IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 13, Issue 3 (Jul. - Aug. 2013), PP 43-52
www.iosrjournals.org

Improving search time for contentment based image retrieval via, LSH, MTRee, and EMD bounds

[1]Bahri abdelkhalak, [1]Hamid zouaki
[1]*Department of Mathematics and Computer Science, Faculty of Science, El jadida, Morocco*
[1]*Equipe : Modélisation Mathématiques et Informatique Décisionnelle*

ABSTRACT: *Comparison of images requires a distance metric that is sensitive to the spatial location of objects and features. The Earth Mover's Distance was introduced in Computer Vision to better approach human perceptual similarities. Its computation, however, is too complex for usage in interactive multimedia database scenarios. Nearest neighbor (NN) search in high dimensional space is an important problem in many applications, in particular if the method of similarity measure used be the EMD. Ideally, a practical solution (i) should be implementable in a relational database, and (ii) its query cost should grow sub-linearly with the dataset size, regardless of the data and query distributions. Despite the bulk of NN literature, no solution fulfills both requirements, except locality sensitive hashing (LSH). In this paper, we propose a new index structure, named LSH-LUBMTree, for efficient retrieval of multimedia objects. It combines the advantages of LSH [27,28], the technique used to embedding [33] the EMD, and the advantages, of LUBMTree [19]. Unlike the images of each bucket are stored in the LUBMTree.*
Experimental results show that LSH-LUBMTree performs better than the standard LSH in term of search time.
Key words: *CBIR, LSH, EMD, MTree, LUBMTree*

I. Introduction

Any image database, whether a newspaper photo archive, a repository for biomedical images, or a surveillance system, must be able to compare images in order to be more than an expensive file cabinet.

Content-based access is key to making use of large image databases, such as a collection of decades' worth of diverse photographs or the torrent of images from new, high-throughput microscopes [1]. Having a notion of distance is also necessary for global analyses of image collections, ranging from general-purpose techniques such as clustering and outlier detection to specialized machine learning applications that attempt to model biological processes. Comparing two images requires a feature extraction method and a distance metric. A feature is a compact representation of the contents of an image. The MPEG-7 standard [2] specifies a number of image features for visual browsing. A distance metric computes a scalar distance between two features: examples are the Euclidean (L2) distance, the Manhattan (L1) distance, and the Mahalanobis distance [3]. The choice of image feature and distance metric depends on the nature of the images, as well as the kind of similarity one hopes to capture. For many classes of images, the spatial location is important for whether two images should be considered similar. For example can be constructed from photographs or surveillance images where one wishes to discount small rotations or translations in defining image similarity. The earth mover's distance (EMD), first proposed by Werman et al. [6], captures the spatial aspect of the different features extracted from the images. The distance between two images measures both the distance in the feature space and the spatial distance.

The EMD has strong theoretical foundations [7] and is robust to small translations and rotations in an image. It is general and flexible, and can be tuned to be having like any Lp-norm with appropriate parameters. It has also been successfully applied to image retrieval based on contours [10] and texture [11], as well as similarity search on melodies [12], graphs [13], and vector fields [14].The complexity of the EMD algorithm is more than O(N3) [1], where N is the number of clusters in the signatures. Some derivation algorithms of the EMD have been proposed to reduce the time computation. Pele and Werman [16] proposed the fast algorithm of $\hat{EMD}$. Ling and Okada [15] proposed the fast algorithm of EMD_ L1. The $\hat{EMD}$ can be obtained by replacing the ground distance of the EMD with the threshold distance. EMD-L1 is a replacement of the ground distance with the L1 distance. Both $\hat{EMD}$ and EMD-L1 are different from the original EMD. They only indicate high-speed calculation methods in the case that the threshold distance and L1 distance were used as the ground distance.

The most common and straightforward method for solving exact NN problem is based on hierarchical space partitioning, resulting in various kinds of tree structure indexes [32], [29], [26], [30]. The multi-dimensional feature space is split into smaller partitions and organized as a tree structure. Data close to each other are grouped in the node so that they can be pruned together without accessing each individual point inside.

However, the pruning power of these indexes decreases as dimensionality grows and most of the tree nodes will be accessed, taking considerable CPU and I/O cost. In this case, the performance of existing index structures degrades rapidly and even becomes worse than a simple sequential scan of the data [24]. Due to this curse of dimensionality, it is difficult to build indexing support to efficiently answer exact NN queries. To provide efficient similarity search, the research community has focused on approximate NN search in recent years. Among various efforts, locality sensitive hashing (LSH) [28], [27] and its variants have received considerable attention. The LSH family adopts hash functions that preserve the distance in the Euclidean space so that similar objects have a high probability of colliding in the same bucket. If there are l hash tables and each table is associated with m hash functions, an object o will be hashed to H(o) = [h1, h2, ..., hl]. Given a query object q, the search space includes the buckets in the l hash tables where q is located. All the objects in these buckets are scanned to return the approximate NN result. As m increases, the bucket size becomes smaller and more false positives are removed. Precision increases but recall degrades. Similarly, as l increases, more buckets are examined. Recall is improved but precision may become worse. Thus, the main challenge of LSH is to tune a good tradeoff between precision and recall. To achieve a high search accuracy, hundreds of hash tables are normally used [27] and require a large amount of memory space. In [31], multi-probe LSH was proposed to reduce the number of hash tables and obtain the same search quality.

