Evaluation of strategies for multiple sphere queries with local image descriptors

Nouha Bouteldja, Valérie Gouet-Brunet and Michel Scholl

CEDRIC/CNAM - 292, rue Saint-Martin - F75141 Paris Cedex 03

{nouha.bouteldja,valerie.gouet,scholl}@cnam.fr

ABSTRACT

In this paper, we are interested in the fast retrieval, in a large collection of points in high-dimensional space, of points close to a set of m query points (*a multiple query*): we want to efficiently find the sequence $A_{i,i \in \{1,m\}}$ where A_i is the set of points within a sphere of center query point $p_{i,i \in \{1,m\}}$ and radius ϵ (*a sphere query*). It has been argued that beyond a rather small dimension ($d \geq 10$) for such sphere queries as well as for other similarity queries, sequentially scanning the collection of points is faster than crossing a tree structure indexing the collection (the so-called curse of dimensionality phenomenon). Our first contribution is to experimentally assess whether the curse of dimensionality is reached with various points distributions. We compare the performance of a single sphere query when the collection is indexed by a tree structure (an SR-tree in our experiments) to that of a sequential scan. The second objective of this paper is to propose and evaluate several algorithms for multiple queries in a collection of points indexed by a tree structure. We compare the performance of these algorithms to that of a naive one consisting in sequentially running the m queries. This study is applied to content-based image retrieval where images are described by local descriptors based on points of interest. Such descriptors involve a relatively small dimension (8 to 30) justifying that the collection of points be indexed by a tree structure; similarity search with local descriptors implies multiple sphere queries that are usually time expensive, justifying the proposal of new strategies.

Keywords: CBIR, Local Descriptors, Multidimensional Indexing, Multiple Queries.

1. INTRODUCTION

During the last decade, similarity search has drawn considerable attention in multimedia systems, decision support and data mining. In these systems, there is the need to find a small set of objects which are similar to a given query object. Content based image retrieval is one example application where the objects to be retrieved are images, videos or parts of images. Usually, similarity is not measured on the objects themselves but on object abstractions called *features* which are points in some high-dimensional space. The number of dimensions may vary from moderate (8 to 30) to large (64 to 300) or over 900 in some applications such as astronomy. The more pertinent the object abstraction or *description*, the better the estimation of the similarity of two objects by the similarity of the features extracted from the objects. The similarity of two features (points) is a function of their distance in the high-dimensional space. Efficient similarity search is supported by a multidimensional index structure.

In this paper we are interested in the fast retrieval, in a large collection of points in a high-dimensional space, of points close to a set of m query points (a multiple query). We want to efficiently find the sequence $A_{i,i \in \{1,m\}}$ where A_i is the set of points within a sphere of center query point $p_{i,i \in \{1,m\}}$ and radius ϵ (ϵ-sphere). It has been argued that beyond a rather small dimension ($d \geq 10$) for such sphere queries as well as for k-NN queries, sequentially scanning the collection of points is faster than crossing the tree-structured multidimensional index (*curse of dimensionality*[1]). The more uniform the point distribution in space, the smaller such a threshold dimension. The first contribution of this paper is to experimentally assess whether the curse of dimensionality is reached with various points distributions when the number of dimensions is moderate. Considerable attention has been devoted to data structures that delay the curse of dimensionality but surprisingly no study experimentally evaluates until which dimension a tree structure is still appropriate. We compare the performance of a single sphere query when the collection is indexed by a tree structure (an SR-tree in our experiments) to that of a sequential scan. For this we take as an example of real data, the points obtained by the description of the visual content of images. We compare the performance to that obtained with various synthetic distributions.

The second and main objective of this paper is to evaluate several algorithms for multiple queries in a collection of points indexed by a tree structure. We compare their performance to that of a naive one consisting in sequentially running the m queries (sequentially finding the A_i sets of answer points). Similarity search in a collection of images where each image (as well as each query) is described by a set of local descriptors is one example of application characterized by a moderate dimension (8 to 30) justifying that the collection of points be indexed by a tree structure. The first improvement aims at saving I/O by avoiding scanning a node several times for different queries points. The second improvement attempts to save CPU time by avoiding some distance computations. This solution uses the triangle inequality applicable to a metric space. We rely on five lemmas among which three are novel contributions.

Related work

To our knowledge, Braunmüller et al.[2] were the first to study multiple similarity queries for mining in metric databases. They propose a general framework for answering simultaneously a set of similarity queries (k-NN and sphere queries) and give algorithms both for reducing I/O cost and CPU time. An experiment is provided for multiple k-NN queries that evaluates the speed-up for a linear scan of the collection of points as well as for an X-tree traversal. It is shown that parallelization yields an additional significant speed up. They propose some CPU saving by using the triangle inequality and two lemmas. For the X-tree, the saving is said to be of the order of 2 for two databases. Our work uses similar ideas for reducing I/O cost and CPU time but is applied to sphere queries instead of k-NN queries. For improving CPU time we also use the distance triangle inequality relying on the lemmas proposed by Braunmüller and al.[2] as well on three other novel lemmas. However, our extensive evaluation of the impact of applying the triangle inequality on the saving of expensive distance computations, does not lead to the same conclusions (see section 4.2). Last while Braunmüller et al.[2] propose an incremental processing of a multiple query that allows to provide partial answers to the user at an early stage of the evaluation, our algorithm performs the usual single depth first traversal of the tree. Each relevant node is accessed only once in both algorithms.

