PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces

Paolo Ciaccia
DEIS - CSITE-CNR, University of Bologna
Bologna, Italy
pciaccia@deis.unibo.it

Marco Patella
DEIS - CSITE-CNR, University of Bologna
Bologna, Italy
mpatella@deis.unibo.it

Abstract

In high-dimensional and complex metric spaces, determining the nearest neighbor (NN) of a query object q can be a very expensive task, because of the poor partitioning operated by index structures – the so-called "curse of dimensionality". This also affects approximately correct (AC) algorithms, which return as result a point whose distance from q is less than $(1+\epsilon)$ times the distance between q and its true NN.
In this paper we introduce a new approach to approximate similarity search, called PAC-NN queries, where the error bound ϵ can be exceeded with probability δ and both ϵ and δ parameters can be tuned at query time to trade the quality of the result for the cost of the search. We describe sequential and index-based PAC-NN algorithms that exploit the distance distribution of the query object in order to determine a stopping condition that respects the error bound. Analysis and experimental evaluation of the sequential algorithm confirm that, for moderately large data sets and suitable ϵ and δ values, PAC-NN queries can be efficiently solved and the error controlled. Then, we provide experimental evidence that indexing can further speed-up the retrieval process by up to 1-2 orders of magnitude without giving up the accuracy of the result.

1. Introduction

Similarity queries have become a fundamental paradigm for multimedia, data mining, decision support, pattern recognition, statistical, and medical applications, to list a few. In its essence, the problem is to determine the object which is most similar to a given query object. This is usually done by first extracting the relevant *features* from the objects (e.g. color histograms from still images [15], Fourier coefficients from time series [1]), and then measuring the *distance* between feature values, so that similarity search becomes a *nearest neighbor* (NN) query over the space of feature values.

To speed-up NN search, feature values, which often are high-dimensional (high-D) vectors, can be indexed by means of either multi-dimensional trees (such as the R*-tree [4], the SR-tree [18], and the X-tree [6]) or *metric* trees (e.g. the M-tree [10] and the mvp-tree [8]). Metric trees only require the distance between feature values to be a *metric*, thus they can be used even when no adequate vector representation for the features is possible.

It is a fact that, depending on the characteristics of the data set at hand, indexing might not be the best solution. Indeed, the performance of index trees has been repeatedly observed to deteriorate in high-D spaces, so that, even for D as low as 10-15, a linear scan of the data set would perform (much) better [7, 24, 18]. Furthermore, recent mathematical studies demonstrate that this unpleasant phenomenon, known as "the curse of dimensionality", is not peculiar to vector spaces, but can also affect more complex metric spaces [20], it being tightly related to the distribution of distances between the indexed objects and the query object [7]. Intuitively, the more such distances are all similar each other, i.e. their *variance* is low, the more searching is difficult.

On the other hand, when objects are naturally organized into clusters or the *intrinsic* (or *fractal*) dimensionality of the data set is low, NN search can be efficiently solved [3, 7, 10, 18]. In this case, a (*multi-step*) filter-and-refine approach has also been proposed, the idea being to initially use an easy-to-compute distance function that lower bounds the original one, and then to compute the actual result by evaluating the original distance function only on the set of *candidates* returned by the filter step. This is also the basic idea underlying the use of dimensionality-reduction techniques [21].

In this paper we pursue a different, yet complementary,

direction that extends previous work on *approximate* NN search, i.e. when one does not require that the result has necessarily to be the "correct" NN of the query object. Approximate queries are suitable to a variety of scenarios, especially when the query specification is itself a "guess". This is the case in exploratory data analysis, in content-based image retrieval, and in many other real-life situations. Furthermore, in many cases the difference between the NN and a "good" approximation is indistinguishable from a practical point of view.

With approximate queries, the two conflicting requirements to be satisfied are low processing costs and high accuracy of the results, i.e. low errors. The approach undertaken by what here we call *approximately correct* NN (AC-NN) queries [2] is to specify the maximum relative error to be tolerated, $\epsilon > 0$, thus one is guaranteed to obtain a result whose distance from the query object does not exceed $(1 + \epsilon)$ times the distance between the query object and its NN. Unfortunately, AC-NN algorithms are still plagued by the dimensionality curse and become unpractical when D is intrinsically high, *regardless* of ϵ.

In this paper we propose a *probabilistic* approach to approximate NN search, which allows two parameters to be specified at query time: the *accuracy* ϵ allows for a certain relative error in the result, and the *confidence* δ guarantees, with probability at least $(1 - \delta)$, that ϵ will not be exceeded. This generalizes both AC-NN queries, obtained when $\delta = 0$, as well as *correct* (C-NN) queries ($\epsilon = \delta = 0$). The basic information used by our PAC (*probably approximately correct*) NN algorithms is the *distance distribution* of the query object, which is exploited to derive a *stopping condition* with provable quality guarantees, the basic idea being to avoid searching "too close" to the query object.

We first analytically and experimentally demonstrate the effectiveness of a PAC-NN sequential algorithm. Results show that, say, with $n = 10^6$ objects and $D = 100$, only about 7000 objects need to be read in order to obtain, with probability ≥ 0.99, a result that differs no more than 10% from the correct one. Since the complexity of the PAC-NN sequential algorithm is at least $O(n\delta^{-1}(1+\epsilon)^{-D})$, thus still linear in the data set size, we introduce a PAC-NN index-based algorithm that we have implemented in the M-tree [10], and experimentally demonstrate that performance can improve by 1-2 orders of magnitude. Although we use the M-tree for practical reasons, our algorithm and results apply to *all* multi-dimensional and metric index trees. We also demonstrate that, for any value of the ϵ accuracy parameter, the δ confidence parameter can be chosen in such a way that the *actual* average relative error stays indeed very close to ϵ. This implies that an user can indeed exert an effective control on the quality of the result, thus trading accuracy for cost.

The rest of the paper is organized as follows. After reviewing the basic logic of C-NN and AC-NN algorithms (Section 2), in Section 3 we emphasize the distinction between the task of "locating" the result (either correct or approximate) and the task of "stopping" the search, and show that the first task is relatively easy, whereas stopping is the real trouble. Then we exploit this observation by introducing PAC-NN queries, and formalize the relationship between the distance distribution and the stopping condition used by PAC-NN algorithms. Section 4 provides analytical and experimental evaluation for sequential data sets, and Section 5 introduces and evaluates the PAC-NN index-based algorithm on both synthetic and real data sets. Finally, in Section 6 we discuss other approaches to approximate NN search and draw our conclusions.

