Chapter 5

Approximate similarity search

5.1 Introduction

Similarity searching has become fundamental in a variety of application areas, including multimedia information retrieval, data mining, pattern recognition, machine learning, computer vision, computational biology, data compression, and statistical data analysis. Though a lot of work has been done to develop structures able to perform similarity search fast, results are still not satisfactory, and much more investigation is needed. Accordingly, approximate similarity search has emerged as a relevant research topic. The idea of this promising approach is that a high improvement of efficiency can be obtained at the price of some controlled imprecision in the results of a query. In the following first an introduction to the issue of approximate similarity search is given, then some of the most promising approaches proposed so far are described.

5.2 Approximate similarity search issues

As discussed in Chapter 3, in order to increase efficiency, tree-based access methods create a partition of searched data set, and bound elements (set of objects) of such partition in regions [Gut84, BKK96, BÖ97, CPZ97, BÖ99, TTSF00]. Each node of the tree corresponds to a set of objects. Consider the example in Figure 5.1. The partition contains three subsets, distinguished respectively by white, black, and gray points. The subset corresponding to white points is bounded by region $\mathcal{R}_1$, the subset corresponding to black points is bounded by region $\mathcal{R}_2$, and the subset corresponding to gray points is bounded by region $\mathcal{R}_3$. When a similarity query should be processed, only nodes bounded by regions overlapping the query region should be accessed, saving a lot of disk accesses and distance computations.

However, access methods typically suffer from the so called *dimensionality curse* problem. It has been observed that, when the number of dimensions of a data set is greater than 10-15, performance of access methods decreases and a linear scan over the whole data set would perform better [BGRS99, WSB98]. One consequence of the dimensionality curse is that the probability of overlaps between the query and data regions is very high and the execution of a similarity query may require to access many of the data regions loosing the advantage of any indexing structure what so ever. Indeed all data regions that overlap the query region must be accessed. For instance, in Figure 5.1, the query region overlaps regions $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ so all of them should be accessed to answer to the query.

Given this inefficiency problem other techniques are being investigated. Here we discuss the *approximate similarity search* approach [AMN+98, PAL99, CP00], that has recentely emerged as important research issue. The idea behind approximate

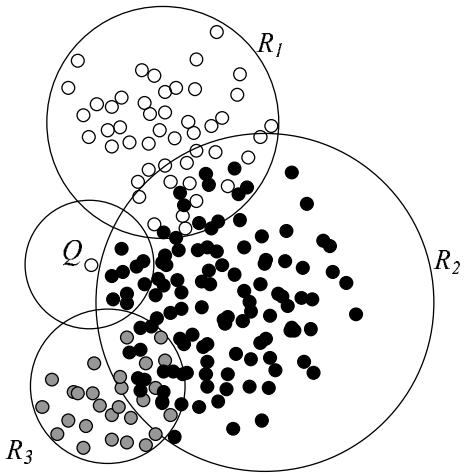

Figure 5.1: Partitions, data regions, and query regions

similarity search is that queries are processed faster at the price of some imprecision in the results. The approximation is obtained by relaxing some constraint at the basis of "exact" similarity search algorithms.

Approximate similarity search is also encouraged by other observations. It can be noticed that the nature of the similarity based search process is intrinsically interactive. Users, typically, issue several similarity queries, to the search system, eventually reusing results of current queries to express new ones. For instance, the user may start using an initial image, to search for similar ones, then uses images returned by the query to issue new similarity search queries. In this case, efficient execution of elementary queries is even more important and users may accept some imprecision, in the temporary results, at the price of faster responses. Furthermore, some controlled imprecision in the result of a similarity search query may not be noticed by users or will be accepted when the increase in performance obtained is high. In fact, similarity is an intuitive and subjective measurement. People, for instance, judge the similarity between two images differently. However, this subjectivity is lost when similarity is

defined by a mathematical formula.

5.3 Survey of existing approaches to approximate similarity search

In the following, we present a short survey of the most important approaches to the approximate similarity search in order to highlight our contribution. Approaches to approximate similarity search can be classified into two broad categories [FTAA01]: (1) approaches able to reduce the size of data objects, and (2) approaches able to reduce the data set that needs to be examined. In the following we discuss these two categories. Specifically, approaches belonging to the second one will be treated more deeply since the techniques that we propose in this thesis belong to it.

5.3.1 First category: approaches able to reduce the size of data objects

The first category is mainly based on dimensionality reduction. Typically linear-algebraic methods such as Karhunen-Loeve Transformation (KLT) [Fuk90], Discrete Fourier Transform (DFT) [OS89], Discrete Cosine Transform (DCT) [Kai85], or Discrete Wavelet Transform (DWT) [Cas96] are used. All these methods assume that a few dimensions are enough to retain the most important information regarding the represented data objects so the other dimensions can be simply ignored.

