco
Email: h.elouardi@ensem.ac.ma

***Abstract*—Many multidimensional hashing schemes have been actively studied in recent years, providing efficient nearest neighbor search. Generally, we can distinguish several hashing families, such as learning based hashing, which provides better hash function selectivity by learning the dataset distribution. The spacial hashing family proposes a suitable partition of the multidimensional space, more adapted to data points distribution. In spite of the efficiency of multidimensional hashing techniques to solve the nearest neighbor search problem, these techniques suffer from scalabity issues. In this paper, we propose a novel hashing algorithm, named Cluster Based Data Oriented Hashing, that combines space hashing and learning based hashing techniques. The proposed approach applies first a clustering algorithm for structuring the multidimensional space into clusters. Then, in each cluster, a learning based hashing algorithm is applied by selecting an appropriate hash function that fits the data distribution. Experimental comparaisons with standard Euclidean Locality Sensitive Hashing demonstrate the effectiveness of the proposed method for large datasets.**

I. Introduction

Finding the nearest neighbors (*NN*) in high dimensional Euclidean spaces is a fundamental problem in many multimedia applications. Recent research attentions have been shifted to develop efficient and scalable indexing methods for solving the *NN* search problem. In recent years, many multidimensional indexing methods have been proposed to solve *NN* search problem. These techniques can be classified in two main families: the family of conventional indexing methods and the family of approximate indexing methods. The first one is based on structuring multidimensional data points according to their distribution and location in the data space. A number of efficient conventional algorithms (e.g. R-tree [1] and KD-tree [2]) have been proposed. Unfortunately, this family of methods suffers from scalability issues, as query time is exponential in the dimension *d*. In fact, for large enough d ($d \geq 16$), the performances of this family get significantly degraded and become worse than the full sequential scan approach that compares a query to every point from the dataset. This problem is known as "*The curse of dimensionality*"[3]. The second family has been proposed to overcome *The curse of dimensionality* problem by using approximation algorithms that aim at finding points close enough to a query instead of finding the real closest ones. This family provides slightly less accurate results, but offers in return a sharp reduction in computation time. We find in this family VA-File [4] and its variant [5], the LSH methods [6], [7], [8], [9], [10], [11], [15], [16], [17], [18], [19], [24], [25], [26], [27], BitMatrix [12], KRA+-Block [13]. We focus in this paper on one of the most promising methods for performing approximate nearest neighbors search in high dimensional space, namely that based on the concept of Locality-Sensitive Hashing (LSH). The choice of this class of methods is justified by their quality and effectiveness with respect to several multidimensional indexing methods [14]. The LSH concept was firstly introduced by Indyk et al [6] for Hamming space, and was later extended to Euclidean space by Datar and al [7] providing the popular package Exact Euclidean LSH (E2LSH[1]).

Existing hashing techniques can be broadly classified in two categories, *data independent hashing* techniques or *randomized hashing* techniques and *data dependent hashing* techniques or *learning based hashing* techniques. In the former, hash functions are chosen independently from the dataset points, using several random projections. These functions allow hashing similar points in the same bucket[2] with high probability, thus accelerating the search process by placing a given query in the appropriate bucket using the same space structuring random projections. Several *independent hashing* approaches have been proposed, including the basic LSH [6]. Although this category has proved to be the most adaptive and effective for structuring high dimensional heterogeneous data spaces, it has limitations that affect structure and query performance. The main limitation of the basic LSH is memory consumption. In order to increase the probability to map similar points to similar hash values, the LSH concatenates several random independent hash functions, which leads to a severe redundancy of the hash values as well as a redundancy of the hash tables. Moreover, when using large scale datasets, more hash tables are required, which leads to high memory consumption. Recent research studies, therefore, have been more interested in *data dependent hashing* techniques [20], [21], [22], [23], [28], to closely fit data distribution and to provide better selectivity than the usual random LSH projection. As a matter of fact, *learning-based hashing* approaches have shown significant improvement in terms of speed and memory consumption over randomized hashing approaches, albeit the improvement has been achieved only for limited hash code sizes. Indeed, the performances of *learning-based hashing* techniques degrade for large scale datasets, due to the limited hash code sizes provided by *learned-based hashing* techniques.

[1]http://www.mit.edu/ andoni/LSH/

[2]A bucket is a data type that groups objects together. The term is used in hashing algorithms, where different items that have the same hash code (hash value) go into the same "bucket".