2. NN and approximate NN search algorithms

For the sake of generality, we develop our arguments by considering that objects are points of a *metric space* $\mathcal{M} = (\mathcal{U}, d)$, where $\mathcal{U}$ is the domain of values and d is a metric – a non-negative and symmetric function which satisfies the triangular inequality, $d(p_i, p_j) \leq d(p_i, p_k) + d(p_k, p_j)\ \forall p_i, p_j, p_k \in \mathcal{U}$ – used to measure the distance (dis-similarity) of points of $\mathcal{U}$.

Some basic definitions are useful for what follows (the relevant notation is summarized in Table 1). For any real $r \geq 0$, $\mathcal{B}_r(c) = \{p \in \mathcal{U} \mid d(c, p) \leq r\}$ is the *r-ball* of point c, that is, the set of points in $\mathcal{U}$ whose distance from c does not exceed r. Given a query point q, the minimum distance between q and a region $R \subseteq \mathcal{U}$ is defined as $d_{min}(q, R) = \inf\{d(q, p) \mid p \in R\}$. Note that $d_{min}(q, R) = 0$ if $q \in R$. Finally, given a set $S \subset \mathcal{U}$ of n points, and a query point $q \in \mathcal{U}$, the *nearest neighbor* of q in S is a point $p(q) \in S$ such that:

$$r^q \stackrel{\text{def}}{=} d(q, p(q)) \leq d(q, p) \qquad \forall p \in S$$

An *optimal* correct nearest neighbor (C-NN) index-based algorithm has first been described for the PMR-Quadtree [16] and then generalized to work with *any* (either multi-dimensional or metric) index tree that is based on a recursive and conservative decomposition of the space [5], thus matching the following generic structure. Each *node* N (usually mapped to a disk page) in the tree corresponds to a *data region*, $Reg(N) \subseteq \mathcal{U}$. Node N stores a set of entries, each entry pointing to a child node N_c and including the specification of $Reg(N_c)$. All indexed feature values are stored in the leaf nodes of the tree, and those in the sub-tree rooted at N are guaranteed to stay in $Reg(N)$.

The `C-NN Optimal` algorithm in Figure 1 uses a priority queue, `PQ`, of references to nodes of the tree, which are kept ordered by increasing values of $d_{min}(q, Reg(N))$. This ensures the algorithm to be *optimal*, since it only accesses those nodes whose region intersects the *NN ball*

Symbol	Description
$\mathcal{U}$	domain of values
D	space dimensionality
d	distance function
$S \subset \mathcal{U}$	data set
$n = \|S\|$	cardinality of the data set
q	query point $q \in \mathcal{U}$
$\mathcal{B}_r(q)$	r-ball of point q
$p(q)$	nearest neighbor of point q
r^q	distance between q and $p(q)$
N	node of a tree
$Reg(N)$	data region corresponding to N
$d_{\min}(q, R)$	minimum dist. between q and region R
ϵ	accuracy (relative error)
ϵ_{eff}	effective relative error
δ	confidence
$F_q(x)$	relative distance distribution of q
$G_q(x)$	distribution of the nearest neighbor of q
r^q_δ	δ-radius of point q

Table 1. Summary of relevant notation.

$\mathcal{B}_{r^q}(q)$ [5]. Note that the computation of $d_{min}(q, Reg(N))$ is the only part of the algorithm that depends on the specific index at hand. The search is stopped at line 5 when the first region in the queue cannot contain any point closer to q than the current nearest neighbor, whose distance from q is r, i.e. $d_{min}(q, Reg(N)) \geq r$.

Algorithm `C-NN Optimal`

Input: index tree $\mathcal{T}$, query object q;
Output: object $p(q)$, the nearest neighbor of q;

1. Initialize PQ with a pointer to the root node of $\mathcal{T}$;
2. Let $r = \infty$;
3. While PQ $\neq \emptyset$ do:
4. Extract the first entry from PQ, referencing node N;
5. If $d_{min}(q, Reg(N)) \geq r$ then exit, else read N;
6. If N is a leaf node then:
7. For each point p_i in N do:
8. If $d(q, p_i) < r$ then: Let $p(q) = p_i$, $r = d(q, p_i)$;
9. else: // N is an internal node
10. For each child node N_c of N do:
11. If $d_{min}(q, Reg(N_c)) < r$:
12. Update PQ performing an ordered insertion of the pointer to N_c;
13. End.

Figure 1. Optimal algorithm for C-NN search.

Although "optimal", above algorithm is effective only when the number of dimensions is relatively low (i.e. $D \leq 10$) after which a sequential scan becomes competitive. This is because in spaces with an intrinsic high-D the distance r^q of the NN of q is "large", and this implies that the probability that a data region intersects the NN ball $\mathcal{B}_{r^q}(q)$ approaches 1 [24].

In order to reduce the complexity of C-NN search, several alternatives have been considered to support *approximate* similarity queries, i.e. queries that are not guaranteed to return the NN of the query point. Here we concentrate on the relevant case of *approximately correct* NN (AC-NN) queries, which, given a value for the *accuracy* parameter (relative error) ϵ, can return any point $p' \in S$ such that:

$$d(q, p') \leq (1 + \epsilon) r^q$$

Point p' is called a $(1+\epsilon)$-approximate NN of q. Above algorithm can be adapted to support AC-NN queries by substituting $r/(1+\epsilon)$ for r at lines 5 and 11. Clearly, when $\epsilon = 0$ one turns back to the usual C-NN search.

Example 1 Refer to Figure 2, where the space is $(\Re^2, L_2)$, i.e. the real plane with the Euclidean distance. We assume that points are indexed by an M-tree, for which regions are *balls*, i.e. $Reg(N) = \mathcal{B}_{r_N}(p_N)$,[1] and $d_{min}(q, Reg(N)) = \max\{d(q, p_N) - r_N, 0\}$.

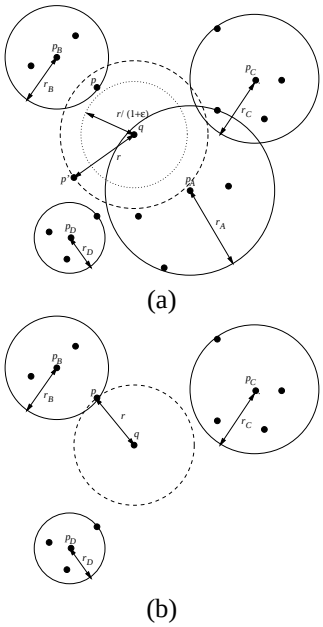


Figure 2. C-NN and AC-NN search in $(\Re^2, L_2)$.

In Figure 2 (a) the current NN is p', $r = d(q, p')$, and the queue contains pointers to nodes A, B, and C, to be visited in this order. Since nothing changes with node A, the C-NN algorithm reads node B and discovers that $d(q, p) < r$, thus setting $r = d(q, p)$ (Figure 2 (b)). At this point, since $d_{min}(q, Reg(C)) > r$ holds, the C-NN search is stopped.