To support efficient NN query processing, we propose a novel index structure, named LSH-LUBMTree, based on the following two observations:

1) In LSH, the hash function is likely to place most of data objects into the buckets near the mean hash value, resulting in a skewed distribution of bucket size. When the size of table decreases, the bucket size increases, so the buckets recursively partitioned to reduce the number of false positives by using the index structure LUBMTree[19].
2) The search time decrease by using the range query algorithm in the LUBMTree (see listing 2) based of the bounds of EMD.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 Describe the EMD distance. Section 4 Describe the MTree and the index structures LUBMTree. Sections 5 present the LSH and our contribution. Section 6 evaluates the search in the proposed index structure experimentally; Finally Section 7 concludes the paper and the future work.

II. Related Work

LSH [28], [27] has been widely applied to answer the approximate NN query and shown to be quite effective for similarity search in multimedia databases including text data [21], audio data [22], images [5] and videos [9]. The query cost grows sub-linearly with the data set size in the worst case. However, it is a trivial job to tune a good tradeoff between the precision and recall. In practive, hundreds of hash tables have to be built for a high search accuracy [27]. To reduce the number of hash tables, Lv et al. proposed multi-probe LSH[31]. Tao et al. have recently [4] recently has proposed LSB-tree to address both the quality and the efficiency of multimedia retrieval. The hash values are represented as 1-dimensional Z-order values and indexed in the B+-tree. Multiple trees can be built to improve the result quality. Compared with existing LSH methods, LSH-LUBMTree only used the EMD, and those bounds, and store the elements of the buckets in the LUBMTree to accelerate the search time. On the other hand, LSh-LUBMTree takes advantage of the LSH and those of the LUBMTree .

III. The Earth Mover Distance

According to the classification result of pixel images, the signatures of images can be of different size, and hence the Euclidean distance, Bhattacharyya distance…etc are not feasible, hence the choice of EMD (Earth Mover Distance) [8] distance, and that one of the most efficient distance to compare sets of features, see e.g. [17, 18].

The EMD is based on the well-known transportation problem. Suppose some suppliers, each with a given amount of goods, are required to supply some consumers, each with a given limited capacity to accept goods. For each supplier-consumer pair, the cost of transporting a single unit of goods is given. The transportation problem is: Find a minimum expensive flow of goods from the suppliers to the consumers that satisfy the consumers' demand.

The Earth Mover's Distance [Rubner et al., 1998] is one of the distances that can work efficiently on signatures which are not necessarily the same number of bins.

The EMD is the minimum amount of work to change a signature into another. The concept of "work" is based on the quantity of the contents of the signature to be transported from one component to another and the distance chosen by the user to measure the distance between two components. The number of components to

compare two signatures may be different, as well as the sum total of the components of the two signatures. Two signatures are compared $P = \{(p_1, w_1), \cdots, (p_m, w_m)\}$ *and* $Q = \{(q_1, u_1), \ldots, (q_n, u_n)\}$

P is the first signature containing m clusters pi of weight wi, and Q the second signature containing n clusters qj of weight uj. According to [1], the first step is to find all content portions of fij to carry the signature of the component i to component j that minimize the cost (work) as follows: $\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}$

Where F={f_{ij}} denotes a set of flows. Each flow f_{ij} represents the amount transported from the i^{th} supply to the j-th demand.

D=[d_{ij}] the ground distance matrix, where d_{ij} is the ground distance between clusters pi and q_j with the following constraints:

$f_{ij} \geq 0, 1 \leq i \leq m, 1 \leq j \leq n$

$\sum_{j=1}^{n} f_{ij} \leq w_i, 1 \leq i \leq m$

$\sum_{i=1}^{m} f_{ij} \leq u_j, 1 \leq j \leq n$

$\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij} = \min(\sum_{i=1}^{m} w_i, \sum_{j=1}^{n} u_j)$

The first constraint only allows movement of components from P to Q.

The following two constraints limit the amount of displaced components of P, and quantity of components received by Q at their respective weights. The last constraint implies the maximum possible displacement of components.

EMD distance is then defined as

$$EMD(q, p) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} f_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}$$

EMD distance is normalized to the minimum sum of weights of signatures to avoid favoring smaller signatures when comparing signatures of different sizes.

