PAC Nearest Neighbor Queries: Using the Distance Distribution for Searching in High-Dimensional Metric Spaces*

Paolo Ciaccia and Marco Patella

DEIS - CSITE-CNR, University of Bologna, Italy
{pciaccia, mpatella}@deis.unibo.it

Abstract. In this paper we introduce a new paradigm for similarity search, called PAC-NN (*probably approximately correct nearest neighbor*) queries, aiming to break the "dimensionality curse" which inhibits current approaches to be applied in high-dimensional spaces. PAC-NN queries return, with probability at least $1 - \delta$, a $(1 + \epsilon)$-approximate NN – an object whose distance from the query q is less than $(1 + \epsilon)$ times the distance between q and its NN. We describe how the *distance distribution* of the query object can be used to determine a suitable *stopping condition* with probabilistic guarantees on the quality of the result, and then analyze performance of both sequential and index-based PAC-NN algorithms. This shows that PAC-NN queries can be efficiently processed even on very high-dimensional spaces and that control can be exerted in order to tradeoff between the accuracy of the result and the cost.

1 Introduction

Similarity queries are a fundamental paradigm for multimedia, data mining, decision support, 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 images [FEF$^+$94], Fourier coefficients from time series [AFS93]), and then measuring the *distance* between feature values, so that similarity search becomes a *nearest neighbor* (NN) query over the feature space. Indexing of feature values, which often are high-dimensional (high-D) vectors, can be done by means of either multi-dimensional trees (e.g. the R-tree [Gut84], the R*-tree [BKSS90], and the SR-tree [KS97]) or *metric* trees (e.g. the M-tree [CPZ97] and the mvp-tree [BÖ97]), the latter only requiring that the distance between feature values is a *metric*, and as such can be used even when no adequate vector representation is possible.

It is nowadays well-known that even for moderately high-D spaces ($D \geq 10$) the NN problem can be very difficult to solve [BGRS99, WSB98]. This phenomenon, traditionally called the "dimensionality curse", is not peculiar to vector spaces, but can also affect more generic metric spaces, as recent mathematical

* This work has been partially supported by MURST and CNR funds.

studies demonstrate [Pes99]. Dimensionality curse is strictly related to the distribution of distances between the indexed objects and the query object [BGRS99]. Intuitively, if these distances are all similar, ie. their *variance* is low, then searching is difficult.

In order to break the dimensionality curse, in this paper we propose a *probabilistic* approach, which allows a NN query to specify two parameters: the *accuracy* ϵ allows for a certain relative error in the result, whereas the *confidence* δ guarantees, with probability $(1-\delta)$, that ϵ will not be exceeded. This generalizes both *correct* (C-NN) and *approximately correct* (AC-NN) NN queries [AMN$^+$], where the latter only consider ϵ and are still plagued by the dimensionality curse.

After reviewing the basic logic of C-NN and AC-NN algorithms and highlighting their limits (Section 2), in Section 3 we introduce PAC (*probably approximately correct*) NN queries, whose basic idea is to avoid searching "too close" to the query object. We then describe how information on the distance distribution can be used to derive a simple and effective *stopping condition* for PAC-NN algorithms. Section 4 provides evaluation for sequential datasets and demonstrates that the complexity of the PAC sequential algorithm is at least $O(n\delta^{-1}(1+\epsilon)^{-D})$, thus still linear in the dataset size n. In Section 5 we evaluate the performance of a PAC algorithm implemented in the M-tree and show how performance can improve up to 2 orders of magnitude. Although we use the M-tree for practical reasons, our results apply to *any* multi-dimensional or metric index tree. We also demonstrate that, for any value of ϵ, δ can be chosen so that the *actual* 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, trading off between accuracy and cost.

2 Preliminaries

We consider that objects' feature values are points of a *metric space* $\mathcal{M} = (\mathcal{U}, d)$, where $\mathcal{U}$ is the domain of values and d is a metric used to measure the distance of points of $\mathcal{U}$. For any real value $r \geq 0$, $\mathcal{B}_r(c) = \{p \in \mathcal{U} \mid d(c,p) \leq r\}$ denotes the r-ball of point c, i.e. the set of points whose distance from c does not exceed r. The minimum distance between a point q and a region $R \subseteq \mathcal{U}$ is defined as $d_{min}(q,R) = \inf\{d(q,p) \mid p \in R\}$. Given a set $S = \{p_1, \ldots, p_n\}$ of n points, and a query point $q \in \mathcal{U}$, the *nearest neighbor* (NN) of q in S is a point $p(q) \in S$ such that $d(q,p(q)) \leq d(q,p), \forall p \in S$.