[1] The actual "shape" of M-tree regions depends on the specific metric space $(\mathcal{U}, d)$. For instance, regions are "diamonds" in $(\Re^2, L_1)$, circles in $(\Re^2, L_2)$, and squares in $(\Re^2, L_\infty)$.

On the other hand, the AC-NN algorithm, before reading node B, discovers that $d(q, p_B) - r_B > r/(1+\epsilon)$ and stops, thus returning point p' for which $d(q, p') < (1+\epsilon)d(q, p)$ holds. □

Performance of the AC-NN algorithm depends on the choice of ϵ. Intuitively, the higher ϵ is, the faster the algorithm is expected to run. However, this can have a negative effect on the quality of the result, that is, on the *effective* error.

Definition 1 *The* effective (relative) error, ϵ_{eff}, *of an approximate (not necessarily AC) NN algorithm that returns a point p' whose distance from q is r is defined as:*

$$\epsilon_{eff} = \frac{r}{r^q} - 1$$

By definition, AC-NN algorithms guarantee that $\epsilon_{eff} \leq \epsilon$, since $r \leq (1+\epsilon)r^q$. □

Experimental results in [2] show that $\epsilon_{eff} \ll \epsilon$ usually holds, with ratios of the order of $0.01 \ldots 0.03$. This fact is only apparently positive, since it implies that users cannot directly control the actual quality of the result, rather only a much-higher upper bound. Furthermore, even if performance improvements are obtainable in low-D spaces, the cost grows exponentially with D in the worst case.[2] Some intuition on the complexity of AC-NN queries can be obtained in the case of indexes that allow data regions to overlap, such as the R*-tree and the M-tree. In this case a lower bound on the cost of an AC-NN query, *regardless of the value of* ϵ, is given by the number of data regions that enclose the query point q. Indeed, if $q \in Reg(N)$ then $d_{min}(q, Reg(N)) = 0$ and node N has necessarily to be accessed (see node A in Figure 2). Figure 3 confirms that the fraction of such regions grows with D and soon reaches a limit beyond which sequential scan becomes more convenient.

3. PAC similarity queries

A basic observation to go beyond limitations of AC-NN queries concerns the very nature of a similarity search process. According to our view, this can be conceptually split into two phases:

Locating: This first phase just consists in determining the result, that is, retrieving the point that will be eventually returned by the algorithm.

Stopping: This phase does not change, by definition, the result, yet it is needed to determine that what discovered so far is a $(1+\epsilon)$-approximation of the NN.

[2] In [2] this is derived by means of combinatorial arguments applied to the BBD-tree, a multi-dimensional index structure with non-overlapping data regions.

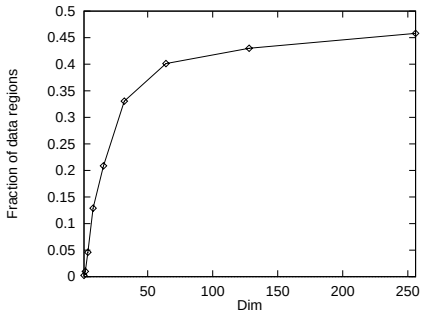


Figure 3. Fraction of data regions containing the query point q as a function of space dimensionality. Euclidean distance, $n = 10^4$ objects indexed by an M-tree.

Figure 4 (a) shows the total (i.e. "locating" plus "stopping") cost, expressed as the number of distance computations executed by the AC-NN algorithm implemented in the M-tree, whereas in Figure 4 (b) we plot the ratio of the "locating cost" to the total cost.

Although the performance rapidly deteriorates as D grows, it can be seen that locating a $(1+\epsilon)$-approximate NN is, in itself, a relatively easy task, whose complexity indeed *decreases* with space dimensionality. This is due to the reduction of the variance of the distances to the query object, which is responsible for the dimensionality curse. We conclude that the hard problem in high-D approximate search is to determine *how to stop*, and that *most of the time spent in an AC-NN search is wasted time*, during which no improvement is obtained.

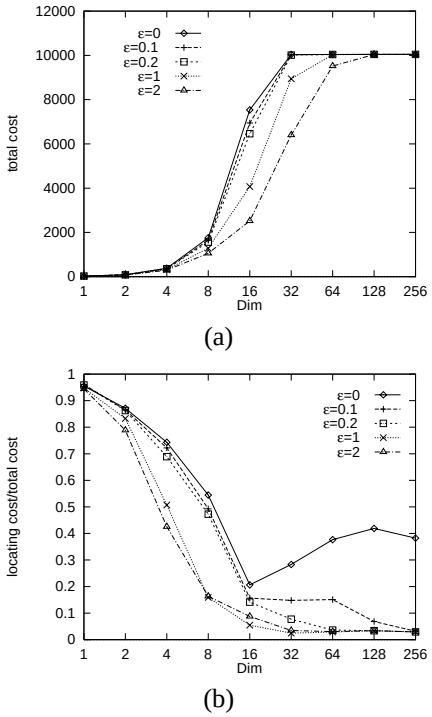
The new approach to similarity search we propose considers a *probabilistic* framework, according to which it is admissible that the result can exceed the error bound ϵ with a certain probability δ. This leads to what we call PAC-NN queries.

Definition 2 *Given a data set S, a query point q, an accuracy parameter ϵ, and a* confidence *parameter $\delta \in [0, 1)$, the result of a PAC-NN (*probably approximately correct*) query is a point $p' \in S$ such that the probability that p' is inside the $\mathcal{B}_{(1+\epsilon)r^q}(q)$ ball is at least $1 - \delta$, that is, $\Pr\{p' \notin \mathcal{B}_{(1+\epsilon)r^q}(q)\} \leq \delta$ or, equivalently:*

$$\Pr\{\epsilon_{eff} > \epsilon\} \leq \delta$$

The result of a PAC-NN query is said to be a $(1+\epsilon; \delta)$-approximate nearest neighbor of q. □

The confidence parameter δ aims to avoid searching "too close" to the query point (this will be made precise in Section 3.2). This exploits the observations that r^q is "large" in high-D spaces and that, nonetheless, stopping an AC-NN



(a)

(b)

Figure 4. (a) Total cost (no. of distance computations) of AC-NN search; (b) Ratio of locating cost to total cost. $n = 10^4$, Euclidean distance, uniform distribution.

search remains a difficult task. A further advantage is that in principle it is possible to choose δ so as to have $\epsilon_{eff} \approx \epsilon$, thus avoiding the mismatch proper of AC-NN algorithms for which $\epsilon_{eff} \ll \epsilon$. This will be investigated in Section 5. Finally, since PAC-NN queries still use ϵ, "locating" is guaranteed to remain a relatively easy task.