IV. Metric Access Methods

Metric access methods (MAMs) [20] are index structures designed to perform efficiently similarity queries in metric spaces. They only use especially the triangle inequality, to filter out objects or entire regions of the space during the search, thus avoiding the sequential (or linear) scan over the database.

MAMs can be classified into two main groups: (1) Pivot based MAMs select from the database a number of pivot objects, and classify all the other objects according to their distance from the pivots (2) MAMs based on compact partitions divide the space into regions as compact as possible.

Each region stores a representative point (local pivot) and data that can be used to discard the entire region at query time, without computing the actual distance from the region objects to the query object. Each region can be partitioned recursively into more regions, inducing a search hierarchy.

1. M-Tree

A. Basic Principles

The M-tree [10] is a dynamic (meaning easily updatable) index structure that provides good performance in secondary memory. The M-tree is a hierarchical index, where some of the data points are selected as centers (local pivots) of regions and the rest of the objects are assigned to suitable regions in order to build up a balanced and compact hierarchy of data regions. Each region (branch of the tree) is indexed recursively. The data is stored in the leaves of the M-tree, where each leaf contains ground entries (grnd(Oi), Oi ∈ S). The internal nodes store routing entries (rout(Oi), Oi ∈ S).

The M-tree organizes the objects into fixed-size nodes. Each node can store up to M entries—this is the capacity of M-tree nodes. An entry for a routing object Or also include a pointer, denoted ptr($T(O_r)$), which references the root of a sub-tree, $T(O_r)$, called the covering tree of Or, a covering radius $r(O_r) > 0$, and $d(O_r, P(O_r))$, the distance to the parent object $P(O_r)$, i.e. the routing object which references the node where the O_r entry is stored.

entry (O_r) = [O_r, ptr($T(O_r)$), $r(O_r)$, $d(O_r, P(O_r))$]

For each (ground) DB object, one entry having the format

entry(O_j) = [O_j, oid(O_j), $d(O_j, P(O_j))$] is stored in a leaf node, where oid(O_j) is the identifier of the object, which is used to provide access to the whole object resident on a separate data file. Starting at the root level, a new object O_i is recursively inserted into the best subtree $T(O_j)$, which is defined as the one where the covering radius $r(O_{j)}$ must increase the least in order to cover the new object. In case of ties, the subtree whose

center is closest to Oi is selected. The insertion algorithm proceeds recursively until a leaf is reached and O_i is inserted into that leaf, at each level storing the distance to the routing object of its parent node (so-called to-parent distance). Node overflows are managed in a similar way as in the B-tree. If an insertion produces an overflow, two objects from the node are selected as new centers, the node is split, and the two new centers are promoted to the parent node. If the parent node overflows, the same split procedure is applied. If the root overflows, it is split and a new root is created. Thus, the M-tree is a balanced tree.

The semantics of the covering radius is the following: All the objects stored in the covering tree of Or are within the distance r(Or) from O_r, i.e. $\forall\ O_i \in T(O_r)$, $d(O_r, O_i) \leq r(O_r)$.

A routing object O_r, hence, defines a region in the metric space M, centered on O_r and with radius $r(O_r)$ (see Fig 1).

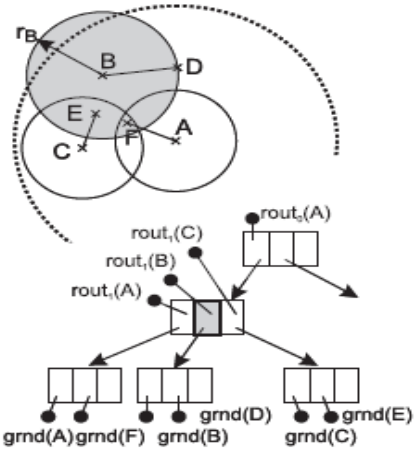

Fig 1 : Example on an MTree

The M-tree, therefore, organizes the space into a set of (possibly overlapping) regions, to which the same principle is recursively applied. The covering radius, $r(O_r)$, and the distance between the object and its parent object, $d(O_r, P(O_r))$, both stored in each entry of the tree, are used to "prune" the search space during the search phase

B. Searching the M-tree

Before presenting specific algorithms for building the M-tree, we show how the information stored in nodes is used for processing similarity queries. Clearly, the performance of search algorithms is largely influenced by the actual construction of the M-tree, even though the correctness and the logic of search are independent of such aspects.

In order to minimize both the number of accessed nodes and computed distances, all the information concerning (pre-computed) distances which are stored in the M-tree nodes, i.e. $d(O_i, P(O_i))$ and $r(O_i)$, is used to efficiently apply the triangle inequality property.

The query range (Q, r_Q) requests for all the database objects O_i , such that $d(O_i,Q) \leq r_Q$. Therefore, the algorithm rangeQuery has to follow all such paths in the M-tree which cannot be excluded from leading to objects satisfying the above inequality. Since the query threshold, r_Q, does not change during the search process, and provided the response set is required as a unit, the order with which nodes are visited has no effect on the performance of **rangeQuery** algorithm.