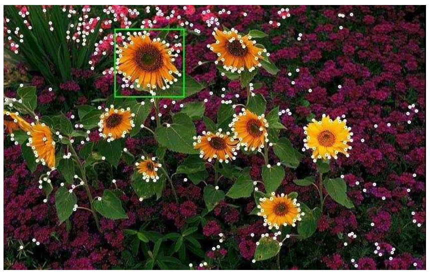
The rest of this paper is organized as follows: sections 2 and 3 respectively describe the feature extraction and the algorithms for single and multiple sphere queries. A database of images is taken as an example of real data and the description of images by content with emphasis on image description with *local descriptors* is studied (see section 2). Several tree structures have been proposed in the past decade as an improvement of the R*tree for multidimensional indexing.[3] Section 3, justifies our choice of the SR-tree,[4] and gives an algorithm for the sphere query. Then section 3.2 presents several strategies for multiple sphere queries. Section 4 describes our experiments and provides some evaluation results for I/O and CPU time saving.

2. IMAGE RETRIEVAL WITH LOCAL DESCRIPTORS

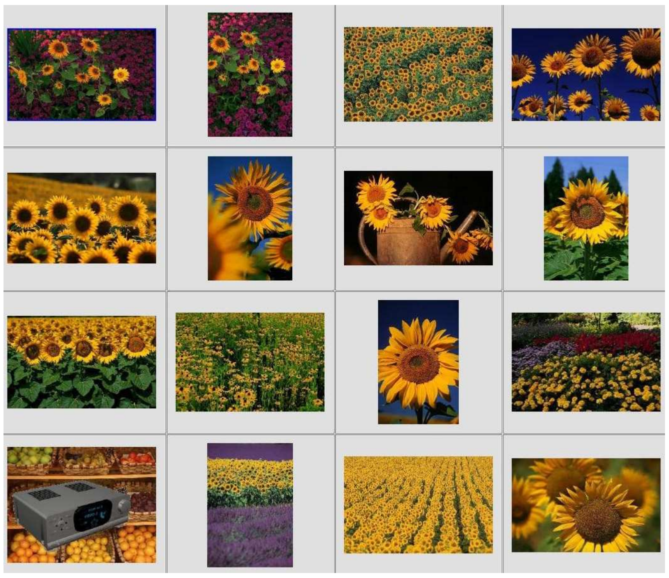
Content-based image descriptors mainly depend on the applications and datasets of images of interest. When considering *global* or *approximate* image retrieval, most of the descriptors are based on the global color, texture and shape of the image. Object or sub-image retrieval involves *local* descriptors (see figure 1 for an example of object retrieval). In this paper, we focus on such descriptors, because similarity search with local descriptors implies multiple sphere queries that are usually time consuming, justifying our study.

Local descriptors are usually based on *points of interest* extraction and local characterization. Such an approach allows to gain robustness against occlusions and cluttering since only a local description of the patch of interest is involved. Many techniques of points of interest[5–7] and performance evaluations[8] have been proposed in the literature of Computer Vision and CBIR. They depend on the application and more precisely on the image transformations authorized.

Image retrieval based on local descriptors follows a classical computation flow: first a robust salient feature detector is designed. Such a detector determines the location of salient points in the image, plus the support regions around them where the local descriptor is to be computed. Second, a rich and compact descriptor is computed for each salient point, by analyzing the signal within the support region exhibited.



(a) Query support



(b) Responses

Figure 1. *Example of object retrieval using local descriptors. (a) is the query support presented with extracted points of interest (in white). The query is a sunflower that has been manually determined by the user (bounding rectangle). (b) shows the 16 first retrieved images from a data set of generic images, presented by decreasing order of similarity. These results were obtained with IKONA, the CBIR platform developed at INRIA (http://www-rocq.inria.fr/imedia/ikona.html).*

A typology of feature spaces

When considering similarity search, local approaches differ from global ones by the *dimensionality* of the feature space involved: the content of the image is represented by *several vectors* in a *relatively low-dimensional (moderate) feature space*, usually having between 8 and 30 dimensions*. In comparison, global descriptors involve only *one vector* in a *high-dimensional space*. Characteristics of feature spaces for some global and local descriptors are resumed in table 1. The size (number of vectors per image) and the dimension given are representative of the descriptors considered.

Descriptors	Category	Size	Dim.
Color histogram	global	1	>200
Gabor filters	global	1	∼48
Regions of interest	local	4-20	∼20
Points of interest	local	100-500	8-30

Table 1. *Examples of image descriptors.*

The voting algorithm

Image retrieval with local descriptors implies a specific algorithm that proceeds in several steps:

1. For each query point $p_{i,i\in\{1,m\}}$, find in the feature space a set of neighbors points called $N(p_i)$;
2. For each image I_j of the database, compute a vote v_j, that is a function of the number of neighbors retrieved and belonging to I_j and of their associated distances. A similar image is associated with a vote large enough that expresses that it contains a lot of points close to most of the query points;
3. Finally, the images are usually sorted by decreasing order of their vote, as illustrated in figure 1.

Finding the nearest neighbors $N(p_i)$ of step #1 can be done according to two strategies.[9] Either a k*-NN search*; the advantage is that k can be easy to determine but the drawback is that some of the retrieved images may not be similar to the query image. Or an ϵ*-sphere query*, where ϵ can be estimated of fixed by the user; generally here the difficulty lies in determining ϵ.

*Except the SIFT descriptor[6] that involves 128 components, invariant to several image transformations.