3.1. The distance distribution

PAC-NN algorithms need some information about r^q in order to provide a probabilistic guarantee on the quality of the result. Our solution exploits results from [11, 9] on the estimation of search costs for similarity queries on metric spaces. For this, it is adequate to consider *probability metric spaces*, $\mathcal{M} = (\mathcal{U}, d, \mu)$, where μ is a measure of probability over $\mathcal{U}$ [11]. To help intuition, we slightly abuse terminology and also call μ the *data* distribution over $\mathcal{U}$. The models in [11] and [9] show that the costs for determining the NN of q can be accurately predicted if one knows the *relative distance distribution of* q, formally defined as:

$$F_q(x) = \Pr\{d(q,p) \leq x\} \tag{1}$$

where p is distributed according to μ. Intuitively, $F_q(x)$ represents the fraction of objects in $\mathcal{U}$ whose distance from q does not exceed x.

In [11] we have also demonstrated that the *distribution of the nearest neighbor* of q with respect to a data set of size n is given by

$$G_q(x) \stackrel{\text{def}}{=} \Pr\{r^q \leq x\} = 1 - (1 - F_q(x))^n \tag{2}$$

Example 2 Consider the metric spaces $l^D_{\infty,U} = ([0,1]^D, L_\infty, U)$, where points are uniformly (U) distributed over the D-dimensional unit hypercube, and the distance is measured by the L_∞ "max" metric, $L_\infty(p_i, p_j) = \max_k\{|\ p_i[k] - p_j[k]\ |\} \leq 1$. When the query point coincides with the "center" of the space, $q^{cen} = (0.5, \ldots, 0.5)$, it is immediate to derive that $F_{q^{cen}}(x) = (2x)^D$, thus $G_{q^{cen}}(x) = 1 - (1 - (2x)^D)^n$. On the other hand, when the query point is one of the 2^D corners of the hypercube, it is $F_{q^{cor}}(x) = x^D$ and $G_{q^{cor}}(x) = 1 - (1 - x^D)^n$. □

3.2. Stopping the search in PAC-NN algorithms

The basic idea of PAC-NN algorithms is to avoid to search in a region that, according to $G_q(\cdot)$, is reputed to be "too small" to contain at least a point. How the δ confidence parameter is related to the volume of such region is formalized by the following definition.

Definition 3 *Given a data set S of n points, a query point q with distance distribution $F_q(\cdot)$, and a confidence parameter δ, the* δ-radius *of q, denoted r^q_δ, is the maximum value of distance from q for which the probability that there exists at least a point $p \in S$ with $d(q,p) \leq r^q_\delta$ is not greater than δ, that is, $r^q_\delta = \sup\{r \mid \Pr\{\exists p \in S : d(q,p) \leq r\} \leq \delta\} = \sup\{r \mid \Pr\{G_q(r) \leq \delta\}$. If $G_q(\cdot)$ is invertible, then r^q_δ can be expressed as:*

$$r^q_\delta \stackrel{\text{def}}{=} G^{-1}_q(\delta) \tag{3}$$

Example 3 For the metric spaces $l^D_{\infty,U}$, when the query point is $q^{cen} = (0.5, \ldots, 0.5)$ it is derived (see Example 2) that

$$r^{q^{cen}}_\delta = G^{-1}_{q^{cen}}(\delta) = \frac{1}{2}\left(1 - (1-\delta)^{1/n}\right)^{1/D} \tag{4}$$

For instance, when $D = 50$ and $n = 10^6$, if we set $\delta = 0.01$ then $r^{q^{cen}}_{0.01} \approx 0.346$ results. This is to say that with probability at least 99% the hypercube centered on q^{cen} with side 2×0.346 is empty. □

The δ-radius is the basis to determine a stopping condition with probabilistic quality guarantees.

Lemma 1 *Given a data set S of n points, a query point q with distance distribution $F_q(\cdot)$, an accuracy parameter ϵ,*

and a confidence parameter δ, let p' be the closest point to q discovered so far by a PAC-NN algorithm, and let $r = d(q, p')$. If

$$r \leq (1+\epsilon) r^q_\delta \stackrel{\text{def}}{=} r^q_{\delta,\epsilon} \quad (5)$$

then p' is a $(1+\epsilon;\delta)$-approximate NN of q. □

Proof: By definition of PAC-NN queries, it has to be shown that $\Pr\{\epsilon_{eff} > \epsilon\} \leq \delta$, that is, $\Pr\{r/r^q - 1 > \epsilon\} = \Pr\{r^q < r/(1+\epsilon)\} \leq \delta$. Since the last probability equals $G_q(r/(1+\epsilon))$ and $r/(1+\epsilon) \leq r^q_\delta = G_q^{-1}(\delta)$, from the monotonicity of $G_q(\cdot)$ it follows that $G_q(r/(1+\epsilon)) \leq G_q(G_q^{-1}(\delta)) = \delta$. □

The stopping rule (5) provides a simple interpretation of the behavior of PAC-NN algorithms. Given a value of δ, the algorithm first determines the δ-radius r^q_δ, then stops the search as soon as it finds a point p' such that $d(q,p')/(1+\epsilon) \leq r^q_\delta$. Thus, the algorithm will *avoid searching points within the $\mathcal{B}_{r^q_\delta}(q)$ ball*, which is empty with probability at least $1-\delta$. It is indeed this phenomenon that is not exploited at all by C-NN and AC-NN algorithms.

3.3. When are PAC-NN queries meaningful?

After [7], this section addresses an important conceptual issue, concerning the very reason to be of (approximate) NN search. This is an important point, since in [7] it is clearly demonstrated that, under specific conditions related to $F_q(\cdot)$, the NN problem can lose interest. This happens when the distance from q to its NN is comparable to the distance from q to its "farthest neighbor" in the data set. The most well-known case for which this holds are high-D Euclidean spaces with a uniform distribution of data points (this case has been extensively analyzed in [24]). Clearly, in such situations not only C-NN search is meaningless, but also AC-NN and PAC-NN queries are of no interest.

The scenarios we consider are clearly those for which approximate NN search *is* meaningful, yet C-NN and AC-NN algorithms would perform poorly. This holds, say, for the metric spaces $l^D_{p,U} = ([0,1]^D, L_p, U)$ when D is in the range from 20 to 100 or something more. For such dimensionalities the performance of known algorithms deteriorates, yet the variance of distances still makes the search meaningful.

Figure 5 aims to support the above claims and to provide a graphical intuition on how PAC-NN algorithms work. The figure shows graphs of both $F_q(\cdot)$ and $G_q(\cdot)$, together with values of δ and ϵ. When the two distributions are quite well separated (as it happens in the scenarios we focus on), ϵ and δ can be chosen so that the value of $(1+\epsilon)r^q_\delta$ stays well on the left of the zone where $F_q(\cdot)$ sharply increases, that is where most distance values are concentrated. This is also to say that in this case the result of the PAC-NN query is indeed meaningful.