Unfortunately, as most of current hashing based techniques suffer from scalability issues, it is difficult to perform an efficient quick search in high dimensional datasets without consuming large memory space.
Recently a new class of hashing techniques called *Space hashing*, combining the two categories mentioned above, has been proposed. This new family, based on space partitioning, includes *structured* space hashing techniques, ans consists of partitioning the dataset space into equal sized cells using random projection or lattices; we find in this class of methods [7], [27], [15]. By contrast, *unstructured* space hashing techniques were introduced in [16], for a suitable partition of the multidimensional space, in the sense that it is more adapted to the distribution of data points. *Unstructured* hashing techniques [16] [25] [26] have provided good results for real databases.
In this paper, we propose a new hashing scheme combining *unstructured* and *learning based hashing*, called *Cluster-based Data Oriented Hashing* for efficient high dimensional NN search. The proposed hashing scheme uses the clustering algorithm for structuring the multidimensional space into clusters. Then, in each cluster, a *learning based hashing* algorithm is applied, by selecting the appropriate projections that fit data distribution within that cluster. Our new approach outperforms the E2LSH method in terms of computation time and memory space without compromising search quality. The rest of this paper is organized as follows. We first discuss preliminaries in Section 2, and then introduce our algorithm in Section 3. Section 4 gives the results of the experiments that compare our algorithm with the E2LSH approach. Finally, conclusions and perspectives are presented in Section 5.

d	Dimension
U	Universe
$B(p,r)$	Ball of radius r centered at a data point p
P_r	Probability
h	Hash function
$\|.,.\|$	The Euclidean Distance
$O(.)$	Complexity

Table 1. Notations

II. Preliminaries

A. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is one of the most popular methods for solving the *NN* search problem in high dimensional space. The LSH is based on the use of several hash functions describing the distribution of data in multi-dimensional space. The hash function assigns the same hash value to the closest points with high probability. Then, the *NN* search is performed by hashing the query point and retrieving points stored in the same hash table buckets to the query. The LSH approach relies on *locality-sensitive* hash functions. A family F of hash functions is called a *locality-sensitive* family if it satisfies the following definitions:
Definition.1 [6]: An LSH family $F = \mathbb{R}^d \to U$ is called (r_1, r_2, P_1, P_2)-sensitive, if for any $p, q \in R^d$, with $P_1 > P_2$ and $r_1 < r_2$:

$$\text{If } p \in B(q, r_1) \text{ then } P_{r_F}[h(p) = h(q)] \geq P_1$$

$$\text{If } p \in B(q, r_2) \text{ then } P_{r_F}[h(p) = h(q)] \leq P_2$$

Definition.2 [17]: A *locality-sensitive hashing* scheme is a distribution on a family F of hash functions operating on a collection of points, such that for two points p, q :

$$P_{r_F}[h(p) = h(q)] = sim(p, q)$$

Here $sim(.,.)$ is some similarity function defined for a collection of points. To increase the gap between high probability P_1 and low probability P_2, and to achieve high search accuracy, it is necessary to use several hash functions $g_j : \mathbb{R}^d \to \mathbb{Z}^k$ defined as: $g_j(p) = (h_{1,j}, h_{2,j}, ..., h_{k,j})$, characterizing the hash tables, where $h_{i,j}$ with $(1 \leq i \leq k, 1 \leq j \leq L)$ are selected randomly and independently from F. These functions hash all data points and build the index structure, by placing each point p in a bucket $g_j(p)$. It is worth noting that the buckets store only the references of points to facilitate access to data points. The search algorithm for a query point q begins by computing the hash values $g_1(p), g_2(p), ..., g_L(p)$, then the distance between the query point and the points of the same bucket. Finally, the nearest points to the query are selected; they represent the results of the search. One of the most promising *LSH* algorithms is Exact Euclidean LSH (E2LSH) method. E2LSH was proposed in [7] as a solution for the high dimensional $(R, 1-\delta)$-*NN* search problem in the Euclidean space l_2. For a query point q, the E2LSH method reports all points $p \in \mathbb{R}^d$ satisfying $\|p, q\|_2 \leq R$ with a probability at least equal to $(1-\delta)$ (δ is the probability that a near neighbor is not retrieved). The index structure of the E2LSH is performed by calculating the hash value of each point of the dataset, using the hash function $h_{a,b} : \mathbb{R}^d \to \mathbb{Z}$, defined as:

$$h_{a,b}(p) = \lfloor \frac{a.p + b}{w} \rfloor \tag{1}$$

a is a random d-dimensional vector of p-stable distribution [7] and b is a real number chosen uniformly from $[0, w]$, where w is a positive real number representing the width of the LSH function. Intuitively, the hash function $h_{a,b}$ projects the point p on a line whose direction is identified by a. Then, the projection of the point p is shifted by a constant b. In such a scheme, the line is segmented into intervals of length w. To improve the hashing discriminative power, a second hash function g_j is constructed using several functions defined as: $g_j(p) = (h_{1,j}, h_{2,j}, ..., h_{k,j})$, where $(1 \leq j \leq L)$. The similar points of the data set (having the same g_j) are stored in the same hash tables bucket. We note that the E2LSH performance depends crucially on parameters K and L: K refers to the number of hash functions and L represents the number of hash tables. The value of L is determined by requiring that the probability to find the true *NN* is at least $(1-\delta)$, which implies: $L = \lfloor \frac{log(\delta)}{log(1-P_1^k)} \rfloor$. The K value is chosen as a function of the dataset to minimize the query time. Although the E2LSH is a promising approach for solving the *NN* search problem, it presents a limitation in terms of memory space storage. To ensure good performances, the E2LSH uses several hash tables, which requires too much memory space.