Algorithm 1: S(T,q)

```
Input  : T a tree root, q a sphere query (sphere with center p and radius $\epsilon$)
Output: V a set of answers.
// Computes the set V of vectors of tree with rootpage T inside the sphere q
V:= empty;
readpage T;
if T is not a leaf then
    for each child C in T do
        if C.R intersects q and C.S intersects q then
            // C.R (C.S) minimum bounding rectangle (sphere) of C
            V+= {S(C, q)};
else
    for each point t in T do
        if t in q then then V+={t};
return V;
```

With local descriptors, the strategy usually follows an ϵ-sphere query. The k-NN strategy is not well suited because k may be very different for each query point p_i and not easy to determine intuitively as for global descriptors. If ϵ is known or estimated, the set $N(p_i)$ is defined by: $N(p_i) = \{n_k \in V_d \ / \ dist(p_i, n_k) < \epsilon\}$ where $dist$ is the similarity measure associated with the feature space. Traditionally for local descriptors, the similarity measure associated is L_2 or more generally the Mahalanobis distance δ_2. In the rest of the paper, q_i will represent the sphere query of center p_i and radius ϵ, and $d_{p_i p_j}$ the distance between two points p_i and p_j.

3. STRATEGIES FOR SIMILARITY SEARCH

We want to check that the gain of a tree traversal over a sequential scan is important for the applications we have in mind: in other words, we want to check that the performance does not suffer from the curse of dimensionality phenomenon .[1] Several index structures have been proposed in the literature of databases, see for example the survey proposed by Böhm and al..[9] We have chosen the SR-tree,[4] because its code[†] was available on the line at the time when the experiment started. On top of this code, we developed code for sphere queries. Although some other tree-structures such as the X-tree[10] or the Pyramid-tree[11] might slightly outperform the SR-tree, the latter scheme performance is by far better than that of the regular R*tree.[3]

In the next section, we give the algorithm for single sphere queries with an SR-tree. In section 3.2, we present several strategies for multiple sphere queries. The proposed or customized strategies have been implemented with an SR-tree, but are easily adaptable to other tree structures.

3.1. SR-Tree for single sphere queries

The structure of the SR-tree is a mixture of that of the R*tree[3] and that of the SS-tree .[12] It uses bounding regions wich are specified by the intersection of spheres and rectangles. For a detailed construction of the tree, see Katayama and Satoh's paper.[4]

We have adapted the SR-tree code for sphere queries. The approach, denoted by $S(T, q)$, is described in Algorithm 1. T is the tree root page and q is a sphere query (a sphere with center p and radius ϵ). Similar to the R*tree one, it consists of a straightforward depth-first traversal of the tree.

[†] *http : //research.nii.ac.jp/ katayama/homepage/research/srtree/*

Algorithm 2: SM(T,Q)

```
Input  : T a tree root, Q a set of sphere queries $< q_1, q_2, ..., q_m >$
Output: V a sequence of sets of vectors $< V_1, V_2, ..., V_m >$
for i in 1..m do  $V_i$:= S(T,$q_i$);
return V;
```

Algorithm 3: SM(T,Q)

```
Input  : T a tree root, Q a set of sphere queries
Output: V a set of vectors $< V_1, V_2, ..., V_m >$
readpage T;
if T is not a leaf then
    for each child C in T do
        $Q_c$:= empty;
        for each q in Q do
            if C.R intersects q and C.S intersects q then $Q_c$+= {q};
        if $Q_c$ not empty then V+= {SM(C, $Q_c$)};
else
    for each point t in T do
        for each q in Q do
            if t in q then $V_q$+={t};
return V;
```

3.2. Multiple sphere queries

Let T be a tree structure, and $Q = \{q_1, q_2, ..., q_m\}$ be a multiple sphere query (a set of sphere queries). The algorithm for a multiple sphere query, called SM(T,Q), returns a sequence of answers $< V_1, V_2, ..., V_m >$, where V_i is the set of points, answer to $S(T, q_i)$.

We give three strategies for answering such a query and evaluate their performance in the three following sections. In our experiments, T is an SR-tree structure, but it is important to note that the proposed or customized strategies are easily adaptable to other tree structures.

3.2.1. Strategy #1

The first and simplest strategy consists in sequentially evaluating Algorithm 1 for each of the queries in Q. It is described in Algorithm 2.

3.2.2. Strategy #2

The second strategy (see Algorithm 3) saves I/O time by reading a single page (node or leaf of the tree) only once for the whole set of queries. Sphere (rectangle) to sphere intersection tests as well as point in sphere inclusion test imply time consuming distance computations, where the distance function is the one of section 2.

3.2.3. Strategy #3

The third strategy is described in Algorithm 4. It attempts to save CPU time by avoiding distance computations as much as possible in applying the distance triangle inequality. It supposes that, prior to the sphere query, the distances $d_{p_i p_j}$ between any two points p_i and p_j, in query Q, have been precomputed. Such a strategy is motivated by the fact that tests implied by the triangle inequality are significantly cheaper than computing a distance.