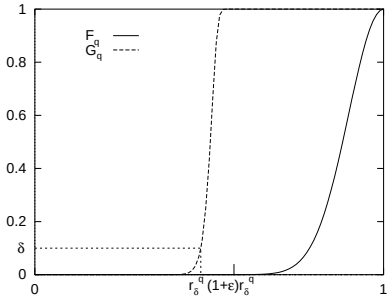


Figure 5. How $F_q(\cdot)$, $G_q(\cdot)$, ϵ, and δ interact in PAC-NN search.

4. The PAC-NN sequential algorithm

The PAC-NN sequential algorithm is suitable when the data set is stored as a sequential file and no index is available. Note that, regardless of ϵ, an AC-NN algorithm would necessarily scan the whole file, thus approximation alone (without δ) would be hopeless.

Given a file of n records/points and a query point q, our algorithm reads the records one by one, and stops as soon as it finds a point p' for which $d(q,p') \leq r^q_{\delta,\epsilon}$ holds. The expected cost, measured as the number of distance computations (*probes*), is estimated by considering a *random sampling process with repetitions* (i.e. a point can be probed more than once). This is an adequate model as long as there is no correlation between the distances of the points to q and their positions in the file, n is large, and the estimated cost is (much) lower than n. On the other hand, when the analysis derives that the cost is comparable to n, then predictions deviate from the actual performance and only provide a (non-tight) upper bound of the cost.

The search process can be analyzed by observing that the cost M is a *geometric* random variable,[3] where the probability of success of a single probe is given by $F_q(r^q_{\delta,\epsilon})$. From this it immediately follows that the expected value of M is simply the inverse of the probability of success at each probe:

$$E[M] = \frac{1}{F_q(r^q_{\delta,\epsilon})} = \frac{1}{F_q((1+\epsilon)G_q^{-1}(\delta))} \quad (6)$$

Note that, since $E[M] = 1/F_q(r^q_{\delta,\epsilon})$ it follows that *varying δ and ϵ will not influence the search cost as long as $r^q_{\delta,\epsilon}$ stays constant*.

[3] This is because we have assumed a "sampling with repetitions" process.

Example 4 Refer to Example 3. By substituting the value of $r_\delta^{q\,^{c\epsilon n}}$ given by Eq. 4 into Eq. 6, it is derived:

$$E[M] = \frac{1}{(1+\epsilon)^D (1-(1-\delta)^{1/n})} \quad (7)$$

Experimental results shown in Table 2 are in line with the analysis. This, as expected, breaks down when $E[M] \ll n$ does not hold, whereas estimates are quite accurate in the other cases.[4] When $\epsilon \geq 0.2$, PAC-NN reduces to randomly sampling a single object, that is, NN search becomes meaningless. □

Theoretical analysis of the effective error is somewhat more involved. For space reasons, we just present the final result and omit all the intermediate steps. The distribution of the effective error is derived to be:

$$\Pr\{\epsilon_{eff} \leq x\} = 1 - G_q(r^q_{\delta,\epsilon}/(1+x)) + \int_0^{r^q_{\delta,\epsilon}/(1+x)} \frac{F_q((1+x)y) - F_q(y)}{F_q(r^q_{\delta,\epsilon}) - F_q(y)}\, g_q(y)\, \mathrm{d}y \quad (8)$$

where $1 - G_q(r^q_{\delta,\epsilon}) = \Pr\{\epsilon_{eff} = 0\}$, g_q is the density of G_q, and the denominator in the integral "normalizes" the possible distances to those admissible when $r^q = y$ ($y \leq r^q_{\delta,\epsilon}$), that is, $[y, r^q_{\delta,\epsilon}]$.

Equations 6 and 8 completely characterize the trade-off between accuracy and cost for the sequential case. Table 3 shows some statistics on the effective error distribution for uniformly distributed data sets.

δ	ϵ_{eff} (avg)	ϵ_{eff} (max)	$\epsilon_{eff} > \epsilon$ (% of cases)
0.01	0.087	0.234	1.79
0.05	0.135	0.304	2.95
0.10	0.144	0.304	6.03
0.20	0.179	0.343	17.95

Table 3. Statistics on the effective error. $\epsilon = 0.2, n = 10^5, D = 40$.

As a final observation, asymptotic analysis of Eq. 7 reveals that $E[M]$ grows like $O(n\delta^{-1}(1+\epsilon)^{-D})$, thus linearly with n. From this we conclude that the PAC-NN sequential algorithm is not really suitable for (very) large data sets, especially when ϵ and δ have both small values. We remark, however, that this depends on the specific metric spaces (in particular on the uniform distribution) used in the example.

[4] The table simply reports n if $E[M] \geq n$ results from Eq. 7.

5. Experimenting the index-based PAC-NN algorithm

The PAC-NN algorithm for index-based search is described in Figure 6. As with the AC-NN algorithm, lines 5 and 12 consider $r/(1+\epsilon)$ in place of r, whereas the stopping condition based on r^q_δ is at line 8. No other changes to the logic of `C-NN Optimal` are needed.

Algorithm `PAC-NN`

Input: index tree $\mathcal{T}$, query object q, ϵ, δ, $F_q(\cdot)$;
Output: object p', a $(1+\epsilon;\delta)$-approximate NN of q;

1. Initialize `PQ` with a pointer to the root node of $\mathcal{T}$;
2. Compute r^q_δ; Let $r = \infty$;
3. While `PQ` $\neq \emptyset$ do:
4. Extract the first entry from `PQ`, referencing node N;
5. If $d_{min}(q, Reg(N)) \geq r/(1+\epsilon)$ then exit, else read N;
6. If N is a leaf node then:
7. For each point p_i in N do:
8. If $d(q,p_i) < r$ then:
   Let $p' = p_i$, $r = d(q,p_i)$; If $r \leq (1+\epsilon) r^q_\delta$ then exit;
9. else: // N is an internal node
10. For each child node N_c of N do:
11. If $d_{min}(q, Reg(N_c)) < r/(1+\epsilon)$:
12. Update `PQ` performing an ordered insertion of the pointer to N_c;
13. End.

Figure 6. The index-based PAC-NN algorithm.

In the following we present experimental results on the performance of the PAC-NN algorithm, and compare it with AC-NN search. All the experiments are run by indexing the data set with an M-tree (the node size is 8 KB), executing 100 queries with the same distribution of the data set, and then averaging results. For simplicity, we *do not* use the distance distribution $F_q(\cdot)$ of the query point, rather we approximate it with the *overall* distance distribution, $F(\cdot)$, obtained by sampling the data set at hand. Although this can introduce some estimation error, from a practical point of view differences are minimal, as demonstrated in [11]. Alternatively, a better approximation of $F_q(\cdot)$ can be obtained by using the techniques described in [9], which require to store the distance distribution of a set of "representative points"[5] and then to combine them at query time. The sample size is between 1% (for larger data sets) and 10% of the data set size, and $F(\cdot)$ is represented by a 100-bins equi-width histogram. For space reasons we only present results where the "cost" is measured as the number of distance computations (CPU cost). I/O costs (page reads) are

[5] These are called *witnesses* in [9].