Another approach in this category is the VA-file [WSB98], also introduced in Section 3.2.1. It reduces the size of multidimensional vectors by quantizing the original data objects. In this approach, nearest neighbor search is performed in two steps. In

the first step, the approximated vectors are scanned identifying candidate vectors. In the second step, the found candidates are visited in order to find the actual nearest neighbors. A modification to VA-files was proposed in [FTAA00], where the creation of a VA-file is improved by first transforming the data using KLT in a more suitable domain. Though the improvement of such techniques is significant, they only work for vector spaces.

An interesting approach also falling in this category, which can also be applied to non vector data, is the FASTMAP [Fal96]. It supposes to have n objects and an $N \times N$ distance matrix. FASTMAP tries to project these objects in a dim dimensional vector space, by only using the information given by the distance matrix, in such a way that distances are preserved. Of course, Euclidean distance between points in the vector space approximates the real distance between objects defined in the matrix. The quality of the approximation depends on the number of dimensions of the target vector space and on the specific distance matrix. The idea behind FASTMAP is to project objects on a specific line passing by two pivot objects in a dim dimensional vector space. A special heuristic is used to carefully select the two pivot objects.

5.3.2 Second category: approaches able to reduce the data set that needs to be examined

Algorithms of the second category use strategies to reduce the data set that needs to be examined. These can be further classified depending on the specific strategy used to achieve their goal:

Relaxed branching strategies This strategies can be used with access methods based on hierarchical decomposition of the space. Their aim is not to access

regions when they are not likely to contain desired results, or when access would only marginally improve the existing results. In this case, an approximate *pruning condition* is used to decide if a region should be accessed or not.

Early termination strategies In this case, search algorithms are prematurely stopped when current result is judged to be satisfying the approximation requirements. This strategies uses a *stop condition* to decide if it is time to stop the algorithm – the search terminates as soon as the chances for obtaining significantly better results become low. Here the hypothesis is that after some steps of search iteration, good approximation is obtained while further improvement is of minor importance and consume most of the total search costs.

Our algorithms for approximate similarity search belongs to this category since they aim at reducing the portion of the data set needed to be examined in order to find the result. Two of them use relaxed branching strategies, the other two adopt early termination strategy. Accordingly, they relay on an accurate definition of either a pruning condition or a stop condition respectively.

In the following we discuss more in details some of the most authoritative approaches belonging to this category.

5.3.3 Approximate nearest neighbors searching using BBD trees

Suppose to have a set of points $\mathcal{DS}$ in a *dim* dimensional vector space and a query object O_q. Let O_N be the nearest neighbor of O_q, and O_A some other object in the searched collection. Given $\epsilon > 0$, provided that $0 < d(O_N, O_q) \leq d(O_A, O_q)$, O_A is an

(1+ϵ)-nearest neighbor of O_q if

$$\frac{d(O_A, O_q)}{d(O_N, O_q)} \leq 1 + \epsilon$$

That is, O_A is within relative error ϵ of the true nearest neighbor. This idea can be generalized to the case of the j-th nearest neighbor of O_q, for $1 \leq j \leq n$, where n is the size of the database. Using respectively O_A^j and O_N^j to designate, the j-th approximate and nearest neighbor, the constraint should be modified as follows

$$\frac{d(O_A^j, O_q)}{d(O_N^j, O_q)} \leq 1 + \epsilon.$$

If this constraint is satisfied, O_A^j is called the *(1+ϵ)-j-approximate nearest neighbor* of O_q.

The algorithm proposed in [AMN+98], given a data set $\mathcal{DS}$ represented in a vector space with distances measured using functions of the Minkowski family, and given a query object O_q, guarantees to find an (1+ϵ)-approximate nearest neighbor of O_q in $O(\log n)$ time. Alternatively, when k-nearest neighbors search is considered, the algorithm guarantees to find k (1+ϵ)-k-approximate nearest neighbors[1] of O_q in $O(k \log n)$ time. The parameter ϵ can be used to control the tradeoff between efficiency and quality of the approximation. The higher ϵ, the higher the performance, the higher the error.

This algorithm uses as underlying indexing structure a so called *balanced box-decomposition (BBD) tree* that is a variant of a quad-Tree [Sam95] and is similar to other balanced structures based on box-decomposition [BET95, Bes95, CK95]. Specifically, this structure is based on a hierarchical decomposition of the space where

[1]Notice that also one of the techniques proposed in this thesis is designed in such a way that it returns k (1+ϵ)-k-approximate nearest neighbors, see Section 6.6 and [ZSAR98].