Since, when accessing node N, the distance between Q and Op, the parent object of N, has already been computed, it is possible, by applying the triangle inequality, to prune a whole sub-tree without computing any new distance at all. The condition for pruning is as follows.

Lemma 1 [25]

If $d(O_r,Q) > r_Q + r(O_r)$, then, for each object O_j in $T(O_r)$, it is $d(O_j,Q) > r_Q$. Thus,
T(Or) can be safely pruned from the search.

Proof: Since $d(O_i,O_r) \leq r(O_r)$ holds for any object O_j in $T(O_r)$, it is

$d(O_i,Q) \geq d(O_r,Q) - d(O_i,O_r)$ (triangle inequality)
$\geq d(O_r,Q) - r(O_r)$ (def. of covering radius)
$> r_Q$ (by hypothesis)

Range query algorithm in M-tree [25]

Algorithm 1. (Range query algorithm)

QueryResult **rangeQuery**(Node N, Query (Q,rQ))
{

```
    // if N is root then d(Or,Op)=d(Op,Q)=0
    Let P be the parent rooting object of N
     If  N  is not a leaf then {
          For  each root ( Or) in N do {
               If |d(Op,Q) -  d(Or,Op)| ≤ r(Or)+rQ    then
                    Compute  d(Or,Q)
                    If d(Or,Q) ≤ r(Or)+rQ       then
                         rangeQuery(ptr(T( Or)),(Q, rQ))
     }
} /* for each….*/
     }
     Else{
          For  each grnd  (Oj) in N do {
               If | d(Op,Q) -  d(Oj,Op)| ≤ rQ    then
                    Compute  d(Oj,Q)
                    If d(Oj,Q)≤  rQ   then
                         Add Oj to the query result
     }
} /* for each….*/
     }
} /* rangeQuery…*/
```

In order to apply Lemma 1, the distance $d(O_r,Q)$ has to be computed. This can be avoided by taking advantage of the following result.

Lemma 2 [25]

If $|d(O_p,Q) - d(O_r,O_p)| > r_Q + r(O_r)$, then $d(O_r,Q) > r_Q + r(O_r)$.

Proof: This is a direct consequence of the triangle inequality, which guarantees that both $d(O_r,Q) \geq d(O_p,Q) - d(O_r,O_p)$ and $d(O_r,Q) \geq d(O_r,O_p) - d(O_p,Q)$ hold (see Fig.2).

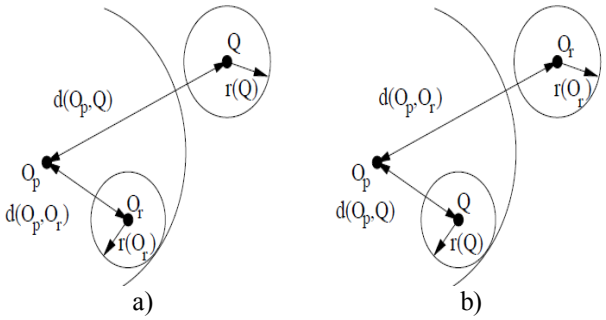

Fig 2: How Lemma 2 is used to avoid computing distances. Case a): $d(O_r,Q) \geq d(O_p,Q) - d(O_r,O_p) > r(Q)+r(O_r)$; Case b): $d(O_r,Q) \geq d(Or,O_p) - d(O_p,Q) > r(Q)+r(O_r)$. In both cases computing $d(O_r,Q)$ is not needed.

Range queries are implemented by traversing the tree, starting from the root. The nodes which parent region (described by the routing entry) is overlapped by the query ball are accessed (this requires a distance computation). As each node in the tree (except for the root) contains the distances from the routing/ground entries to the center of its parent node (the to-parent distances), some of the non-relevant branches can be further filtered out, without the need of a distance computation, thus avoiding the "more expensive" basic overlap check.

In the MTree structure, the objects in a single node may belong to different classes. During the search if the leaf node is selected, then the distances between the query object and all objects in it are calculated. And therefore we can have additional calculations between the query object and the objects that are not similar to this one. To reduce the number of calculations, the auteur [19] create one index structure called LUBMTree.

2. LUBMTree index structure

The LUBMTree is a metric access method, and it is an extension of MTree. Its goal is to accelerate the search of objects. By using our index structure LUBMTree, the original EMD can be calculated efficiently. The characteristic of the search in this structure is that it does not necessarily calculate all distances between different objects of the same node by using the EMD lower and upper bounds.

2.1 LUBMTree index structure

The LUBMTree structure has the same attributes as MTree, in addition two other attributes. The first attribute represent the lower bound of the EMD distance between the routing object and its parent. The second attribute represent the lower bound of the EMD distance of the covering radius of routing object.
The construction of the structure LUBMTree is built in the same way as MTree, while adding two attributes lower bounds, and two other attributes upper bounds for each object.