$\epsilon \downarrow$ $\delta \rightarrow$	0.01		0.05		0.1		0.2		0.5	
0.01	10^6	(982869)	10^6	(952869)	10^6	(843738)	10^6	(663542)	533381	(391212)
0.05	756640	(470758)	148255	(154617)	72176	(71741)	34079	(33479)	10971	(11944)
0.10	7221	(7138)	1415	(1410)	689	(683)	326	(327)	105	(107)
0.20	2	(2)	1	(1)	1	(1)	1	(1)	1	(1)

Table 2. Expected costs and (in parentheses) actual results of the PAC-NN sequential algorithm for a "center" query point. $n = 10^6$, $D = 100$. Results are averaged over 10^4 data sets.

not shown, since they follow the same trend of CPU costs, up to a scale factor that depends on the average number of entries in each node.

5.1. Synthetic data sets

We start with data sets consisting of $n = 10^5$ uniformly distributed objects. For high-D spaces, Figure 7 shows how the cost varies with D, for different values of δ and $\epsilon = 0.1$. It is clear that the AC-NN algorithm ($\delta = 0$) is completely useless at such high dimensionalities, whereas the cost of PAC-NN queries remains quite low (note that the cost axis uses a logarithmic scale).

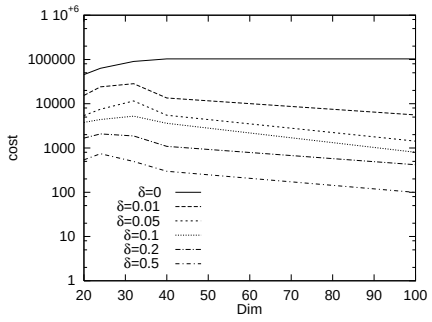


Figure 7. Uniform data sets. $\epsilon = 0.1$.

Figure 8 shows results for the case $D = 40$, from which it is evident that ϵ alone is ineffective, whereas the cost is highly dependent on ϵ when $\delta > 0$.

In low- to medium-D spaces both PAC-NN and AC-NN algorithms can be profitably used, with Figure 9 showing typical trends. As for the cost, Figure 9 (a) shows that ϵ alone has a minimal influence.[6] As for the effective error, Figure 9 (b) confirms that PAC-NN search can exceed the error bound, the average amount depending on the choice of δ.

[6]This does not contradict results in [2], since in that paper much higher values of ϵ are considered, up to $\epsilon = 10$.

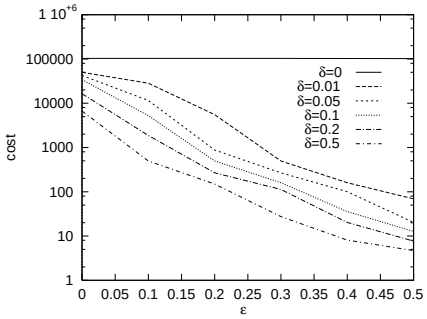
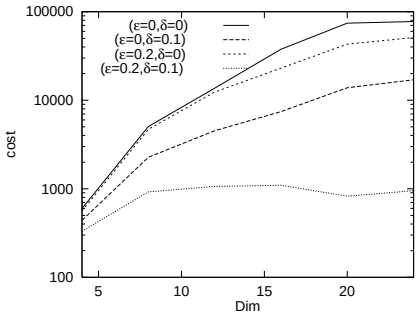
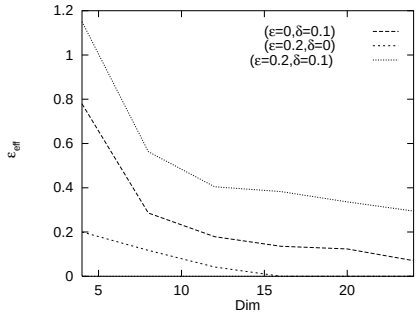


Figure 8. Uniform data sets. $D = 40$.



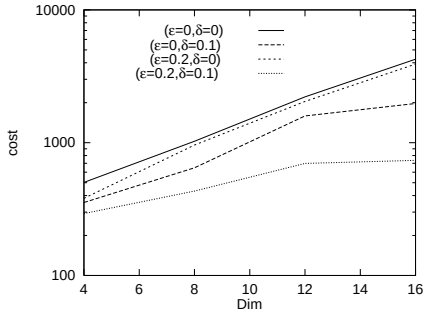
(a)



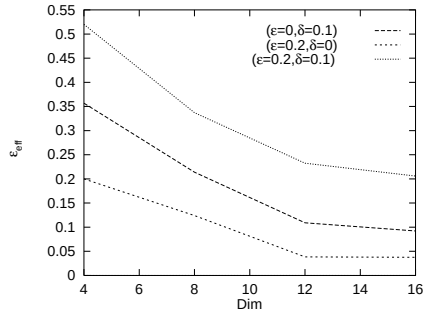
(b)

Figure 9. Low- and medium-D spaces. (a) Cost; (b) Effective error.

Figure 10 analyzes the case of *clustered* data sets. Each data set consists of D-dimensional vectors normally-distributed (with $\sigma = 0.1$) in 10 clusters over the unit hypercube, with clusters' centers randomly chosen. Comparing with Figure 9, it can be observed that both costs and effective errors are now reduced. This confirms that also for PAC-NN queries uniformly distributed data sets are harder to deal with.



(a)



(b)

Figure 10. Clustered data sets. (a) Cost; (b) Effective error.

5.2. Real data sets

Here we present results of experiments with two real-life data sets. The first data set consists of $11,648$ 45-dimensional feature vectors extracted from color images. Each image is first decomposed into five overlapping parts, then from each part a 9-dimensional feature vector is extracted using the first three moments of the distribution of the 3 HSV color channels, as described in [23]. The Euclidean distance is used to compare the so-obtained 45-dimensional vectors. In general, as Figure 11 shows, average costs are reduced up to 50% by using the PAC-NN algorithm. Note that, because of the different distance distribution, higher values of ϵ, as compared to those shown in Section 5.1 for uniform and clustered data sets, are now used.

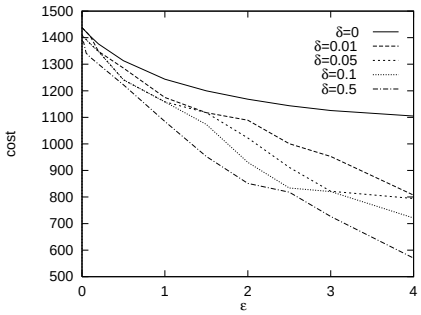


Figure 11. Image data set. Cost vs. ϵ.