B. Space hashing

Space hashing techniques have been proposed to improve LSH performance. Classical LSH can be seen as a projection onto one dimension, which explains information loss in the dimensionality reduction process. Several *structured* based space hashing methods have been proposed to overcome the information loss problem by projection onto higher dimensions. The Leech Lattice [15] proposed the projection onto 24 dimensions while $E8$ Lattice enables 8-dimensions projections [27], that offer a structured partition of the space better than random partition used in classical LSH. These structured space hashing approaches offer an efficient regular partition adapted for uniformly-distributed datasets (Figure 1a and Figure 1b), but for real datasets, their performance degrades due to the imbalance in the distribution of multidimensional points in hash table buckets, which may increase the computation time for filled hash table buckets and degrade the search quality for empty hash table buckets. To overcome these problems, *unstructured* space hashing methods have been proposed. Based on clustering algorithms, this class of methods partitions the dataset space into *Voronoi* cells, with size and shape adapting to real dataset space (Figure 1c and Figure 1d). Then, each point in the dataset space is hashed to a cell index using the hashing function defined as:

$$h_C(p) = argmin_{i=1,...,K} d(p, c_i) \tag{2}$$

Where $C = \{c_1, c_2, ..., c_K\}$ represents the set of *Voronoi* diagram seeds and the integer *K* is the number of cells. Clustering is the process of organizing data objects into a set of disjoint classes called Clusters. *K*-means is one of the most popular clustering methods, it has been widely used in many applications. The main idea is very simple: starting with a set of randomly chosen initial centers, one repeatedly assigns each multidimensional point to its nearest center, and then recomputes the centers given points assignment. This clustering technique has proven to be effective in many applications. However, the algorithm requires prior knowledge of the number of clusters which is difficult to define. The *K*-means algorithm presents another limitation: it is sensitive to the choice of initial centers, which leads to imbalanced distribution in clusters. Many clustering algorithms have been proposed to solve the *K*-means problem: [29] proposed an automatic selection of the number of clusters, and [30] and [31] proposed a better initialization of centers taking into account the distribution of data. *K*-means LSH [16] is an *unstructured data-adaptive hashing* method for approximate *NN* search, where each hash function represents the *Voronoi* diagram obtained from the *K*-means or hierarchical *K*-means [32] applied on the dataset. Each *Voronoi* cell hash function returns a hash value referring to the points it contains. Distribution Free Locality Sensitive Hashing (DFLSH) [26] uses the same principle of *Voronoi* cell hash function as *K*-means LSH approach, but instead of applying the *K*-means clustering to obtain *Voronoi* diagram seeds, it chooses randomly seeds from the dataset, which generalizes the application of the DFLSH to different metric spaces. *Voronoi*-LSH [25] is a generalized unstructured space hashing method that proposes the use of different center initialization algorithms for *Voronoi* diagram seeds, and proposes a parallel design for the index structure scalability.

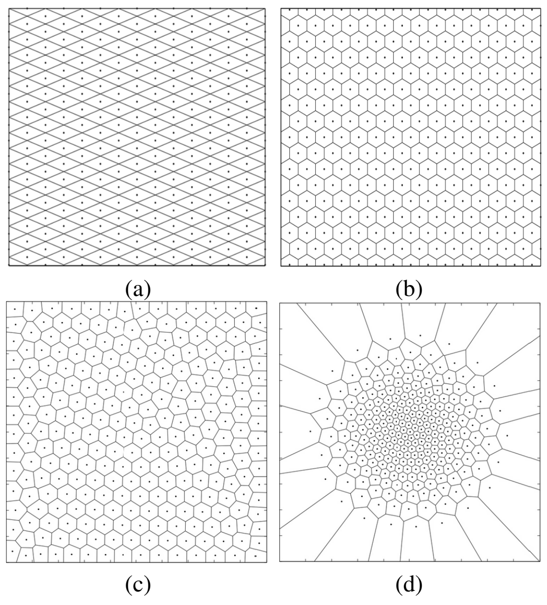

Figure 1. Voronoi diagram of random projection (a), Lattice A2 (b) and k-means clustering for uniform and Gaussian distribution (c, d) [16]