An *optimal* correct nearest neighbor (C-NN) algorithm has been described in [BBKK97]. It can be used with *any* multi-dimensional and metric index tree which is based on a recursive and conservative decomposition of the space, 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 pointing to a child node N_c and including the description of $Reg(N_c)$. All indexed feature values are stored in the leaf nodes 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 containing references to nodes, which is kept ordered by increasing values of $d_{min}(q, Reg(N))$, that is, the minimum distance between q and a point $p \in Reg(N)$. This ensures the algorithm to be *optimal*, since it only accesses those nodes whose region intersects the *NN ball* $\mathcal{B}_{d(q,p(q))}(q)$ [BBKK97]. If the first region in the queue cannot contain any point closer to q than the current nearest neighbor, then the search is stopped (line 5). This algorithm is effective only when D is low (i.e. ≤ 10), after which a sequential scan becomes competitive. This is because in high-D spaces the distance $d(q, p(q))$ of the NN of q is "large", and the probability that a data region intersects the NN ball $\mathcal{B}_{d(q,p(q))}(q)$ approaches 1 [WSB98].

Algorithm `C-NN Optimal`

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

```
1.  Initialize the priority queue 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 $d_{min}(q, Reg(N)) > r$ 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 $d_{min}(q, Reg(N_c)) < r$ then:
12.                 Update PQ performing an ordered insertion of the pointer to $N_c$;
13. End.
```

Fig. 1. Optimal algorithm for correct NN search.

In order to reduce the complexity of C-NN search, several alternatives have been considered to support *approximate* similarity queries, ie. queries which 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 a value for the *accuracy* parameter (relative error) ϵ, can return any point $p' \in S$ such that:

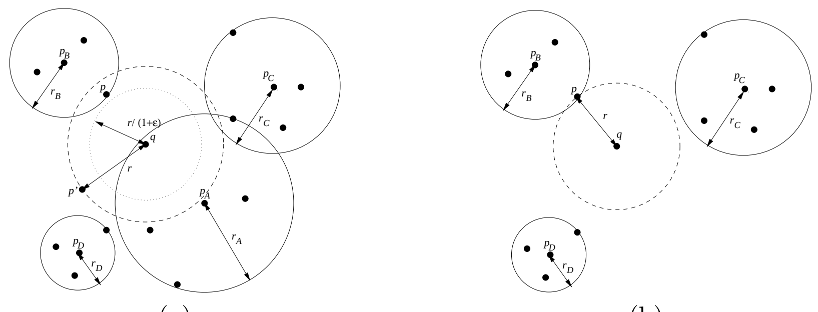
$$d(q, p') \leq (1 + \epsilon) d(q, p(q))$$

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.

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*, $Reg(N) = \mathcal{B}_{r_N}(p_N)$,[14] and $d_{min}(q, Reg(N)) = \max\{d(q, p_N) - r_N, 0\}$.

[14] The actual "shape" of M-tree regions depends on the specific metric space $(\mathcal{U}, d)$.

In Figure 2 (a) p' is the current NN, $r = d(q, p')$, and the queue contains pointers to nodes A, B, C, and D. Since nothing changes with node A, the C-NN algorithm fetches node B from disk and discovers that $d(q, p) < r$, thus setting $r = d(q, p)$ (see Figure 2 (b)). At this point, since $d_{min}(q, Reg(C)) = d(q, p_C) - r_C > r$ holds, the C-NN search is stopped. The AC-NN algorithm, before retrieving node B, discovers that $d(q, p_B) - r_B > r/(1 + \epsilon)$ and therefore stops, thus returning point p' for which $d(q, p') < (1 + \epsilon)d(q, p)$ holds. $\square$



(a) (b)

Fig. 2. C-NN and AC-NN search in $\Re^2$ with L_2.

Performance of the AC-NN algorithm largely depends on the choice of ϵ. Intuitively, the higher ϵ is, the faster the algorithm runs. However, this can have a negative effect on the quality of the result, that is, on the *effective* error. If an approximate (not necessarily AC) NN algorithm returns a point p' whose distance from q is r, the *effective (relative) error*, ϵ_{eff}, is defined as:

$$\epsilon_{eff} = \frac{r}{d(q, p(q))} - 1$$

By definition, the AC-NN algorithm guarantees that $r \leq (1 + \epsilon)d(q, p(q))$, thus $\epsilon_{eff} \leq \epsilon$ holds. Experimental results reported in [AMN$^+$] show that usually it is $\epsilon_{eff} \ll \epsilon$, with ratios typically 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 experimental results show the improvements obtainable from AC-NN search in low-D spaces, the complexity remains exponential in D [AMN$^+$]. In the case of indexes which allow the overlap of data regions (e.g. the R-tree and the M-tree), 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 which enclose the query point q. Indeed, if $q \in Reg(N)$ then $d_{min}(q, Reg(N)) = 0$ and node N cannot be pruned (see node A in Figure 2 (a)). Figure 3 confirms that the fraction of such regions grows with D and soon reaches a limit beyond which a sequential scan would be more convenient.