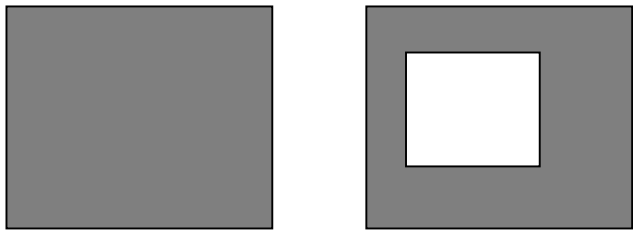

Figure 5.2: Possible regions in a BBD tree: a) *dim*-dimensional rectangle and b) set theoretic difference of two rectangles.

regions are represented by nodes of a tree that has $O(\log n)$ height. Each node of the tree is associated with a region and points to other nodes. Regions associated with nodes pointed by the same parent node do not overlap each other. There are two types regions in a BBD tree. A region can be a *dim*-dimensional rectangle or the set theoretic difference of two rectangles, one enclosed within the other, as depicted in Figure 5.2 respectively by picture a) and b). These rectangles are *fat*, in the sense that the ratio between the longest and the shortest sides is bounded. The tree has $O(n)$ nodes and it can be built in $O(dim \cdot n \cdot \log n)$ time, where n is the size of the data set. Each region associated with a leaf node contains a single object. The set of leaf regions defines a partition of the space.

The nearest neighbor algorithm on this data structure is intuitively defined as follows. Given a query object O_q, the tree is traversed and the leaf node associated with the region containing the query is found. Given the properties of the BBD trees, the leaf node is found in $O(\log n)$ and there is only one leaf node that contains it. At this point, a *priority search* is performed by enumerating leaf regions in increasing order of distance from the query object. The distance from an object O to a region is computed as the distance of O to the closest point that can be contained in the

region. When a leaf region is visited, the distance of the associated object from O_q is measured and the closest point seen so far is recorded. Let us call O_A such a current closest point. The algorithm stops when current leaf region is such that its distance is larger than $d(O_q, O_A)$, that is, the current region cannot contain objects whose distance from the query object is shorter than that of O_A. Since all remaining leaf regions are farther than current region, it means that O_A is the nearest neighbor to O_q.

The approximate nearest neighbor algorithm uses a stop condition to prematurely stop the search algorithm. Specifically, the algorithm stops as soon a the distance to the current leaf region exceeds $d(O_q, O_A)/(1+\epsilon)$. It is easy to show that in these circumstances O_A is a $(1+\epsilon)$-approximate nearest neighbor. To clarify the behavior of the exact and approximate nearest neighbor search algorithms consider Figure 5.3. Data objects are represented by black spots. Each object is included in a rectangular region associated with a leaf node. Each region is identified by a number assigned incrementally according to the distance of the region from the query object O_q. Thus, region 1 is the closest to O_q, in fact it contains O_q, while region 10 is the farthest. The search algorithm starts from region 1 to collect the potential nearest object to O_q. In the figure, we suppose that region 3 was accessed and object O_A was found as the current closest object. The circumference in the figure has in fact radius equal to $d(O_q, O_A)$. The exact algorithm would continue accessing regions that overlap the circumference and it would stop after accessing region 10, which contains the exact nearest neighbor. The approximate algorithm, on the other hand, would access only regions that overlap the dotted circumference, which has a radius equal to $d(O_q, O_A)/(1+\epsilon)$. Therefore it would stop after accessing region 8, missing the

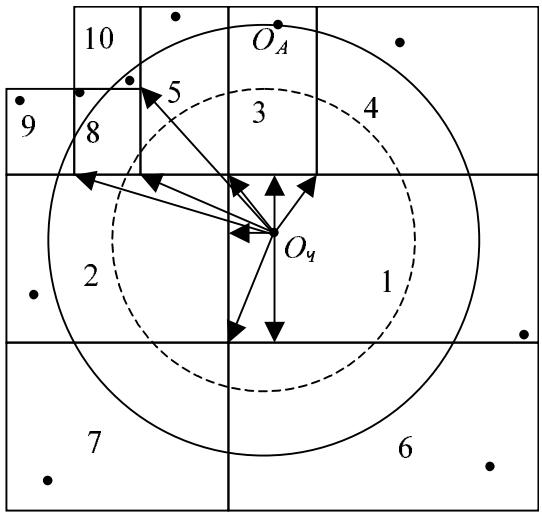

Figure 5.3: Overview of the approximate nearest neighbors search algorithm using BBD trees

exact nearest neighbor.

The priority search can be performed in $O(\log n)$ times the number of regions that are visited, by using an auxiliary heap. Let m be the number of regions visited, m has an upper bound depending only on dim and ϵ, for any Minkowski metric, defined as $\lceil 1+6dim/\epsilon \rceil^{dim}$. This bound arises from the maximum number of disjoint dim-cubes of diameter $\epsilon/3$ that can intersect a dim-ball region of radius 1. Since the upper bound can be considered a constant, provided that dim and ϵ are fixed, the algorithm finds the $(1+\epsilon)$-approximate nearest neighbor in $O(\log n)$ time.