C. Learning based Hashing

The hashing methods mentioned above propose to partition the multidimensional space before proceeding to hashing, which creates a suitable index structure for different metric spaces. In recent years, a new family of hashing functions has been developed; it consists of learning the multidimensional space, to consider the distribution of data points and design a better index structure. Most of hash functions used on *learning based hashing* rely on hyperplanes for partitioning the data points space into two sets and then assign two different binary codes. Embedding data into binary codes in Hamming space provides fast distance computation and hence fast data retrieval and less memory consumption. One of the most popular hash functions is the linear projection:

$$h(p) = sign(a^T.p) \in \{0, 1\} \tag{3}$$

where $sign(a^T.p) = 1$ if $a^T.p \geq 0$ and $sign(a^T.p) = 0$ otherwise. a is a d-dimensional random vector defining the separating hyperplane. Despite the success of this learning based hashing methods, their performance decreases with scaling up: for high dimensional data spaces, the binary codes are generated with information loss. Thus re-ranking by exact Euclidean distances is required to find the nearest neighbors, which increases the computation time. Another learning based hashing, proposed in [24], consists of selecting adaptive projecting vectors, using PCA algorithm. Adapted to Euclidean space, PCA-LSH uses function (1) as hash function, where the d-dimensional vector a of equation (1), defines the PCA vector projection; this approach provided better performance when compared to the classical Euclidean LSH.

III. Proposed Approach

In this section, we present the index structure and the nearest neighbor search algorithm of the new proposed approach, Cluster-based Data Oriented Hashing.

A. The Basic Idea

The Cluster-based Data Oriented Hashing is a new efficient indexing approach based on a quantization algorithm (K-means clustering algorithm) for structuring the multidimensional space into clusters, and in each cluster a hashing approach is applied making easy and quick the search process. In the following, we detail the structure index and the search process of the proposed approach.

B. Index Structure

After a thorough study of different hashing techniques, it was found that the *spatial hashing* methods provide good performance, particularly *unstructured hashing* methods, as they adapt well to different multidimensional space distributions. *Learning based hashing* techniques showed also good results for small databases requiring less memory space. Based on research results on hashing based techniques, we propose a new hashing approach, inspired by these two classes of hashing methods: *learning* and *space hashing*. The proposed approach begins with structuring the multidimensional space into cells, using the K-means algorithm. In each cluster, the learning based hashing technique, PCA-Hashing is applied for structuring clusters subspaces. The function of equation (1) is used as hash function in each cluster, with the d-dimensional vector a defining the PCA vector projection. The PCA-Hashing algorithm has proven to be very effective for structuring subspace clusters. This is due to the reduced number of points in clusters. The approach Cluster-based Data Oriented Hashing provides an efficient index structure, where each hash table presents a cluster with a center c_i, $(1 \leq i \leq K)$ (Figure 2). Therefore, the number of hash tables that defines the memory space is represented by the number of clusters K. The proposed approach uses one more hash table for storing the cluster centers to speed up the search process for locating a given query point. So the memory space required by the structure index is of the order $O(n+K(1+d))$ (n is the dataset size and d is the dataset dimension).