For instance, regions are "diamonds" in $(\Re^2, L_1)$, circles in $(\Re^2, L_2)$, and squares in $(\Re^2, L_\infty)$.

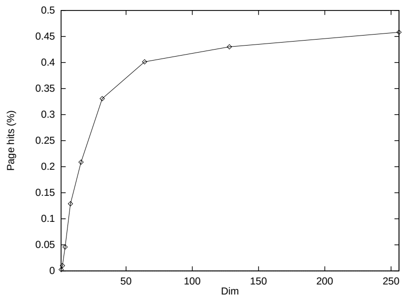


Fig. 3. Percentage of data regions containing the query point q as a function of space dimensionality. Euclidean distance, $n = 10^4$ objects indexed by an M-tree.

3 Probably Approximately Correct 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 phase just consists in determining the result, that is, retrieving the point which will be eventually returned by the algorithm.

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

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.

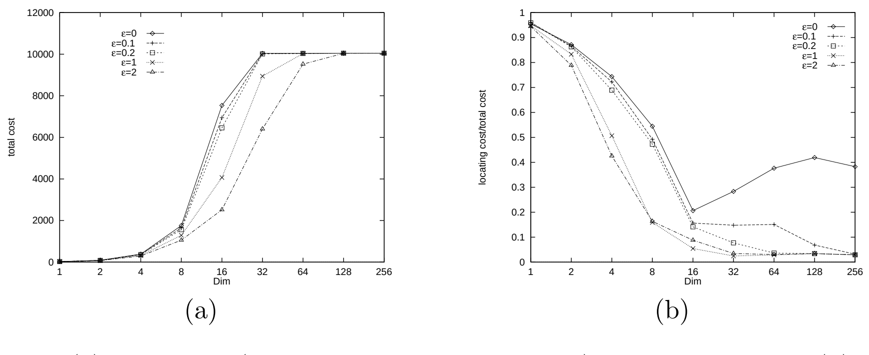


Fig. 4. (a) Total cost (no. of distance computations) of AC-NN search; (b) Ratio of locating cost to total cost. $n = 10^4$, Euclidean distance, uniform data distribution.

Besides confirming that the performance rapidly deteriorates as D grows (Figure 4 (a)), in Figure 4 (b), where the ratio of the "locating cost" to the total cost is graphed, it is shown that locating a $(1 + \epsilon)$-approximate NN is, in itself, a relatively easy task, whose complexity indeed *decreases* with space dimensionality. This is a direct consequence of 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*.

The new approach 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 1. Given a dataset 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 $\mathcal{B}_{(1+\epsilon)d(q,p(q))}(q)$ ball is at least $1-\delta$, that is,

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

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

The confidence parameter δ aims to avoid searching "too close" to the query point. This exploits the facts that $d(q,p(q))$ is "large" in high-D spaces and that, nonetheless, stopping an AC-NN search remains a difficult task. A further advantage is that in principle it is possible to choose δ so as to have $\epsilon_{\mathit{eff}} \approx \epsilon$, thus avoiding the mismatch proper of AC-NN algorithms. Finally, since PAC-NN queries still use ϵ, "locating" is guaranteed to remain a relatively easy task.

PAC-NN algorithms need some information about $d(q,p(q))$ in order to provide a probabilistic guarantee on the quality of the result. Our solution exploits results from [CPZ98a, CNP99] where *random metric spaces*, $\mathcal{M} = (\mathcal{U}, d, \mu)$, are considered, μ being a measure of probability over $\mathcal{U}$. To help intuition, we slightly abuse terminology and also call μ the *data* distribution over $\mathcal{U}$. The models in [CPZ98a, CNP99] show that 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. In [CPZ98a] it is also demonstrated that the *distribution of the nearest neighbor* of q with respect to a dataset of size n is given by

$$G_q(x) \overset{\text{def}}{=} \Pr\{d(q,p(q)) \leq x\} = 1 - (1 - F_q(x))^n \tag{2}$$

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 the "max" metric, $L_\infty(p_i,p_j) = \max_k\{\mid p_i[k] - p_j[k] \mid\} \leq 1$. 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 $G_{q^{cor}}(x) = 1-(1-x^D)^n$. $\square$

3.1 Stopping the Search in PAC-NN Algorithms

The basic idea of PAC-NN search is to avoid to search in a region which, 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 this region is formalized by the following definition.