The algorithm can be easily extended to the case of k nearest neighbors. In fact, in this case the current k closest objects are recorded and the algorithm stops when the distance of the current k-th closest object O_A^k is such that $d(O_q, O_A^k)/(1+\epsilon)$. In this case O_A^k is a $(1+\epsilon)$-k-approximate nearest neighbor. In this algorithm, the upper bound for the number of accessed regions m is $2k + \lceil 1 + 6dim/\epsilon \rceil^{dim}$. Therefore, provided that dim and ϵ are fixed, the algorithm finds the $(1+\epsilon)$-k-approximate

nearest neighbor in $O(k \log n)$ time.

Notice that upper bound m does not depend on the size of the data set n. However, it depends exponentially on *dim* so this algorithm can be practically used only on vector spaces with a small number of dimensions, e.g. in the range from 2 to 20. This algorithm can only be used in case of data represented in a vector space with distances measured by using Minkowski metrics. In particular, it cannot be used in case of data represented in a generic metric space.

5.3.4 Approximate range searching using BBD trees

The use of BBD trees, briefly described in previous section, was also exploited to design an approximate range search algorithm. This approach was proposed in [AM95]. Let us suppose to have a data set $\mathcal{DS}$ of objects defined in a *dim* dimensional vector space, and to use a function of the Minkowski family to measure distances among objects. Let us also suppose to have a range query $\mathbf{range}(O_q, r_q)$ defined by the ball region $\mathcal{B}(O_q, r_q)$. Typically, the goal of a range query is to retrieve the set of objects of $\mathcal{DS}$ included in $\mathcal{B}(O_q, r_q)$. Alternatively, in some cases one may just need to count the number of objects that qualify for the range query or, more generally, we may suppose that objects of $\mathcal{DS}$ have been assigned a weight, and the goal is to compute the accumulated weight $weight(\mathbf{range}(O_q, r_q))$ of the objects that qualify for the query. Retrieving the set of objects included in a range query has in general a cost higher than counting them, however in this work only the counting version of the problem was considered.

The exact range search algorithm returns the exact weight of the result set, while the approximate range search algorithm returns a weight that approximates the real

weight of the actual result set. The idea of this approximation technique is to consider range queries as *fuzzy* range queries. In a fuzzy range query objects that are "close" to the boundary of the query may or may not be included in the count.

Given a range query defined by the ball region $\mathcal{B}(O_q, r_q)$, and given a real value $\epsilon > 0$, regions $\mathcal{B}^-(O_q, r_q/(1+\epsilon))$ and $\mathcal{B}^+(O_q, r_q \cdot (1+\epsilon))$ can be defined. Regions $\mathcal{B}^-$ and $\mathcal{B}^+$ have the same center of $\mathcal{B}$, however, their radius is respectively reduced or increased of a factor $1+\epsilon$. In the following, for sake of simplicity, by using $\mathcal{B}^-$, $\mathcal{B}$, and $\mathcal{B}^+$ we also refer to the set of objects of $\mathcal{DS}$ respectively included by these regions.

The value $weight(\mathcal{R}^A)$, where $\mathcal{R}^A \subseteq \mathcal{DS}$, is a *legal answer* to an *(1+ϵ)-approximate range query* when

$$\mathcal{B}^- \subseteq \mathcal{R}^A \subseteq \mathcal{B}^+.$$

That is, all objects of $\mathcal{B}^-$ should be included, all objects of $\mathcal{DS} \setminus \mathcal{B}^+$ should not be included, and objects of $\mathcal{B}^+ \setminus \mathcal{B}^-$ may or may not be included.

Notice that this approach allows for false dismissals and false hits. In fact, when some objects of $\mathcal{B} \setminus \mathcal{B}^-$ are not included false dismissals occur, while when some objects of $\mathcal{B}^+ \setminus \mathcal{B}$ are included false hits occur.

Range queries can be answered by using search algorithms on BBD trees. Let us first consider the exact range search algorithm. We suppose that, given a node N of a BBD tree, the value $weight(N)$, defined as the sum of the weights of all objects included in the region associated with N, is also registered in the node N, so no access to children is required to compute it. In the following, by using N we also refer to the set of objects contained in node N. Given a range query $\mathbf{range}(O_q, r_q)$, the exact range search algorithm traverse the BBD tree counting the weight of objects inside $\mathcal{B}(O_q, r_q)$ as follows. The algorithm starts from the root node of the tree and

initialize a global variable *count* to 0. Given a node N_i of the tree the algorithm does the following:

(a) if $N_i \subset \mathcal{B}$ add $weight(N)$ to *count*; Stop.