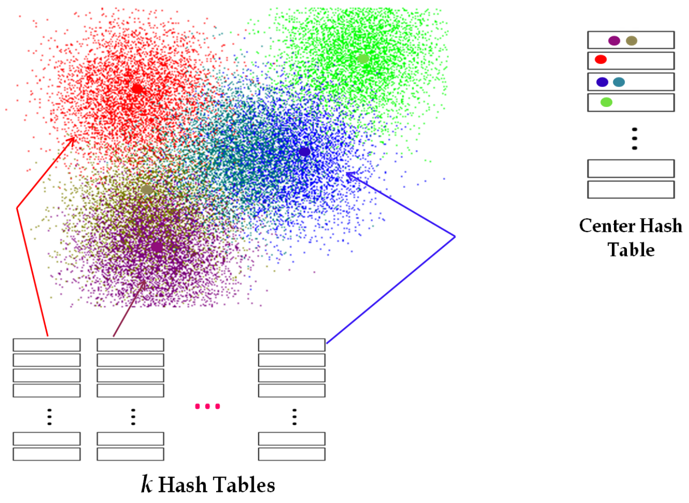

Figure 2. Index Structure of the Cluster-based Data Oriented Hashing approach

C. Search Process

The search process consists first, of pinpointing the location of the query point in the space by defining its nearby clusters. The proposed approach provides two ways for the location of the query, depending on the number of clusters. If the number of clusters K is less or equal to a fixed threshold K_{Max} $(K \leq K_{Max})$, the new approach defines for each cluster i a cluster membership degree β_i, $(1 \leq i \leq K)$, that represents the probability to visit the cluster i, defined by the distance between the query point and the center of the cluster i divided by the sum of the distances between the query and the clusters center. If the cluster membership degree is less than a membership degree threshold β_T than the query is hashed inside to the cluster but if not the cluster is not visited. But if the number of clusters is large $(K > K_{Max})$, which requires a significant distance computation time, our approach proposes to hash the query point in a center hash table and retrieve the closest clusters. After the location of nearby clusters, the query point is hashed in the hash tables of these clusters, which enables hashing the query point into the appropriate buckets. Then the Euclidean distances between the query point and the points of the same bucket are computed and the nearest neighbors are retrieved.
The algorithm 1 describe the proposed search process.

```
Input  : Queries $Q$
         Membership degree threshold $\beta_T$
         Number of clusters $K$
         Threshold $K_{Max}$
         Search Radius $R$
         Set of centers of cluster $C$
         Set of closest centers $C_{centers}$
         Set of point in the bucket $B$
Output: Query result List
Initialization : Set result List empty;
for Query point $q$ in $Q$ do
    if $(K \leq K_{Max})$ then
        for Center $c$ in $C$ do
            Calculate the membership degree $\beta_c$;
            if $(\beta_c \leq \beta_T)$ then
                Add center c to $C_{centers}$;
            end
        end
    else
        Hash the query point $q$ in the center hash table
        to find the corresponding bucket;
        Add centers of the bucket to $C_{centers}$;
    end
    for Center $c$ in $C_{centers}$ do
        Hash the query point $q$ in the cluster hash table
        centered on c and find the corresponding bucket
        $B$;
        for Point $p$ in $B$ do
            Calculate $d(q,p)$;
            if $(d(q,p) \leq R)$ then
                Add point p to the result List;
            end
        end
    end
end
```

Algorithm 1: Search Process

IV. Experimental Result

In this section we present the experimental results obtained after performing several tests on our proposed approach and on the E2LSH, using real and synthetic databases.

A. Datasets and Evaluations Metrics

To evaluate the performance of the proposed approach, we compute query time, space consumption and accuracy on real and synthetic datasets. The real dataset used in our experiments, *GIST1M*[3], is a big image descriptor dataset, containing one million points of 960 dimensions. To observe the behavior of the new approach in scaling, we use a synthetic dataset created from *GIST1M* database. We created 250.000 and 500.000 datasets by selecting randomly 250.000 and 500.000 points from 1.000.000 points in GIST1M dataset. Then for the 250.000 dataset we reduce the dimension from 960 to 60 to obtain four datasets with different dimensions. The evaluation of the accuracy is performed by using the ratio criterion described in [8]. For a given query q, let $p_1, p_2, ..., p_r$ be the *r-NN* retrieved, sorted in ascending order of their distances to q, and let $p_1^*, p_2^*, ..., p_r^*$ be the true *r-NN* obtained using the sequential scan. We define the *overall ratio* (*OR*) for approxiamte *NN* search with respect to q as:

$$OR(q) = \frac{1}{r}\sum_{l=1}^{r} \frac{\|p_l, q\|}{\|p_l^*, q\|} \quad (4)$$

In our accuracy test, we report the *mean overall ratio* (*MOR*) over the query set. In our experiments, the queries dataset is obtained by randomly selecting 1000 points from the datasets and the the remaining ones are used to form the test databases. We report also in our experiments query time, which is the time consumed to perform queries search and the memory space for storing the index structure.

B. Parameter Settings

To construct the index structure, the Cluster-based Data Oriented Hashing approach requires many parameters, including the number of clusters *K*, the number of PCA vectors in each cluster, w the length of the interval and β_T the membership degree threshold : The number of clusters is chosen in such a way to ensure efficient search without increasing the query time (a large number of clusters may provide a bad index structure which degrades search accuracy and query time. A small number of clusters may increase the query hashing process time in visited clusters hash tables). The number of PCA vectors is determined by the number of eigenvectors corresponding to the highest eigenvalues carrying most data distribution information. After several experimental tests, we choose a suitable interval length w, membership degree threshold β_T and an appropriate number of clusters for each database.

C. Performances on Real Dataset

We evaluate first the performance of the proposed approach on the real dataset. Table 2 summaries the results obtained. We note that the proposed approach provides an accuracy search similar to that of E2LSH, while requiring less time and memory space. As we can see, the proposed geometric structure provides efficient performance and is suitable for high dimensional datasets.

Methods	**MOR**	**Query Time**	**Memory**
E2LSH	1,0002347	2,62015	10.56
CDO-Hashing	1,00002074	0,860347	0.011176

Table 2. Performance of the Cluster-based Data Oriented Hashing (CDO-Hashing) and the E2LSH on real Dataset

The Cluster-based Data Oriented Hashing approach has proven to be very effective in structuring the large *GIST1M* database when compared to the E2LSH approach that requires a lot of space memory. The proposed approach requires a small space memory with respect to the database size and the number of clusters.

D. Performances on Synthetic Datasets

To evaluate the behavior of the proposed approach in scaling, we performed several tests on different datasets with different sizes and dimensionalities.