Definition 2. Given a dataset S of n points, a query point q with distance distribution $F_q(\cdot)$, and a confidence parameter $\delta \in (0,1)$, the *δ-radius* of q, denoted r^q_δ, is the maximum value of distance from q for which the probability that exists at least a point $p \in S$ with $d(q,p) \leq r^q_\delta$ is not greater than δ, that is, $r^q_\delta = \sup\{r \mid \Pr\{\exists p \in S : d(q,p) \leq r\} \leq \delta\}$. If $G_q(\cdot)$ is invertible, r^q_δ can also be more conveniently expressed as:

$$r^q_\delta \overset{\text{def}}{=} G_q^{-1}(\delta) \tag{3}$$

□

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

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

When $D = 50$, $n = 10^6$, and $\delta = 0.01$, then $r^{q^{cen}}_{0.01} \approx 0.346$ results. This means that there is a probability of 99% that the hypercube centered on q^{cen} with side 2×0.346 is empty.

The following lemma establishes the stopping condition for PAC-NN search.

Lemma 3. *Given a dataset 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*

$$r \leq (1+\epsilon) r^q_\delta \tag{5}$$

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

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

Figure 5 provides 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. 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 $d(q,p')/(1+\epsilon)$ does not exceed r^q_δ (see Eq. 5). This corresponds to *avoid searching points within the $\mathcal{B}_{r^q_\delta}(q)$ ball*, which, according to the information conveyed by the distance distribution, is empty with probability at least $1-\delta$. It is indeed this phenomenon, typical

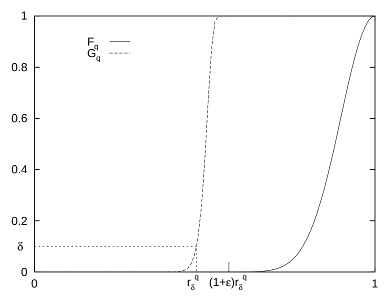


Fig. 5. How $F_q(\cdot)$, $G_q(\cdot)$, ϵ, and δ interact in PAC-NN search.

of high-dimensional spaces, that is not exploited at all by C-NN and AC-NN algorithms.

It is clear that the NN problem loses of interest when the distance from q to its NN is comparable to the distance from q to all other points, as it happens in (very) high-D Euclidean spaces with uniformly distributed points [WSB98]. The scenarios we consider are those for which approximate NN search *is* meaningful, yet C-NN and AC-NN algorithms would fail. This holds, say, for the metric spaces $l_{p,U}^D = ([0,1]^D, L_p, U)$ with $D \in [20, \approx 100]$. If the two distributions in Figure 5 are well separated (as it happens in the cases we focus on), ϵ and δ can be chosen so that $(1+\epsilon)r_\delta^q$ stays well on the left of the zone where $F_q(\cdot)$ sharply increases, i.e. where most distance values are concentrated. This is also to say that the PAC-NN query is indeed meaningful.

4 The PAC-NN Sequential Algorithhm

We first consider the case where the dataset S is stored as a sequential file, thus an AC-NN search would necessarily scan the whole file. The PAC-NN algorithm reads the records one by one, and stops when it finds a point p' such that $d(q,p') \leq (1+\epsilon)r_\delta^q$. The expected cost, measured as the number of distance computations, can be estimated by considering a *random sampling process with repetitions* (ie. a point can be examined more than once). This is adequate 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 only provide a (non-tight) upper bound of cost.