Algorithm 4: SM(T,Q)

```
Input  : T a tree root, Q a set of sphere queries
Output: V = $< V_1, V_2, ..., V_m >$
readpage T;
if T is not a leaf then
    for each child C in T do
        $Q_c$:= empty;
        Q':= empty;
        for each q in Q do
            if Avoidable (q, Q', C) = 0 then
                if C.R intersects q and C.S intersects q then
                    $Q_c$+= {q}; Q'+={q};
            else if Avoidable (q, Q', C) = 3 then $Q_c$+= {q};
            if $Q_c$ not empty then V+= {SM(C, $Q_c$)};
else
    for each point t in T do
        Q':= empty;
        for each q in Q do
            if Avoidable (q, Q', t) = 0 then
                if t in q then
                    $V_q$+={t}; Q'+={q};
            else if Avoidable (q, Q', t) = 3 then $V_q$+={t};
return V;
```

Function `Avoidable(`*q, Q', o*`)`

```
Input  : q a sphere query, Q' a set of sphere queries, o a region or a point
Output: if lemma i is satisfied by q' of Q' then return i, otherwise return 0
for each q' in Q' do
    if lemma1(q', q, o) then return 1;
    else if lemma2(q', q, o) then return 2;
    else if lemma3(q', q, o) then return 3;
return 0;
```

One of the following 5 lemmas can be used for this third strategy. Algorithm 4 simultaneously uses the three first lemmas. Lemmas 1 and 2 were already proposed by Braunmüller and al.[2] Lemmas 2a, 3 and 3a are new contributions. Here, q_1 and q_2 are two queries (of centers p_1 and p_2) and o represents a node or a point in a leaf.

Lemma 1 (see Braunmüller and al.'s paper [2] and Figure 2)

If $dmin_{op_1} > d_{p_1p_2} + \epsilon$ holds, then $dmin_{op_2} > \epsilon$ holds.

where $d_{p_1p_2}$ is the distance between p_1 and p_2 (see section 2). For an SR-tree, the distance *dmin* between a region o and a point p_i is defined as the maximum of (1) the minimal distance between p_i and the bounding sphere of o and (2) the minimal distance between p_i and the bounding rectangle of o. If o is a point, *dmin* represents the distance inter-points.

Lemma 2 (see Braunmüller and al.'s paper [2] and Figure 2)

If $d_{p_1p_2} > dmax_{op_1} + \epsilon$ holds, then $dmin_{op_2} > \epsilon$ holds.

Here, if o is an SR-tree region, $dmax$ stands for the maximum distance between p_i and the bounding sphere of o, i.e. the distance to the sphere center plus the sphere radius. If o is a point, $dmax$ represents the distance inter-points.

Lemma 3 (see Figure 2)

If $dmin_{op_1} \leq \epsilon - d_{p_1p_2}$ holds then $dmin_{op_2} \leq \epsilon$.
Proof: immediate.

Function *Avoidable* in algorithm 4 checks whether for a couple of query points (q_1,q_2), an object (a point or an tree region) lies in a given region. Algorithm 4 can be improved (Algorithm 4a not shown here) by using the following lemmas 2a and 3a, in addition to the other lemmas. Lemma 2a (lemma 3a) takes advantage of a successful test with lemma 2 (lemma 3) on a couple (q_1,q_2), for other couples (q_1,q_3) (for loop inside function *Avoidable*).

Lemma 2a

Let q_1, q_2, q_3 be three query points and o a region (point).

If $(d_{p_1p_2} > dmax_{op_1} + \epsilon)$ (lemma 2 holds for q_2 wrt q_1) and $(d_{p_1p_2} \leq d_{p_1p_3})$ holds, then $dmin_{op_3} > \epsilon$ holds.
Proof: From $d_{p_1p_2} \leq d_{p_1p_3}$ and lemma 2 premise, by transitivity we have $d_{p_1p_3} > dmax_{op_1} + \epsilon$. Then by lemma 2, we have $dmin_{op_3} > \epsilon$.

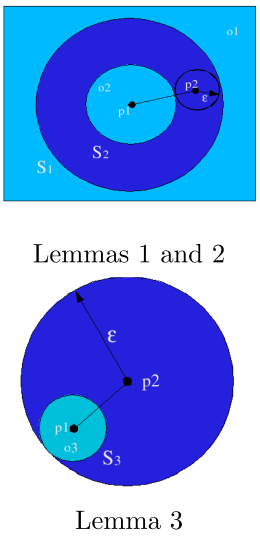


Figure 2. *Illustration of lemmas 1, 2 and 3. Lemma 1 is satisfied when objects (o_1) are outside sphere S_1. Lemma 2 is satisfied with objects (o_2) inside sphere S_2 while lemma 3 defines objects inside sphere S_3.*

Lemma 3a

If $(dmin_{op_1} \leq \epsilon - d_{p_1p_2})$ (lemma 3 holds for q_2 wrt to q_1) and $(d_{p_1p_2} \geq d_{p_1p_3})$ holds then $dmin_{op_3} \leq \epsilon$ holds.
Proof: similar to that of lemma 2a.

4. EXPERIMENTATIONS

We performed an extensive experimental evaluation to evaluate first the performance of SR-trees for single sphere queries with local descriptors, second to compare the various strategies for multiple queries, using synthetic and real data as well as synthetic and real query samples. The tested distributions are presented in the first subsection. All experiments were performed on an Intel Pentium (3.2 GHz) based workstation running Linux with 1Gbytes of main memory. The algorithms and data structures were implemented in C++.

We used the SR-tree code version 2.0 beta 5 provided by the authors and retained the default parameters. The tree is built dynamically and we stress that the index as well as the data are not loaded in main memory: each index node is read from the disk, when necessary.