1) Search Quality Test: Figures 3 and 4 present the accuracy result for different database sizes and dimensions respectively. We note that the proposed approach outperforms the basic Euclidean LSH in scaling. As the dimension and the database size increase, both approach quality decreases, but as we can see, the Cluster-based Data Oriented Hashing keeps a high search quality till the dimension $d = 760$, while the search quality of the E2LSH approach degrades starting from the dimension $d = 60$. As the dataset size increases, we observe that the search quality of both approaches decrease till the size 1.000.000 where we can note a slight improvement but the accuracy remains low for both approaches with a bit better rate for the proposed approach.

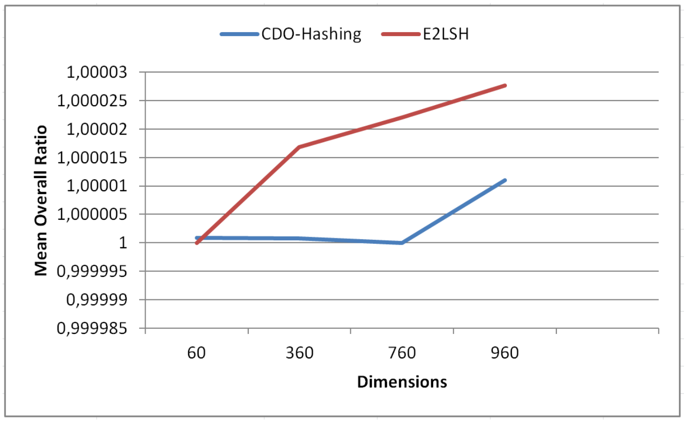

Figure 3. Mean Overall Ratio of 1000-*NN* vs. Dimension for dataset size $n = 250.000$

[3]http://corpus-texmex.irisa.fr

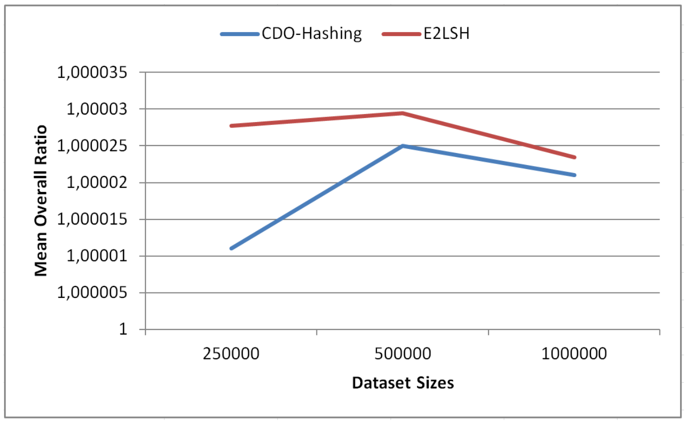

Figure 4. Mean Overall Ratio of 1000-*NN* vs. Dataset Size for dimension $d = 960$

2) Query Time Test: To evaluate the Cluster-based Data Oriented Hashing approach query time in scaling, we computed the mean query time of a set of queries for different dimensionalities and database sizes.