(b) if $N_i \cap \mathcal{B} = \emptyset$ do nothing; Stop.

(c) if N_i is a leaf node check if the associated object is included in $\mathcal{B}$, and in that case add its weight to *count*; Stop.

(d) if N_i is an internal node, recursively consider all its children and add the weights respectively obtained.

The behaviour of this algorithm is sketched in Figure 5.4a). If the region associated with the current node is included in $\mathcal{B}$, its weight is immediately considered, since it is stored in the node, without accessing its children. Therefore, in the figure, the weight of $\mathcal{R}_1$ is immediately considered. If the region associated with the current node is outside the query region, as $\mathcal{R}_3$ in the figure, it is immediately discarded. In the other cases, the specific children are accessed. Therefore, children of the node corresponding to region $\mathcal{R}_2$ are accessed.

This algorithm can be slightly modified to answer to an $(1+\epsilon)$-approximate range query using an approximate pruning condition as follows:

(a) if $N_i \subset \mathcal{B}^+$ add $weight(N)$ to *count*: Stop.

(b) if $N_i \cap \mathcal{B}^- = \emptyset$ do nothing; Stop.

(c) if N_i is a leaf node check if the associated object is included in $\mathcal{B}$, and in that case add its weight to *count*; Stop.

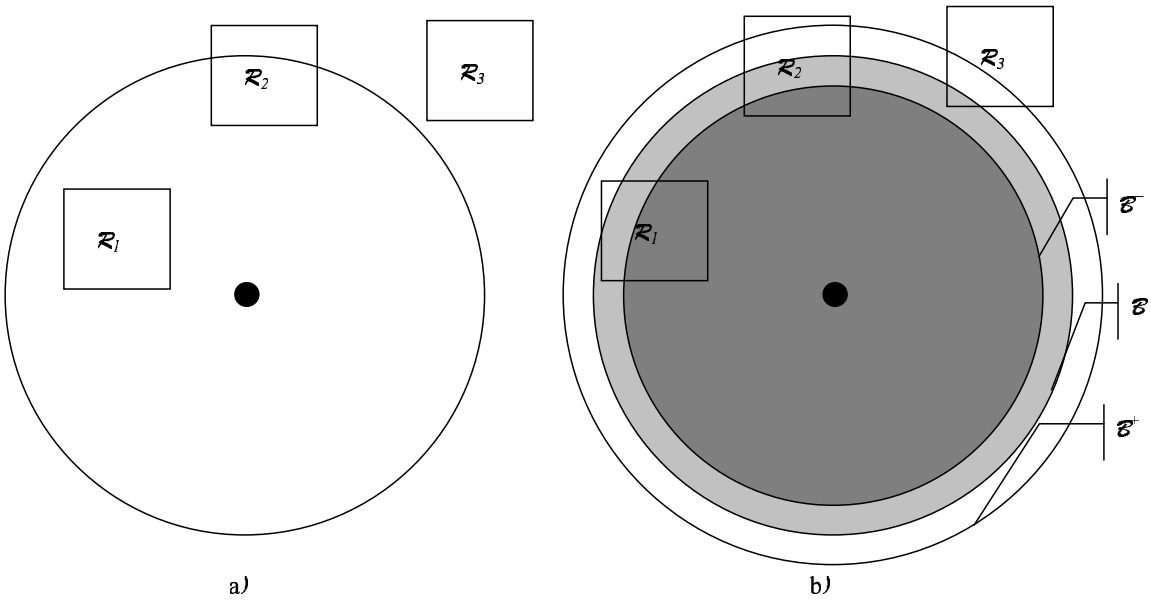

Figure 5.4: Range queries using BBD tree: a) exact behaviour and b) approximate behaviour

(d) if N_i is an internal node, recursively consider all its children and add the weights respectively obtained.

The behaviour of the approximate algorithm is sketched in Figure 5.4b). If the region associated with the current node is included in $\mathcal{B}^+$, its weight is immediately considered without accessing its children – false hits may occur, since objects outside the query region $\mathcal{B}$ may be counted. In the figure, the weight of $\mathcal{R}_1$ is consequently immediately considered. If the region associated with the current node is outside the region $\mathcal{B}^-$ and not included in $\mathcal{B}^+$, as $\mathcal{R}_3$ in the figure, it is immediately discarded – false dismissals may occur, since objects inside $\mathcal{B}$ may be missed when they are outside $\mathcal{B}^-$. In the remaining cases, children of the corresponding node are accessed. Thus, in the figure, children of the node corresponding to region $\mathcal{R}_2$ are accessed.