2.2 Searching in LUBMTree

Lemma 3[19]

Let P be the parent object of a data region (R, r). Let $d^{lb}(\cdot)$ and dub(.) are a lower and upper-bounding distance to EMD(·) respectively.

If $d(P,Q)-d_p^{ub} > r+\varepsilon_w \vee d_p^{lb} - d(P,Q) > r+\varepsilon_w$ Then the data region is not relevant to the query and can be filtered.

Proof [19]: This is a direct consequence of the triangle inequality, which guarantees that both $d^{ub}(R,Q) \geq r + \varepsilon w$ and $d^{lb}(R,Q) \geq r + \varepsilon_w$ hold (see Fig.3).

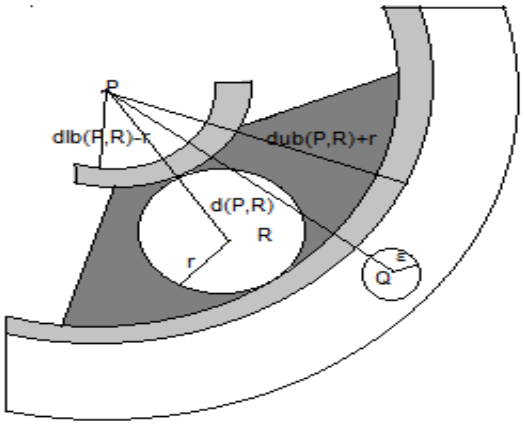

Fig. 3: Outer parent filtering in LUBMTree.

2.3 Similarity queries

Range query algorithm in LUBMTree *[19]*

Algorithm 2. (Range query algorithm)

```
QueryResult     rangeQueryLUB( Node N, RQuery (Q,ε))
{
    // if N is root then d(R,O)=d(P,Q)=0
    Let P be the parent rooting object of N
     Let's  d^lb(R,P)= be the lower bound of the EMD(R,P) distance
     If  N  is not a leaf then {
          For  each root ( R) in N do {
               If d(P,Q) -  d^ub(R,P) ≤ r+ε    and    d^lb(R,P) – d(P,Q) ≤ r+ ε   then {  // lemme 3
                         If d(P,Q) -  d(R,P) ≤ r +ε  and  d(R,P)  – d(P,Q) ≤ r + ε   then {  // lemme 2
                              Compute  d(R,Q)
                              If d(R,Q)≤ r_ub + ε   then    // lemme 1
                              rangeQueryLUB(ptr(T( R)),(Q, ε))
                         }
     }
} /* for each….*/
     }
     Else{
          For  each grnd  ( R) in N do {
                    If d(P,Q) -  d^ub(R,P) ≤ r+ε    and   d^lb(R,P) – d(P,Q) ≤ r+ ε   then {  // lemma 3
                         If d(P,Q) -  d(R,P) ≤ r +ε  and d(R,P)  – d(P,Q) ≤ r + ε   then {  // lemma 2
                              Compute  d(R,Q)
```

```
                    If d(R,Q)≤ ε then
                        Add R to the query result
                }
    }
} /* for each….*/
    }
} /* rangeQuery…*/
```

V. LOCALITY SENSITIVE HASHING

5.1 Details of hash function

Let h(o) be a hash function that maps a d-dimensional point o to a one-dimensional value. It is locality sensitive if the chance of mapping two points o1, o2 to the same value grows as their distance decreases. Formally:

Definition (LSH). Given a distance r, approximation ratio c, probability values p1 and p2 such that p1 > p2, a hash function h(.) is (r, cr, p1, p2) locality sensitive if it satisfies both conditions below:

1. *If* $\|o_1,o_2\| \le r$, *then* $\Pr[h(o_1)=h(o_2)] \ge p_1$;
2. *If* $\|o_1,o_2\| > cr$, *then* $\Pr[h(o_1)=h(o_2)] \le p_2$.

LSH functions are known for many distance metrics. For l_p norm, a popular LSH function is defined as follows [23]:

$$h(o)=\left\lfloor \frac{\vec{a}.\vec{o}+b}{w} \right\rfloor.$$

Here, $\vec{o}$ represents the d-dimensional vector representation of a point o; $\vec{a}$ is another d-dimensional vector where each component is drawn independently from a so-called p-stable distribution [23]; $\vec{a}\cdot\vec{o}$ denotes the dot product of these two vectors. w is a sufficiently large constant, and finally, b is uniformly drawn from [0,w). All L hash tables use the same primary hash function t1 (used to determine the index in the hash table) and the same secondary hash function t2. These two hash functions have the form

$$t_1(a_1,a_2,...,a_k)=((\sum_{i=1}^{k} r_i' a_i) \bmod P) \bmod tableSize \quad (1)$$