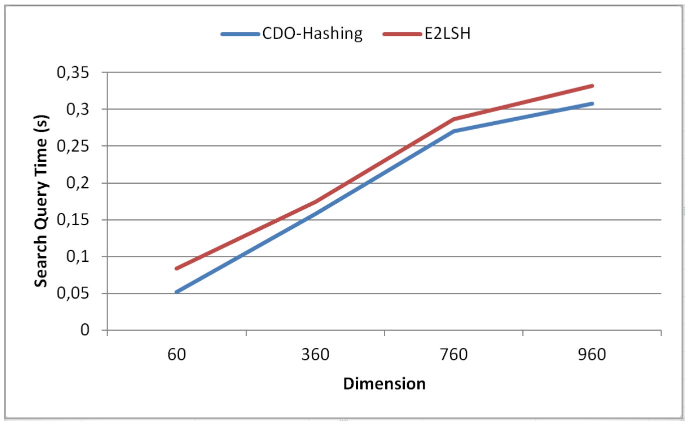

Figure 5. Search Query time vs. Dimension for dataset size $n = 250.000$

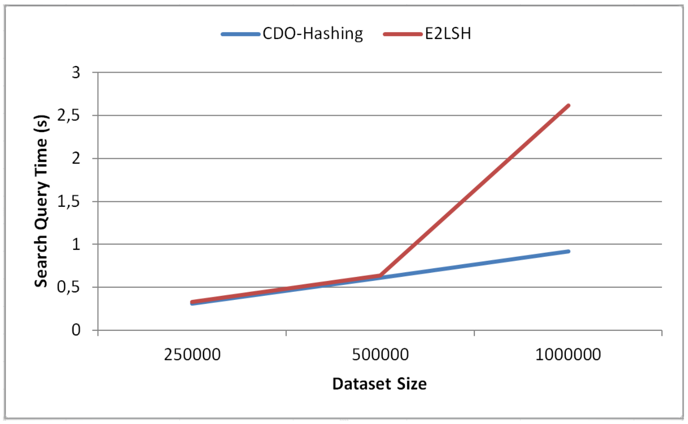

Figure 6. Search Query time vs. Dataset Size for dimension $d = 960$

Figures 5 and 6 show the speed of the Cluster-based Data Oriented Hashing and the E2LSH approaches with respect to database size and dimensionality. We note that both approaches present the same evolution rate in computing time as the database size and dimension increase. In fact, we can see that the proposed approach is slightly faster than E2LSH. But when the dataset size exceeds $n = 500.000$, the E2LSH becomes very slow in comparison to the new hashing scheme that keeps the same time evolution rate.

3) Memory Consumption Test: Since the Cluster-based Data Oriented Hashing approach organizes data points of each cluster in one hash table, it requires only *O(n+K(1+d))* (*n* is the dataset size and *d* is the dataset dimension) of memory space. However, the E2LSH, based on random projections, requires many hash tables to ensure good search accuracy, and its required space memory is equivalent to *O(nL)*. Table 3 and 4 summarize the obtained results on different dimensionalities and database sizes.

Dimensions	**60**	**360**	**760**	**960**
E2LSH	1,655	0,642	6,527	10,41
CDO-Hashing	0,00282	0,003	0,00324	0,00336

Table 3. Space Memory (GB) vs. Dimension for dataset size $n = 250.000$

Dataset Size	**250.000**	**500.000**	**1.000.000**
E2LSH	10,41	8,281	10,561
CDO-Hashing	0,00336	0,00644	0,0123

Table 4. Space Memory (GB) vs. Dataset Size for dimension $d = 960$

Our approach is much more economical in terms of memory space and speed than the E2LSH. The latter requires a huge memory space to ensure efficient search quality.

E. Discussion

The Cluster-based Data Oriented Hashing is a solution of nearest neighbors search problem in high dimensional space. Initially, a quantization algorithm is applied (K-means clustering algorithm) for structuring the multidimensional space into a set of clusters to help locating nearest points in the space. Then in each cluster a hashing algorithm is applied to capture similar points in the same bucket of the hash table, which reduced the query time while ensuring good search accuracy, by using the PCA directions as a projection lines in equation (1). The Cluster-based Data Oriented Hashing mainly depends on the parameters: the interval length w, the membership degree threshold β_T and the number of clusters K that define the performance of the approach. The parameters selected in our experimental tests were fixed by choosing the ones that provide sufficient accuracy. In the tests above, we note that the proposed approach provides good performance than the E2LSH, for different database sizes and dimensions. However, one can also notice that the dimension 760 is a critical point for our approach, which brings us to the limits of the Cluster-based Data Oriented Hashing approach, a judicious choice of parameters, can provide very good research results in an appropriate search time but an absurd choice may lead to a sharp performance reduction. In our experimental tests, we set the parameter values so as to ensure an acceptable quality, considered sufficient to demonstrate the performance of the proposed approach, but an in-depth study of the setting will provide better results. For our eventual work, we intend to find a solution for the choice of suitable parameters, and consider larger databases especially in terms of dimensionality to evaluate the proposed approach performance.

V. Conclusion

In this paper, we proposed a novel index structure, Cluster-based Data Oriented Hashing, for high dimensional nearest neighbors search. The proposed approach combines the advantages of both spacial hashing and learning based hashing techniques, and provide good performance with respect to the popular E2LSH. In our future work, we intend to further improve the search accuracy, especially for high dimensional datasets, and evaluate the proposed approach on bigger dataset, a parallel processing can be considered to accelerate more the search process and to manage larger query sets.

Acknowledgment

This work is supported by Mines Telecom / Telecom Sud-Paris Institute - France and the National Center for Scientific and Technical Research (CNRST) - Morocco.

References

[1] A. Guttman. R-trees: A dynamic index structure for spatial searching. In SIGMOD, 1984, pp. 47-57.

[2] J. L. Bentley. K-d trees for semi dynamic point sets. In SoCG, 1990, pp. 187-197.

[3] Weber, H.J. Schek and S. Blott, A quantitative analysis and performance, study for similarity-search methods in high-dimensional spaces. In Proceedings of 24rd International Conference on Very Large Data Bases, New York, USA, 1998, pp.194-205.

[4] S. Blott and R. Weber.A Simple Vector-Approximation File for Similarity Search in High-Dimensional Vector Spaces. Technical Report 19, ESPRIT project HERMES, March 1997.

[5] L. Ye and Y. Hua. The CMVAI-File: An Efficient Approximation-Based High-Dimensional Index Structure, Multimedia Information Networking and Security (MINES), 2010.