The complexity of the search algorithm is proven to be $O(\log n + (1/\epsilon)^{dim})$ and the

lower bound of the maximum number of nodes visited is $\log n + (1/\epsilon)^{dim-1}$. Notice that for values of ϵ smaller that 1, the complexity increases exponentially with the number of dimensions.

The algorithm as it is, can only be used to count (compute weight of) objects that qualify for an approximate range query. Specifically, it cannot be used to retrieve the set of objects qualifying for the approximate range query. In fact, in order to obtain that, leaf nodes should always be accessed, since leaf nodes are the only nodes containing pointers to real objects.

5.3.5 Approximate nearest neighbors searching using angle property

In [PAL99, PL99] an original technique for reducing the number of node accessed during nearest neighbors searching is proposed. The main novelty of this technique consists in the use of the angle between objects contained in a ball region, defined in a vector space, and a query object, with respect to the center of the ball region as shown in Figure 5.5. Exploiting this angle, some heuristics to decide whether a region should be accessed or not are proposed. This technique was applied to SS-Trees [WJ96], however it can be applied to all access methods for vector spaces that partition the space, bound elements of the partition with ball regions, and organize regions hierarchically.

The proposed heuristics are justified by the following three properties of data sets represented in high dimensional vector spaces:

Pr_1: As dimensionality rises, the points that are bounded by a ball region become almost equidistant from the center of the ball region.

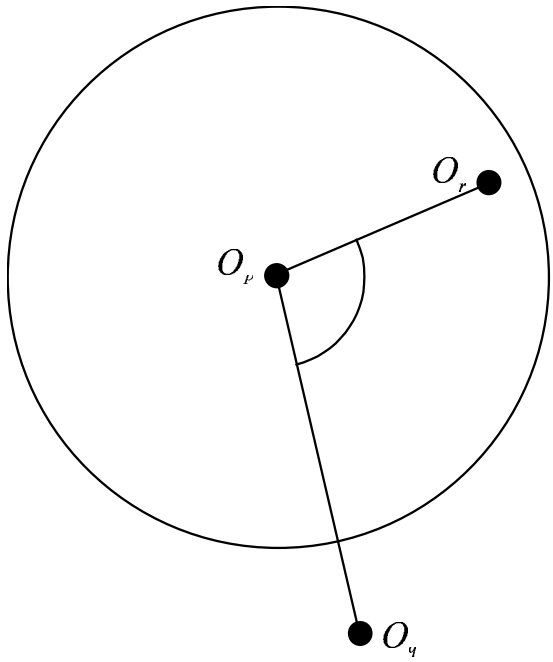

Figure 5.5: Angle between objects contained in a ball region and a query object with respect to the center of the ball region

Pr_2: As dimensionality increases, the radii of the, smaller, child ball regions grow nearly as large as the radius of the larger parent ball region, thus also their centers tend to be close each other.

Pr_3: Given a set of points within a bounding ball region and some query point in the vector space, the angle between the query point and each point in the ball region falls inside a decreasing interval of angles around $\pi/2$.

Let us discuss in detail the proposed heuristics. Exact nearest neighbors search algorithms access all regions that overlap the current query region. The proposed heuristics uses an approximate pruning condition to decide whether a region should be accessed or not.

In [PAL99] they propose to access a region if one of the following conditions is true:

C_1: The corresponding node is an internal node.

C_2: The current query region includes the center of the region's parent

C_3: The center of the region resides in the half of the parent ball region closer to the query object (that is the angle between the center of the region and the query object, with respect to the center of the parent's ball region is smaller than $\pi/2$).

Condition C_1 forces all internal node to be examined. This is justified because in SS-Trees much of the performance degradation in nearest neighbor searching is due to the accesses to the leaf nodes. Properties Pr_1, Pr_2 and Pr_3 suggest that in high dimensional vector spaces, objects reside close to the border of their bounding region, and in particular since child regions tend to be as large as their parent, objects are close to the border of the parent region. In addition these objects fall close to a 90 degree angle with a given query object. Therefore as suggested by Figure 5.6 regions whose center are in the half ball region closer to O_q, as required by condition C_3, are more likely to contain qualifying objects. Finally C_2 was proposed to avoid cases where applying C_3 can be harmful since several qualifying objects may be missed.