As for the quality of the result, Figure 12 shows how, for a given ϵ value, accuracy can be controlled by varying δ.

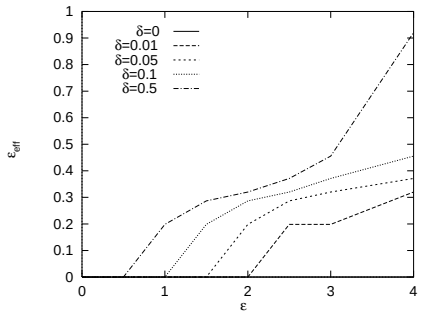


Figure 12. Image data set. Effective error vs. ϵ.

It has to be remarked that in many cases, even using quite high values of ϵ and δ, the PAC-NN algorithm is able to return the correct NN. As an example, consider Figure 13, where the query image is shown on the left and its NN is in the middle. The correct NN is also retrieved by the PAC-NN algorithm as long as $\epsilon < 1$ and $\delta < 0.5$, whereas for higher values of the parameters the PAC-NN search retrieves the approximate NN shown on the right.

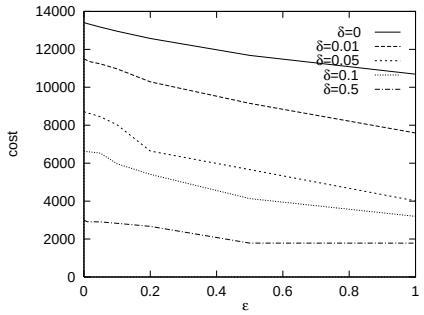
(a) (b) (c)

Figure 13. (a) Query image; (b) The NN of (a); (c) Approximate NN of (a), $(\epsilon, \delta) = (1, 0.5)$.

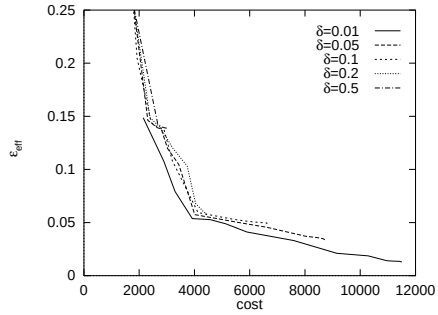
The second data set we experimented with was given us by B.S. Manjunath [19] and consists of $275,465$ 60-dimensional vectors. Each vector contains texture information extracted from a tile of size 64×64 that is part of a

large aerial photograph (there are 40 airphotos in the data set). Each tile is analyzed by means of 30 Gabor filters, and for each filter the mean and the standard deviation of the output are stored in the feature vector.

Figure 14 (a) shows how the cost varies with δ and ϵ, and Figure 14 (b) makes evident the trade-off existing between cost and accuracy. The most important observation, which has general validity and is not restricted to the specific data set, is that ϵ_{eff} *is almost insensitive to the specific choice of* ϵ *and* δ *values*, provided the two parameters are chosen in an appropriate way. This has an explanation similar to the one given for the sequential case (Eq. 6), in that performance mainly depends on the value of $r^q_{\delta,\epsilon}$, rather than on the single ϵ and δ values.



(a)



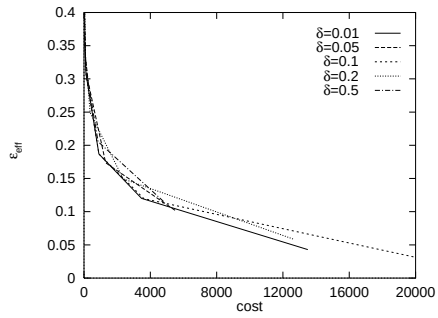
(b)

Figure 14. Airphoto data set. (a) Cost vs. ϵ; (b) Effective error vs. cost.

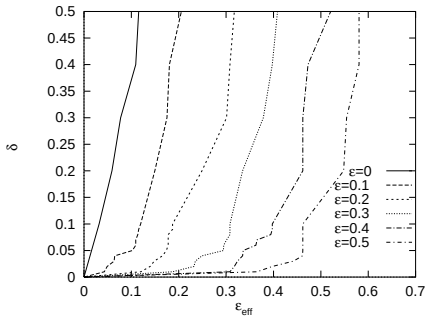
5.3. Tuning PAC-NN search

Since we have not developed yet a model to predict the cost of the PAC-NN index-based algorithm, here we provide some guidelines on how parameters of PAC-NN queries can be chosen in order to achieve a certain trade-off between the actual quality of the result, i.e. ϵ_{eff}, and the cost.

Consider the case of, say, a 40-dimensional data set with 10^5 uniformly distributed points. Figure 15 (a) relates the effective error to the cost and confirms what observed from Figure 14 (b), that is, the trade-off between cost and accuracy is practically independent of the specific ϵ and δ values. Consider also Figure 15 (b), where the values of δ that guarantee to have $\epsilon_{eff} \approx \epsilon$ are shown, for several values of the ϵ parameter.



(a)



(b)

Figure 15. Uniform data sets. $D = 40$. (a) Effective error vs. cost; (b) δ vs. effective error.

A realistic scenario for an user issuing PAC-NN queries on a data set for which statistics like these are available is summarized in Figure 16. The user can either specify a value for the *effective* relative error or limit the cost to be paid. In the first case the system can first choose $\epsilon \approx \epsilon_{eff}$ and then, from Figure 15 (b), the appropriate value for δ. In the second case these steps have to be preceded by an estimate of ϵ_{eff} based on Figure 15 (a). As an example, in order to have $\epsilon_{eff} = 0.2$, Figure 15 (a) predicts a cost in the range $800..1400$, and Figure 15 (b) suggests to use $\delta \approx 0.1$.

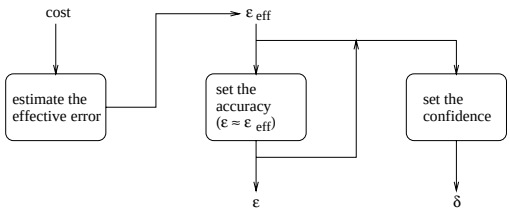


Figure 16. How ϵ and δ values can be chosen so as to to yield a given performance level.

5.4. Sequential vs. index-based PAC-NN search

We conclude by comparing the sequential and the index-based PAC-NN algorithms. Since, as discussed at the beginning of Section 5, the index uses the overall distance distribution (rather than the one specific for the query point at hand) to determine the δ-radius, the same procedure was used for the sequential search, in order to guarantee fairness of comparison.

Table 4 presents results for a 40-dimensional data set with 10^5 uniformly distributed points. The improvement obtainable through indexing is always between 1-2 orders of magnitude, and only reduces when the search becomes easier (i.e. for higher values of ϵ and/or δ, not shown in the table), in which case however NN queries lose interest, as discussed in Sections 3.3 and 4.