$$t_1(a_1,a_2,...,a_k)=((\sum_{i=1}^{k} r_i'' a_i) \bmod P) \quad (2)$$

Where ri' and ri'' are random integers, tableSize is the size of the hash tables, and P is a prime. In the current implementation, a_i are represented by 32-bit integers, and the prime P is equal to $2^{32}-5$. This value of the prime allows fast hash function computation without using modulo operations. If there are *l* hash tables, as *l* increases, more buckets are examined. Recall is improved but precision may become worse. As *tablesize* increases, the bucket size becomes smaller and more false positives are removed. Precision increases but recall degrades. Similarly, as *tablesize* decrease the bucket size becomes bigger and more true positives are retrieved, but search time is increases. To improve this time we propose a new index structure called LSH-LUBMTree that combines the technique LSH with the LUBMTree.

2 LSH-LUBMTree

In the firs we describe the embedding technique of EMD, that is used to building, and to searching in the LSH-LUBMTree.

2.1. The embedding [33]

We formally show how to construct an embedding of EMD into l_1 space.

Let P, Q be tow points sets of cardinality s, each in R^k and $V = P \cup Q$. for any pear $p \in P, q \in Q$, the weight of(p,q) is the Euclidean (l2) distance between p and q. Assume that the smallest inter-point distance is 1, and let D be the diameter of V. The embedding is defined as fellows. We impose grids of R^k of sides 1/2, 1, 2, 4, . . . , 2i, . . . Δ. Let Gi be grid of side 2^i. We impose the condition that the grid G_i is a refinement of grid Gi+1. Moreover, the grid is translated by a vector chosen uniformly at random from $[0, \Delta]^K$.

For each grid G_i, we construct a $v_i(p)$ with one coordinate par cell, where each coordinate counts the number of points in the corresponding cell. In other words, each vi(p) forms a histogram of P. we defined mapping f, by setting f(P) to be the vector $v_{-1}(p)/2$,v0(p),2v1(p),4v2(p), . . . ,2^ivi(p),. . . Note that v(P) lives in an $O(\Delta)^k$-dimensional space, but only O(log(Δ).|P|) entries in this vector are non-zero (i.e, the vector v(P) is sparse).

2.2 Building a LSH-LUBMTree

The construction of the LSH-LUBMTree is very simple. Within each embedding [33], the signature of each image is mapped into a series of vectors, one for each grid, and concatenated into one vector (d dimensional). For each hash table, in the first step we calculate the first index for each d-dimensional object, in the second step we calculate its second index, and we store them in the corresponding bucket. So each bucket is a LUBMTree[19] index structure (see listing 3).

Algorithm 3. (Algorithm preprocessing the index structure LSH-LUBMTree)

Algorithm Preprocessing
Input a set of points S (signatures), *l* (number of hash tables), size table, prime
Output Hash tables T_i, i=1,…,l
P←*embedding(s)*
For each *i=1,…,l*
Initialize hash table T_i by generating a random hash function $g_i(.)$
For each *i=1,…, l*
For each *j=1, …, n*
Ind1←$t1(g_i(p_j))$ // equation (1)
Ind2←$t2(g_i(p_j))$ // equation (2)
Str←find(Ti(ind1),ind2)// find in the list Ti(ind1) the bucket ind2
If(str==null)
Create new *LUBMTree and add them in the list* Ti(ind1)
endif
Store point p_j on the bucket ind2 (e.i in the *LUBMTree)*
End for
End for

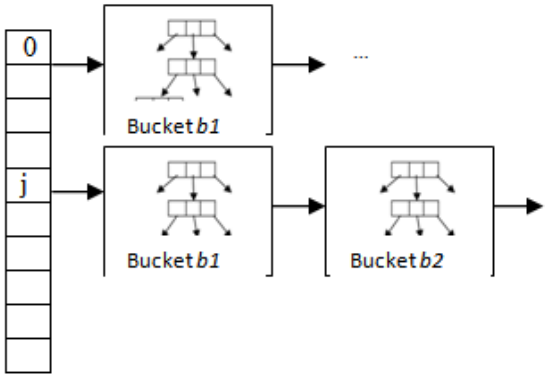

Table i *Fig.4: Schema of LSH-LUBMTree Structure (one table)*

2.3 Searching in LSH-LUBMTree

For each hash table, in the first step we calculate the first index for d-dimensional query, in the second step we calculate its second index to find the corresponding bucket, after this we start the search in the corresponding bucket (LUBMTree) by using the **rangeQueryLUB** (algorithm 2) algorithm (see algorithm 4).