In order to check C_3 the evaluation of the angle between the query region and the region's center, with respect to its parent's center is needed. If the angle is acute then the region's center is in the half of the parent region closer to the query region. Deciding if the angle is acute, obtuse, or right can be done by using the *dot product*. Let us suppose that the query region O_q is $(v_1^q, \ldots, v_{dim}^q)$, the region's center O_r is

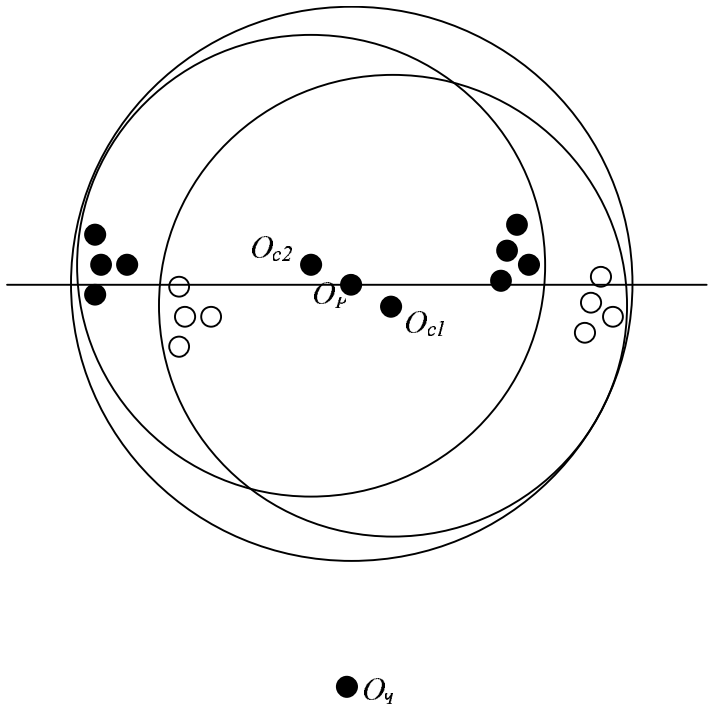

Figure 5.6: Objects belonging to children whose center is in the closest half of the parent node are more likely to contain nearest neighbors

$(v_1^r, \ldots, v_{dim}^r)$, and its parent's center O_p is $(v_1^p, \ldots, v_{dim}^p)$. Then, the dot product $[O_q \cdot O_r]_{O_p}$ of O_q and O_r, with respect to O_p is computed as follows:

$$[O_q \cdot O_r]_{O_p} = \sum_{1}^{dim} (v_i^q - v_i^p)\,(v_i^r - v_i^p)$$

When $[O_q \cdot O_r]_{O_p} > 0$ the angle is acute. If $[O_q \cdot O_r]_{O_p} < 0$ the angle is obtuse. Finally, if $[O_q \cdot O_r]_{O_p} = 0$ the angle is right.

In [PL99] the previous pruning conditions were improved. In fact, the zone where is more likely to find qualifying objects according to properties Pr_1, Pr_2, and Pr_3, is close to the border of the data region, and forms an angle of about 90 degree with the query object, with respect to the region's center. Let suppose to indicate by θ

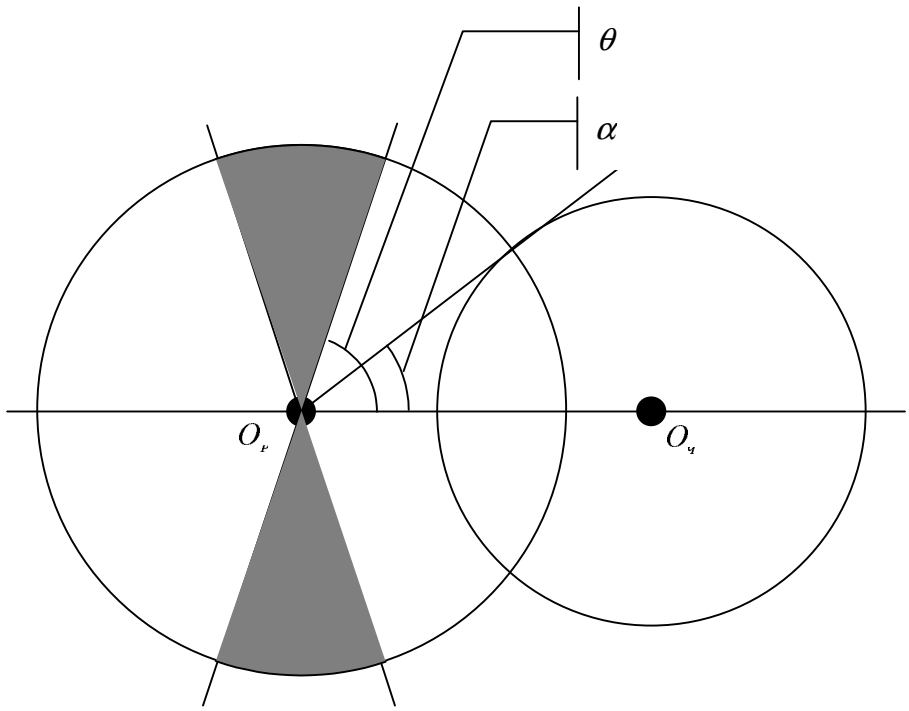

Figure 5.7: If the query region does not intersect promising portions of the data region, this is discarded.

the expected angle of the promising zone. If the angle α obtained considered the intersection between the two regions and the line passing from their centers, with respect to the data region's center, is higher than θ then the region is accessed, elsewhere it is discarded. Figure 5.7 sketches this situation. Notice that when θ is set to 0 all regions overlapping the query region are accessed.