Tested datasets

The experiments were done over 15 different datasets, split into three categories. For each dataset, approximately 218000 points were generated for three different dimensions, respectively 8, 17 and 29 that are representative of local descriptors (see Table 1). More precisely:

- The first category (D_R) corresponds to real points of interest, extracted from the well-known database of images COIL-100 .[13] This database contains 7200 color images representing 100 objects displayed under 72 different viewpoints. Each extracted point is characterized with three different local descriptors, providing three feature spaces of dimensions 8, 17 and 29 (resp. D_R^8, D_R^{17}, D_R^{29});
- In the second category (D_U), synthetic point coordinates are uniformly generated in the interval $[0, 1]$. Three datasets were obtained for the same dimensions (D_U^8, D_U^{17}, D_U^{29});

- The last category represents synthetic clustered datasets obtained for the same three dimensions with different shapes. Each one has 312 clusters of 700 points each, with a randomly generated center in $[0,1]^d$. Points in a cluster follow a gaussian distribution with respective values of $\sigma = 0.01$ (D^8_{C1}, D^{17}_{C1} and D^{29}_{C1}), $\sigma = 0.05$ (D^8_{C2}, D^{17}_{C2} and D^{29}_{C2}) and $\sigma = 0.2$ (D^8_{C3}, D^{17}_{C3} and D^{29}_{C3}).

In the following experiments, only the results associated with the first 100 neighbors for D_R, 700 neighbors for D_{C2} and the complete dataset for D_U are presented, because in the real dataset each point of interest can have at most 72 neighbors (each object is viewed according to 72 viewpoints). For D_{C2}, 700 is the number of points in each cluster; the results of the other clustered distributions are not shown, because of space limitations, but we briefly discuss differences with D_{C2} if any.

4.1. Single sphere queries

In this section, we present the speed up obtained with an SR-tree for single queries. For each dataset, a sample of 500 points was randomly chosen among the points in the data set. For the COIL-100 set, the 500 points correspond to different images. Each performance measure that will be presented corresponds to an average over this sample.

Speed-up SR-tree/Sequential

This experiment compares the performance of sphere queries with an SR-tree to that of a sequential scan and is illustrated in Figure 3. We plot the ratio of the sequential scan CPU time over the CPU time obtained by the SR-tree traversal versus the number of neighbors for D_R and D_{C2}.

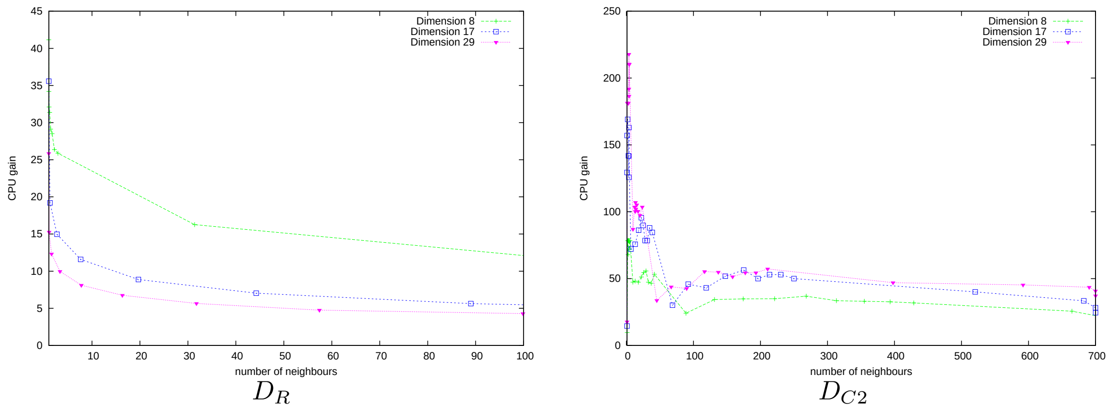


Figure 3. *Sequential scan/SR-Tree gain versus the number of neighbors for the datasets D_R and D_{C2}.*

Clearly, for the real dataset as well as for the clustered distribution, the curse of dimensionality is not reached! The speed up for dimension $d = 29$ is significant even for large values of the number of neighbors. In contrast, with the uniform data set (not shown here), there is already no gain from two retrieved neighbors and a dimension of $d = 17$. This trend is confirmed by the study of the ratio of nodes acceded during the tree traversal. This ratio remains small ($\leq 27\%$) for the D_R and D_{C2} distributions.

4.2. Multiple sphere queries

This section evaluates the strategies for multiple queries presented in section 3.2.

To have a realistic idea of the number m of multiple sphere queries $Q = \{q_1, ..., q_m\}$, we made queries on heterogeneous databases where images have been indexed with roughly 300 local descriptors. Objects of interest were manually chosen via a CBIR interface (as in Figure 1). For 50 queries on very different objects or sub-images providing good retrieval results, we have observed that the average size m is 36.56, with a standard deviation of 17.25. According to these considerations, m is chosen in the set $\{20, 50, 80\}$ for all the experiments.

4.2.1. I/O reduction

Here, we evaluate strategy #2 presented in section 3.2.2, by comparing it to the sequential strategy #1 (section 3.2.1). The I/O cost is estimated by the number of nodes acceded upon the tree traversal. The queries are generated over the sample of 500 points and the average is taken over $500/m$ multiple queries each of them having m points.

In Figure 4, we plot the gain g in acceded nodes versus the number of neighbors found for various sizes m. g is the ratio of the average number of nodes acceded for a single query times m (algorithm 2) over the average number of nodes acceded by algorithm 3. Note that for all values of m, the larger the dimension, the higher the gain. We restrict our experiment report to the real dataset, the clustered distributions yield similar results.