Algorithm 4. (Approximate Nearest Neighbor query answering algorithm)

Algorithm Approximate Nearest Neighbor Query
Input a query point S_q, threshold of range query
Access To hash tables T_i, *i=1,…,l* generated by the preprocessing algorithm
Output nearest neighbors
q ← *Embedding*(S_q)
S←*Φ*
For each *i=1, …, l*
Ind1←$t1(g_i(q))$ // equation of (1)
Ind2←$t2(g_i(q))$ // equation of (2)
Str←find(Ti(ind1),ind2)// find in the list Ti(ind1) the bucket ind2

S←*S* U{points found in rangeQuery(str, threshold, q) by using the search algorithm in *LUBMTree*, e.i see algorithm 2) }
end for
Remove the duplicate and return ***S***

VI. Experimental Evaluation

We performed a number of range queries using 50 classes of the Coil-100 database, which each class contains 72 images. We use also the Wang database, composed by 10 classes, which each class contains 100 images.

All experiments were performed on a PC with a 5.6 Ghz processor and 4 Gb of main memory.
For all experiments we set the parameters k=6 and w=1. Moreover, we set the parameters of false negatives that we can tolerate to 10%. For this choice of parameters, l evaluated to 10.

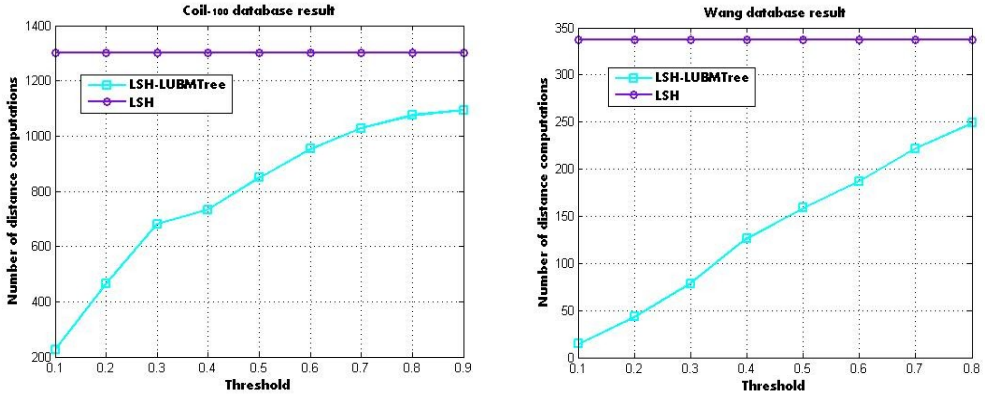

Fig. 5 .Number of distance computations calculate in the range query when searching in the LSH-LUBMTree compared by the search in the LSH

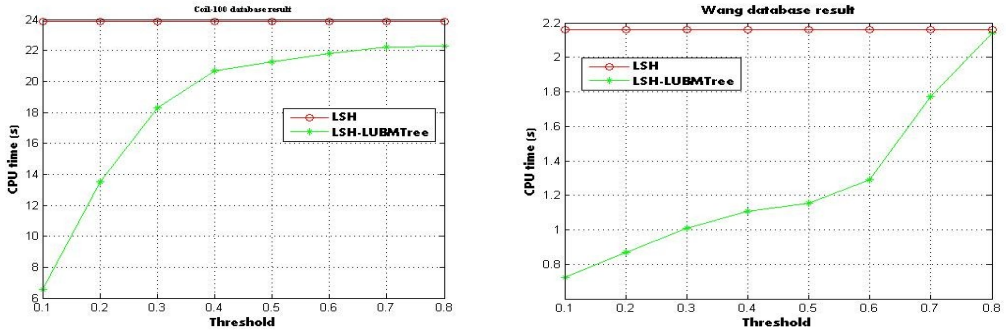

Fig. 6. Time computation calculates in the range query when searching in the LSH-LUBMTree, and in standard LSH.

In the figure 5, the search in the LSH produces more number of distance computations than the search in the LSH-LUBMTree for the some parameters, and so produces more image candidates. The figure 6, show the response time obtained when search in the LSH and in our index structure. We see that the search in our structure is faster than the search in the LSH, because the number of the calculated distances in the search algorithm in our structure is less than the one of the LSH. This improvement is explain by the search algorithm in the proposed index uses the lower and the upper bounding to skipping the supplementary calculs.

Since the index structure LSH-LUBMTree that combines the structure LSH with LUBMTree show its efficiency in term of search time than LSH. For example for the search in coil database, the search time in the index LSH-LUBMTree in the range 0.1 is 6.5s, but the one in the same range for the LSH is 23.8s.

VII. Conclusion

Search in large image or other multimedia databases highly dependents on the underlying similarity model. The Earth Mover's Distance, proposed in Computer Vision literature, is an interesting new approach towards achieving high-quality content-based retrieval. Despite its advantages, this distance measure is computed via a linear programming algorithm which is too slow for today's huge and interactive multimedia.

The disadvantage of the EMD metric is the significant response time. To solve this problem, we have used the index structure LUBMTree.