[6] P. Indyk, and R. Motwani, Approximate nearest neighbor: Towards removing the curse of dimensionality. In Proceeding STOC '98 Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp.604-613.

[7] M. Datar, N. Immorlica, P. Indyk and V. Mirrokni. Locality sensitive hashing scheme based on p-stable distributions. In Proceedings of the ACM Symposium on Computational Geometry, 2004.

[8] J. Gan, J. Feng, Q. Fang and W.Ng, Locality-sensitive hashing scheme based on dynamic collision counting. In proceeding SIGMOD '12 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York, USA, 2012, pp. 541-552.

[9] Y. Liu, J. Cui, Z. Huang, H. Li, and H. H. T. Shen,SK-LSH: an efficient index structure for approximate nearest neighbor search. In: Proceedings of the VLDB Endowment. 40th International Conference on Very Large Data Bases, Hangzhou, China, pp. 745-756. 1-15 September 2014.

[10] D. Gorisse, M. Cord and F. Precioso. Locality-sensitive hashing for chi2-Distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 34, 2012, pp. 402409.

[11] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing.IEEE Trans. Pattern Anal. Mach. Intell., 34(6):10921104, 2012.

[12] C. Calistru, C. Ribeiro and G. David. High- Dimensional Indexing for Video Retrieval. Multimedia - A Multidisciplinary Approach to Complex Issues, 2012.

[13] I. Daoudi, K. Idrissi and S.E.A Ouatik. Kernel region approximation blocks for indexing heterogonous databases. Multimedia and Expo, 2008 IEEE International Conference, 2008, pp. 1237-1240.

[14] S. Chafik, I. Daoudi, M.A EL Yacoubi, H. El Ouardi and B. Dorizzi. Locality sensitive hashing for content-based image retrieval: A comparative experimental study. The Fifth International Conference on Next Generation Networks and services (NGNs), 2014.

[15] A. Andoni and P. Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In 47th Annual IEEE Symposium on foundations of Computer Science, 2006, pp. 459-468.

[16] Paulevé, Loic, Hervé Jgou, and Laurent Amsaleg. "Locality sensitive hashing: A comparison of hash function types and querying mechanisms." Pattern Recognition Letters 31.11 (2010): 1348-1358.

[17] M.S. Charikar. Similarity estimation techniques from rounding algorithms. In Proceedings of the Symposium on Theory of Computing, 2002.

[18] Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn. Beyond locality-sensitive hashing. In SODA, pp 10181028, 2014.

[19] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. PVLDB, 5(5), pp: 430441, 2012.

[20] Weiss, Yair, Antonio Torralba, and Rob Fergus. "Spectral hashing." Advances in neural information processing systems. 2009.

[21] Liu, Wei, et al. "Hashing with graphs." Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011.

[22] Jin, Zhongming, et al. "Density sensitive hashing." Cybernetics, IEEE Transactions on 44.8 (2014): 1362-1371.

[23] Wang, Jingdong, et al. "Hashing for similarity search: A survey." arXiv preprint arXiv:1408.2927 (2014).

[24] Zhang, Wei, et al. "Data-oriented locality sensitive hashing." Proceedings of the international conference on Multimedia. ACM, 2010.

[25] Silva, Eliezer, et al. "Large-Scale Distributed Locality-Sensitive Hashing for General Metric Data." Similarity Search and Applications. Springer International Publishing, 2014. 82-93.

[26] Kang, Byungkon, and Kyomin Jung. "Robust and Efficient Locality Sensitive Hashing for Nearest Neighbor Search in Large Data Sets." NIPS Workshop on Big Learning (BigLearn), Lake Tahoe, Nevada. 2012.

[27] Jégou, Hervé, et al. "Query adaptative locality sensitive hashing." Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on. IEEE, 2008.

[28] X.-J. Wang, L. Zhang, F. Jing, and W.-Y. Ma. Annosearch: Image auto-annotation by search. In IEEE Conference on Computer Vision and Pattern Recognition, pages 14831490, 2006

[29] SUGAR, Catherine A. et JAMES, Gareth M. Finding the number of clusters in a dataset. Journal of the American Statistical Association, 2003, vol. 98, no 463.

[30] ARTHUR, David et VASSILVITSKII, Sergei. k-means++: The advantages of careful seeding. In : Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007. p. 1027-1035.

[31] BAHMANI, Bahman, MOSELEY, Benjamin, VATTANI, Andrea, et al. Scalable k-means++. Proceedings of the VLDB Endowment, 2012, vol. 5, no 7, p. 622-633.

[32] ARAI, Kohei et BARAKBAH, Ali Ridho. Hierarchical K-means: an algorithm for centroids initialization for K-means. Reports of the Faculty of Science and Engineering, 2007, vol. 36, no 1, p. 25-31.