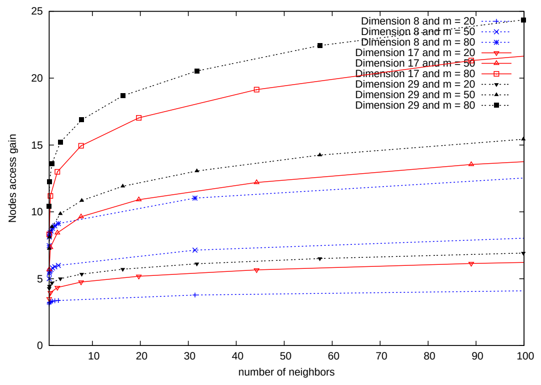


Figure 4. *Nodes' access gain for m = 20, 50, 80.*

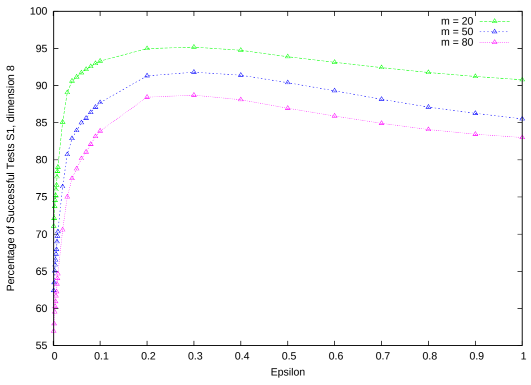


Figure 5. *Successful Tests' Average according to ϵ for different values of m.*

4.2.2. CPU time saving

In this section, we evaluate strategy #3 presented in section 3.2.3. The objective is to assess whether one can save some CPU time by avoiding some distance computations in adding the triangular inequality tests presented in section 3.2.3 (algorithm 4 with lemmas 1, 2, 3 and algorithm 4a with lemmas 1, 2a, 3a).

Tested queries

In contrast to single point queries, we generate clustered samples : three multiple queries samples Q_1, Q_2 and Q_3 are generated with different values of the gaussian standard deviation (0.01, 0.05 and 0.2). Here, the idea is to generate sets of query points in which there is some redundancy, simulated by the clusters. Applied to content-based image retrieval, we think such sets of queries are representative of queries like objects involving textures or patterns. See for example the query of figure 1: each point of interest extracted on the sunflower carries an information which is not independent of the other ones. We expect that for such queries, the application of lemmas yields some CPU time improvement. These queries are synthetically generated in order to get a set large enough for an extensive evaluation, but in the last section (see Figure 11), we validate the approach on a set of real queries.

The performance measures were evaluated for ϵ in the interval [0,1]. Indeed for $\epsilon \leq 1$ over 100 neighbors are found. All the graphs presented are associated with the query set Q_1. We discuss the results obtained with Q_2 and Q_3 whenever their behaviors differ. All the results associated with CPU time include the pre-computation of the distances between query points that are exploited by the triangular inequality. For example, for $d = 8$ and $m = 20$ this cost is negligible in comparison to the time consumed by the similarity search itself, since it represents an average of 4.96×10^{-5}s. Although increasing with m and d, this cost keeps very small. We first apply lemmas 1, 2 and 3 as described in algorithm 4, then we study the impact of each lemma separately.

Impact of the query size

Figure 5 studies the impact of m on the number of successful tests for dimension 8. A test is a call of function *Avoidable* (algorithm 4). It is successful if it avoids some distance computation: it returns 1, 2 or 3. Although, the larger m, the smaller the percentage of successful tests, it is interesting to note that for our sample of queries, this percentage is high: it is larger than 80% and even reaches almost 95% for $m = 20$. The impact of lemmas 1 and 3 is strong when points are close to each other, whereas lemma 2 is helpful for distant points. This behavior becomes more noticeable with sample Q_2 and more particularly with Q_3 where the points are more scattered.

Impact of the dimension

Because of space limitations, only results for $m = 20$ are displayed. The same trends are observed for larger m. Figure 6 displays the ratio of successful tests versus ϵ, again with the use of lemmas 1, 2, 3 used jointly for each dimension. Clearly the lemmas are useful for avoiding distance computations. The denser the cluster the better the success.

Impact on time of lemmas 1, 2 and 3 used jointly

Figure 7 displays the CPU time measured with and without lemmas application (resp. algorithm 4 and algorithm 3) for the three dimensions, versus ϵ. As expected by the results on percentage of successful tests, the lemmas application improves CPU time. The improvement is small for $d = 8$ but significant for higher dimensions. It even reaches 25 % for $\epsilon = 1$. One may wonder why the CPU saving is small while the percentage of successful tests is high. It is perhaps due to the number of calls of function *Avoidable* which in the worst case is quadratic in the number of point queries and the number of objects (regions/points) tested.