These methods can only be applied to nearest neighbors search in vector spaces. The first version of the proposed approach cannot be tuned to trade performance with quality of results, while the second allows one to use the angle θ as a threshold to decide when regions can be discarded.

5.3.6 PAC nearest neighbor searching

In [CP00] an approach for *Probably Approximately Correct* nearest neighbor search in metric spaces is proposed. The idea is to bound the error on the distance of the approximate nearest neighbor so that an $(1+\epsilon)$-approximate nearest neighbor is found, similarly to the technique using BBD trees, see [AMN+98] and Section 5.3.3. In addition, the proposed algorithm may prematurely stop when the probability that the current approximate nearest neighbor is not an $(1+\epsilon)$-approximate nearest neighbor is below a user defined threshold δ. In this proposal the approximation is controlled by two parameters. The ϵ parameter is used to specify the upper bound on the desired relative error on the distance of the approximate nearest neighbor. The δ parameter is used to specify the degree of confidence that the ϵ upper bound is not exceeded. Notice that if δ is set to zero, the proposed algorithm stops when the resulting object is guaranteed to be a $(1+\epsilon)$-approximate nearest neighbor. Values of δ greater than zero may return an object that is not an $(1+\epsilon)$-approximate nearest neighbor. On the other hand, when ϵ is set to zero, δ controls the probability that the retrieved object is not the real nearest neighbor. Of course, when both ϵ and δ are set to zero, the exact nearest neighbor is always found.

More formally, let O_q be the query object, O_N the exact nearest neighbor, and O_N^A the approximate nearest neighbor found. Let ϵ_{act} the actual error on distances, that is

$$\epsilon_{act} = \frac{d(O_q, O_N^A)}{d(O_q, O_N)} - 1.$$

The proposed approximate nearest neighbor algorithm retrieve a $(1+\epsilon)$-approximate nearest neighbor with confidence δ. That is, the retrieved object is such that

$$\Pr\{\epsilon_{act} > \epsilon\} \leq \delta.$$

These ideas were applied to the algorithm for searching the nearest neighbor to a query object in M-Trees [CPZ97]. Therefore this method is not limited to the case of data represented in vector spaces, but data represented in a generic metric space can be searched using this approach.

The authors observe that a similarity search process can be conceptually split in two phases.

Locating In the first phase the search algorithm locates the object that will be returned

Stopping The second phase, the longest one, is used to determine that the object retrieved is in fact the correct one

Accordingly the two approximation parameters ϵ and δ are used respectively to control how the object should be located, and when the algorithm should stop.

Let us see how this two actions are accomplished. The nearest neighbor search algorithm for M-Trees recursively access data regions (nodes) of the tree starting from the root nodes. When the query region, defined by the query object and the distance of the current nearest object found, and a data region overlap, the data region is accessed. The overlap test here is relaxed so that $(1+\epsilon)$-approximate nearest neighbor are considered. This relaxed overlap test was inspired by the technique proposed in this thesis, see Section 6.6 and [ZSAR98]. The exact algorithm stops when no more overlapping regions are found. The proposed algorithm, on the other hand, stops when the probability that the effective error does not exceed ϵ is smaller than δ. This stop condition is obtained using $G_{O_q}(x)$, the *distribution of the nearest neighbors* of

O_q with respect to a specific data set $\mathcal{DS}$ of size n, defined as follows [CPZ98a]:

$$G_{O_q}(x) = \mathtt{Pr}\left\{\exists O \in \mathcal{DS} : d(O_q, O) \leq x\right\} = 1 - (1 - F_{O_q}(x))^n.$$

As previously stated, the algorithm should stop when $\mathtt{Pr}\left\{\epsilon_{act} > \epsilon\right\} \leq \delta$, where ϵ_{act} is the actual relative error on distances. That is when

$$\begin{aligned}&\mathtt{Pr}\left\{\exists O \in \mathcal{DS} : d(O_q, O_N^A)/d(O_q, O) - 1 \geq \epsilon\right\} = \\ &= \mathtt{Pr}\left\{\exists O \in \mathcal{DS} : d(O_q, O) \leq d(O_q, O_N^A)/(1+\epsilon)\right\} \leq \delta\end{aligned}$$

and this correspond to check if

$$G_{O_q}(d(O_q, O_N^A)/(1+\epsilon)) \leq \delta.$$

This algorithm can be used for generic metric spaces and its performance is good, however it is limited to the case of just a single nearest neighbor search.