Finally, we evaluated the query response time as a function of the effective error on the airphoto data set. Experiments were run on a Linux PC with a Pentium III 450 MHz processor, 256 MB of main menory, and a 9 GB disk. It should be remarked that the average response time for *correct* NN queries is 107 seconds for a sequential scan, and 26.3 seconds for an index-based search. As Figure 17 shows, index-based search consistently outperforms the sequential PAC-NN scan, the difference always being about one order of magnitude. For higher values of ϵ_{eff}, not shown in the figure, the stopping condition is satisfied by a large fraction of the points in the data set and therefore the response time for both algorithms is considerably lower.

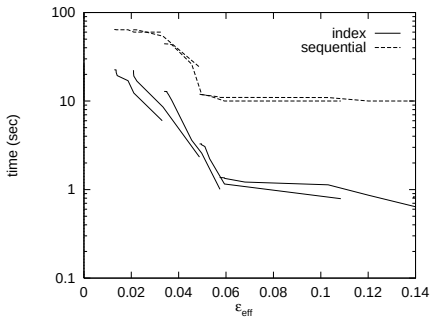


Figure 17. Airphoto data set. Elapsed time vs. effective error.

6. Conclusions

In this work we have introduced a new paradigm for *approximate* similarity queries, in which the error bound ϵ can be exceeded with a certain probability δ and both ϵ and δ can be chosen on a per-query basis. We have analytically and experimentally shown that PAC-NN queries can lead to remarkable performance improvements in high-D spaces, where other algorithms would fail because of the "dimensionality curse". Our algorithms need some prior information on the *distance distribution* of the query point, which, using results in [11], can be however reliably approximated by the *overall* distance distribution of the data set. We have also shown that it is indeed possible to exert an effective control on the quality of the result, thus trading accuracy for cost. This is an important issue that has gained full relevance in recent years [22].

Other approaches, besides the one proposed in [2] and that we have somewhat taken as a reference starting point, exist to support approximate NN search. Indik and Motwani [17] consider a hash-based technique able to return a $(1+\epsilon)$-approximate NN with *constant* probability. Although definitely interesting, this technique is limited to vector spaces and L_p norms, and its preprocessing costs are exponential in $1/\epsilon$, with the drawback that ϵ needs to be known in advance. Also, no possibility to control at query time the probability of exceeding the error bound is given. This is also the case for the solution proposed by Clarkson [13], which applies to exact NN search over generic metric spaces, but whose space requirements depend on the error probability. Finally, Zezula et al. [25] have recently proposed approximate NN search algorithms with good cost performance. However, since the effective error is not bounded by any function of the input parameters, their algorithms do not provide guarantees on the quality of the result.

We have argued and experimentally shown that, even if the "dimensionality curse" can make NN queries meaningless when the distances between the indexed objects and the query objects are all similar [7], there are indeed relevant cases where this is not the case and, at the same time, known algorithms show poor performance. PAC-NN queries and algorithms are best suited to these situations, even if they can be profitably applied also to low-dimensional spaces.

In the future we plan to extend our approach to k-nearest neighbors queries, for which the exact search would retrieve the k best matches of the query object, and to develop a cost model for predicting the performance of the PAC-NN index-based algorithm. Another interesting research issue would be to apply our results to the case of *complex* NN queries, where more than one similarity criterion has to be applied in order to determine the overall similarity of two objects [14, 12].

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *FODO'93*, pages 69–84, Chicago, IL, October 1993.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest

$\epsilon \downarrow \quad \delta \rightarrow$	0.01		0.05		0.1		0.5	
0.1	13498	(93726)	5494	(69704)	3614	(66667)	849	(24741)
0.2	3474	(67548)	1307	(31021)	898	(20741)	108	(4598)
0.3	898	(21232)	257	(4058)	118	(2752)	13	(555)

Table 4. Costs of index-based and (sequential) PAC-NN algorithms. $n = 10^5$**,** $D = 40$**.**

neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[3] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *IEEE Data Engineering Bulletin*, 20(4):3–45, December 1997.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *SIGMOD'90*, pages 322–331, Atlantic City, NJ, May 1990.

[5] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. *PODS'97*, pages 78–86, Tucson, AZ, May 1997.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. *VLDB'96*, pages 28–39, Mumbai (Bombay), India, September 1996.

[7] K. Beyer, J. Goldstein, R. Ramakhrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *ICDT'99*, pages 217–235, Jerusalem, Israel, January 1999.

[8] T. Bozkaya and M. Özsoyoglu. Distance-based indexing for high-dimensional metric spaces. *SIGMOD'97*, pages 357–368, Tucson, AZ, May 1997.

[9] P. Ciaccia, A. Nanni, and M. Patella. A query-sensitive cost model for similarity queries with M-tree. *ADC'99*, pages 65–76, Auckland, New Zealand, January 1999.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *VLDB'97*, pages 426–435, Athens, Greece, August 1997.

[11] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. *PODS'98*, pages 59–68, Seattle, WA, June 1998.

[12] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. *EDBT'98*, pages 9–23, Valencia, Spain, March 1998.

[13] K. L. Clarkson. Nearest neighbor queries in metric spaces. *STOC'97*, pages 609–617, El Paso, TX, May 1997.

[14] R. Fagin. Combining fuzzy information from multiple systems. *PODS'96*, pages 216–226, Montreal, Canada, June 1996.

[15] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[16] G. R. Hjaltason and H. Samet. Ranking in spatial databases. *SSD'95*, pages 83–95, Portland, ME, August 1995.

[17] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *STOC'98*, pages 604–613, Dallas, TX, May 1998.

[18] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. *SIGMOD'97*, pages 369–380, New York, NY, May 1997.

[19] B.S. Manjunath. The airphoto data set. `http:// vivaldi.ece.ucsb.edu/Manjunath/research.htm`.

[20] V. Pestov. On the geometry of similarity search: Dimensionality curse and concentration of measure. Technical Report RP-99-01, School of Mathematical and Computing Sciences, Victoria University of Wellington, New Zealand, January 1999. `http://xxx.lanl.gov/abs/cs.IR/9901004`.

[21] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. *SIGMOD'98*, pages 154–165, Seattle, WA, June 1998.

[22] N. Shivakumar, H. Garcia-Molina, and C. Chekuri. Filtering with approximate predicates. *VLDB'98*, pages 263–274, New York, NY, August 1998.

[23] M. Stricker and M. Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases SPIE*, volume 2420, pages 381–392, San Jose, CA, February 1995.

[24] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *VLDB'98*, pages 194–205, New York, NY, August 1998.

[25] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-trees. *The VLDB Journal*, 7(4):275–293, 1998.