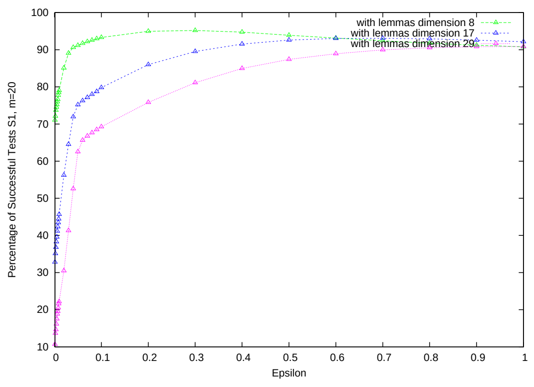


Figure 6. *Ratio of successful tests according to ϵ, for each dimension and lemmas 1,2,3 used together.*

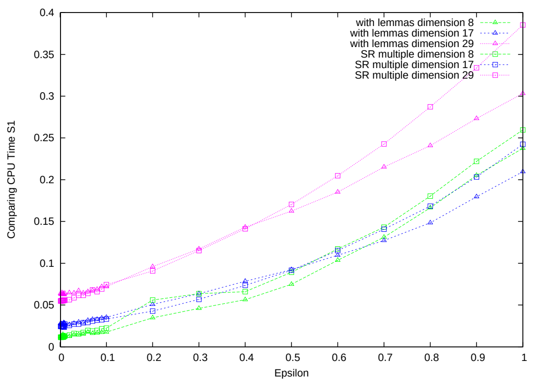


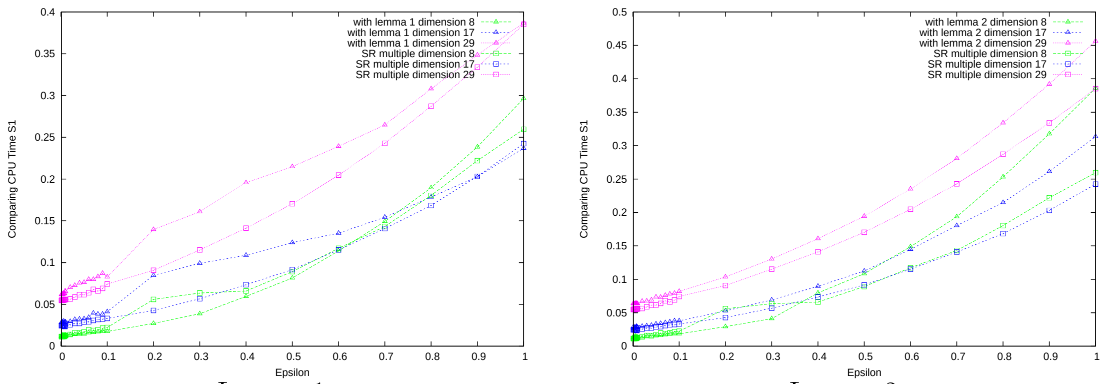
Figure 7. *Time Average according to ϵ, with and without lemmas 1,2,3 used together*

Contribution of each lemma 1,2,3 used alone

To see which of the lemmas was prominent for the CPU time reduction, we run a variant of algorithm 4 in which only one lemma out of the three is tested. Figure 8 displays CPU time versus ϵ, for $m = 20$ and for the three dimensions. For each dimension, two curves are plotted: (i) no lemma application (algorithm 3) and (ii) lemma application. It shows that applying lemma 2 alone does not provide any improvement in time while we found indeed a very significant number of successful tests. In contrast, lemma 1 reduces CPU time. Used alone, lemma 3 does not reduce CPU time except for large values of ϵ ($\epsilon > 5$).

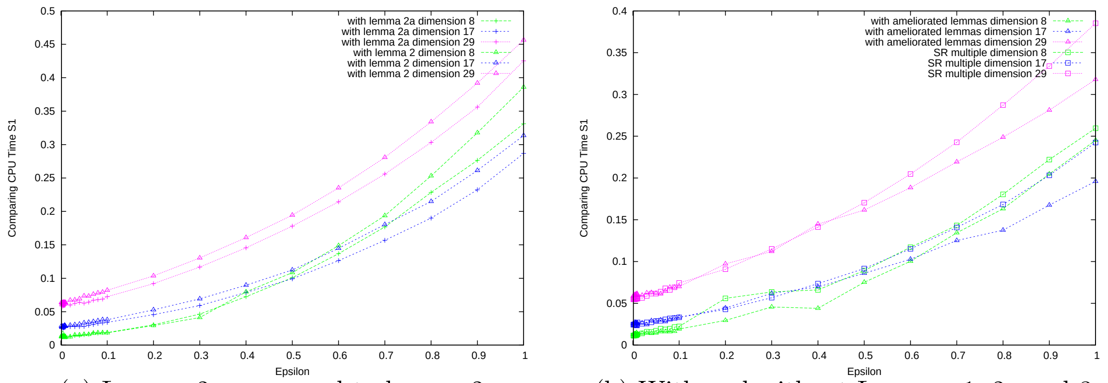
Impact of lemmas 2a and 3a

Here we evaluate the improvement carried by the versions 2a and 3a of the lemmas 2 and 3. Used alone, lemmas 3a does not reduce CPU time for this sample (Q_1) but it improves lemma 3 for Q_2 and Q_3; the corresponding curves are not presented here. But since lemma 2a improves CPU time (see Figure 9.a) we use a variant of



Lemma 1 Lemma 2

Figure 8. *Time Average versus ϵ with and without lemma 1 and 2 used alone.*



(a) Lemma 2a compared to lemma 2 (b) With and without Lemmas 1, 2a and 3

Figure 9. *Time Average versus ϵ when exploiting lemma 2a.*

algorithm 4 with lemmas 1, 2a and 3 (see Figure 9.b) which indeed exhibits a slight improvement in time with respect to regular algorithm 4 (Figure 7).

Finally, we check whether this is still true when the database size increases. This is confirmed by Figure 10, where the database contains 1M of points.