To improve more the search time, we have proposed a new index structure called LSH-LUBMTree that combines the advantages of LSH, the technique used to embedding the color signature (Indyk and all 2003), and

the advantages of LUBMTree. The experimentation shows that the search in the LSH-LUBMTree, become more efficiency in the search time than the standard LSH.

In the feature work, we plan to adapt the signatures that proposed by Rubner, to be supported by the LSH structure.

References

[1] Swedlow, J.R., Goldberg, I., Brauner, E., Sorger, P.K.: Informatics and quantitative analysis in biological imaging. Science 300 (2003) 100–102

[2] Manjunath, B.S., Salembier, P., Sikora, T., eds.: Introduction to MPEG 7: Multimedia Content Description Language. Wiley (2002)

[3] Mahalanobis, P.: On the generalised distance in statistics. Proc. Nat. Inst. Sci. India 12 (1936) 49–55

[4] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data, pages 563–576, New York, NY, USA, 2009. ACM.

[5] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In In ACM Multimedia, pages 869–876, 2004

[6] Werman, M., Peleg, S., Rosenfeld, A.: A distance metric for multi-dimensional histograms. Computer, Vision, Graphics, and Image Proc. 32 (1985) 328–336

[7] Peleg, S., Werman, M., Rom, H.: A unified approach to the change of resolution: Space and gray-level. IEEE Trans. PAMI 11 (1989) 739–719

[8] Y. Rubner and C. Tomasi, Perceptual Metrics for Image Databases Navigation, Kluwer Academic Publishers, Dordrecht, The Ntherlands, 2001

[9] W. Dong, Z. Wang, M. Charikar, and K. Li. Efficiently matching sets of features with random histograms. In MM '08: Proceeding of the 16th ACM international conference on Multimedia, pages 179–188, New York, NY, USA, 2008. ACM.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In Proc. 23rd Conference on Very Large Databases (VLDB'97), pages 196–435. Morgan Kaufmann, 1997.

[11] Lazebnik, S., Schmid, C., Ponce, J.: Sparse texture representation using affineinvariant neighborhoods. In: Proc. CVPR. (2003)

[12] Typke, R., Veltkamp, R., Wiering, F.: Searching notated polyphonic music using transportation distances. In: Proc. Int. Conf. Multimedia. (2004) 128–135

[13] Demirci, M.F., Shokoufandeh, A., Dickinson, S., Keselman, Y., Bretzner, L.: Manyto-many feature matching using spherical coding of directed graphs. In: Proc. European Conf. Computer Vision (ECCV). (2004)

[14] Lavin, Y., Batra, R., Hesselink, L.: Feature comparisons of vector fields using earth mover's distance. In: Proc. of the Conference on Visualization. (1998) 103–109

[15] H. Ling and K. Okada. EMD-L1 : An Efficient and Robust Al-gorithm for Comparing Histogram-Based Descriptors. ECCV, p. 330-343, 2006.

[16] O. Pele and M. Werman. Fast and robust earth mover's distances. In ICCV, 2009.

[17] Y. Liu, D. Zhang, G. Lu, W.-Y. Ma, "Region-based image retrieval with high-level semantic color names," IEEE Int. Multimedia Modelling Conference, pp. 180–187, 2005.

[18] G. Dvir, H. Greenspan, Y. Rubner, "Context-Based image modelling," ICPR, pp. 162–165, 2002.

[19] A. Bahri and H. Zouaki. Fast Retrieval Algorithm Using EMD Lower and Upper Bounds and a Search Algorithm in multidimensional index

[20] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroqu´ın. Searching in metric spaces. ACM Computing Surveys, 33(3):273–321, 2001.

[21] B. Stein. Principles of hash-based text retrieval. In SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 527–534, New York, NY, USA, 2007. ACM.

[22] R. Cai, C. Zhang, L. Zhang, and W.-Y. Ma. Scalable music recommendation by search. In MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia, pages 1065–1074, New York, NY, USA, 2007. ACM.

[23] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In SoCG, pages 253–262, 2004.

[24] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[25] Tomáš Skopal Ph.D. Thesis, " Metric Indexing in Information Retrieval", 2004

[26] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree: An index structure for high-dimensional data. In VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[27] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[28] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pages 604–613, New York, NY, USA, 1998. ACM.

[29] N. Katayama and S. Satoh. The sr-tree: an index structure for high dimensional nearest neighbor queries. In SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pages 369–380, New York, NY, USA, 1997. ACM.

[30] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: an index structure for high-dimensional data. Technical report, College Park, MD, USA, 1994.

[31] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In Proceedings of the 33rd international conference on Very large data bases, VLDB '07, pages 950–961. VLDB Endowment, 2007.

[32] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data, pages 10–18, New York, NY, USA, 1981. ACM.

[33] P. Indyk and N. Thaper. Fast image retrieval via embeddings. In 3rd International Workshop on Statistical and Computational Theories of Vision (at ICCV), 2003.