Evaluation on real queries samples

The last experiment consists in validating the proposed strategies on *real queries*. A database of 800 heterogeneous images was indexed with roughly 300 local descriptors per image. As queries, 50 areas involving objects of interest were manually determined by a user via a CBIR interface. The selected area involving a sunflower in Figure 1 is one of these query samples. Figure 11 shows the average CPU time retrieval over these samples, without using the triangle inequality (strategy #2) and using it (strategy #3). These results show that the application of the lemmas slightly improves the computing time. While the time is divided by two for small values of ϵ, for $\epsilon = 0.3$ (corresponding to an average of 41 neighbors per query point) the improvement is 6.9%.

5. CONCLUSION

This paper was devoted to the study of similarity search in a collection of points in high-dimensional space. We focused on sphere queries and on SR-trees as a structure for accelerating sphere queries. Our first contribution was to experimentally show that when the dimension of the space is moderate, i.e. between 8 and 30, the tree structure indeed performs well: it exhibits a significant gain wrt to sequential scan even in a 29-dimensional

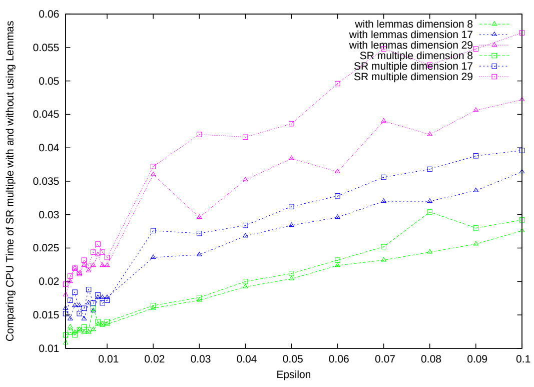


Figure 10. *Time average on 1M points with lemmas 1, 2a and 3 used jointly.*

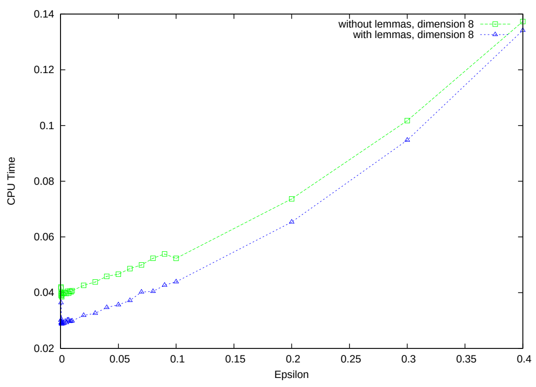


Figure 11. *Time average on real multiple queries samples with and without the lemmas.*

space. Our second and main focus was on the improvement of multiple sphere queries. The latter are useful in a large number of data mining applications as well as for image retrieval by content. As shown by Braunmüller and al.[2] for X-trees and k-NN queries, we showed that for sphere queries on an SR-tree, I/O cost is significantly reduced by processing simultaneously all queries and accessing only once each relevant node tree for all queries. Last, and this is our main contribution, an extensive performance evaluation showed that when the points in the multiple query are clustered, then CPU time can be saved as well, in processing all queries simultaneously, by avoiding distance computations. For this we enlarged the set of lemmas based on the triangle inequality proposed by Braunmüller and al..[2] Our performance study relied on a real data collection drawn from a content based image search application as well as on four synthetic data sets. These trends were confirmed by querying a database of images with queries on objects of interest.

REFERENCES

1. R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces.," in *VLDB*, pp. 194–205, 1998.
2. B. Braunmüller, M. Ester, H.-P. Kriegel, and J. Sander, "Efficiently supporting multiple similarity queries for mining in metric databases.," in *ICDE*, pp. 256–267, 2000.
3. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pp. 322–331, ACM Press, 1990.
4. N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries.," in *SIGMOD Conference*, pp. 369–380, 1997.
5. C. Schmid and R. Mohr, "Local grayvalue invariants for image retrieval," IEEE *Transactions on Pattern Analysis and Machine Intelligence* **19**, pp. 530–534, May 1997.
6. D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, pp. 1150–1157, (Corfu, Greece), 1999.
7. V. Gouet and N. Boujemaa, "Object-based queries using color points of interest," in *IEEE Workshop on Content-Based Access of Image and Video Libraries (CBAIVL 2001)*, pp. 30–36, (Kauai, Hawaii, USA), 2001.
8. K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Intl. Computer Vision and Pattern Recognition* , 2003.
9. C. Böhm, S. Berchtold, and D. Keim, "Searching in high-dimensional spaces - index structures for improving the performance of multimedia databases," *ACM Computing Survey* **33**(3), pp. 322–373, 2001.
10. S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An index structure for high-dimensional data," in *Proceedings of the 22nd International Conference on Very Large Databases*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, eds., pp. 28–39, Morgan Kaufmann Publishers, (San Francisco, U.S.A.), 1996.
11. S. Berchtold, C. Böhm, and H.-P. Kriegel, "The pyramid-technique: towards breaking the curse of dimensionality," in *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 142–153, ACM Press, (New York, NY, USA), 1998.
12. D. A. White and R. Jain, "Similarity indexing with the SS-tree," in *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pp. 516–523, IEEE Computer Society, (Washington, DC, USA), 1996.
13. COIL-100, "The Columbia Object Image Library (100 objects)," *http://www1.cs.columbia.edu/CAVE/* .