Since the cost M is a *geometric* random variable, where the probability of success of a trial is $F_q((1+\epsilon)r_\delta^q)$, it is $\Pr\{M=m\} = (1-F_q((1+\epsilon)r_\delta^q))^{m-1}F_q((1+\epsilon)r_\delta^q)$, and the expected value of M is simply the inverse of the trial success probability:

$$E[M] = \sum_m m \Pr\{M=m\} = \frac{1}{F_q((1+\epsilon)r_\delta^q)} = \frac{1}{F_q((1+\epsilon)F_q^{-1}(1-(1-\delta)^{1/n}))} \tag{6}$$

As an example, by substituting the value of $r_\delta^{q^{cen}}$ given by Eq. 4 into Eq. 6, it is obtained:

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

Table 1 shows estimates and actual results for $E[M]$ when the number of points is $n = 10^6$ and $D = 100$.[15] It can be observed that the ϵ parameter has a strong influence on the performance. Also the effects of the δ confidence parameter are in line with expectation, even if at this point it is not clear yet which is its influence on the effective error. As expected, the analysis breaks down below a certain value of ϵ, whereas estimates are quite good in the other cases. When $\epsilon \geq 0.2$, PAC-NN reduces to randomly sampling a single object, this is to say that in this case NN search is indeed meaningless. Asymptotic analysis of Equation 7 reveals that $E[M]$ grows like $O(n\delta^{-1}(1+\epsilon)^{-D})$, thus linearly with n. From this we conclude that the sequential algorithm is not suitable for (very) large datasets, especially when ϵ and δ have both small values.

$\epsilon \downarrow \quad \delta \rightarrow$	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 1. Expected costs and (in parentheses) actual results of the PAC-NN sequential algorithm.

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 11 consider $r/(1+\epsilon)$ in place of r, whereas the stopping condition based on r_δ^q is at line 8. No other changes to the logic of `C-NN Optimal` are needed.

In the experiments we present, each dataset is indexed by an M-tree and results are averaged over 100 queries. We concentrate on uniform datasets, since with *clustered* datasets both costs and effective errors are (much) lower, as expected. For simplicity, we approximate the query distance distribution, $F_q(\cdot)$, with the *overall* distance distribution, $F(\cdot)$, obtained by sampling the dataset at hand. From a practical point of view estimation errors are minimal, as demonstrated in [CPZ98a].[16] The sample size is between 1% (for larger datasets) and

[15] The table simply reports n if $E[M] \geq n$ results from the analysis.

[16] Alternatively, a better approximation of $F_q(\cdot)$ can be obtained by using the techniques described in [CNP99].

Algorithm `PAC-NN`

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

```
1.  Initialize the priority queue PQ with a pointer to the root node of $\mathcal{T}$;
2.  Compute $r^q_\delta$; Let $r = \infty$;
3.  While PQ $\neq \emptyset$ do:
4.      Extract the first entry from PQ, referencing node $N$;
5.      If $d_{min}(q, Reg(N)) > r/(1+\epsilon)$ 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.
```

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

10% of the dataset size, and $F(\cdot)$ is represented by a 100-bins equi-width histogram. We only present results where the "cost" is measured as the number of distance computations, since I/O costs (page reads) follow a similar trend.

PAC-NN Versus AC-NN Search. Figure 7 (a) contrasts PAC-NN and AC-NN search costs in high-D spaces. It is clear that AC-NN queries ($\delta = 0$) cannot be issued at such high dimensionalities, whereas the cost of PAC-NN queries remains quite low. Figure 7 (b) presents a more detailed analysis for the case $D = 40$, which confirms that ϵ alone is uneffective. On the other hand the cost becomes highly dependent on ϵ when $\delta > 0$ is used.

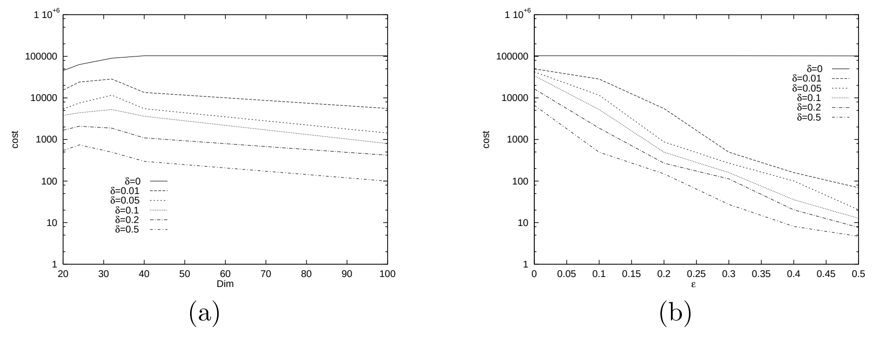
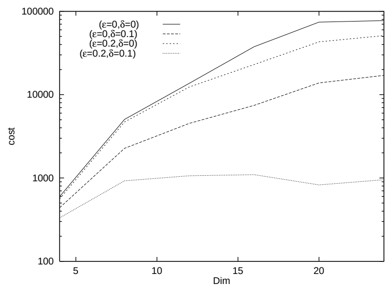
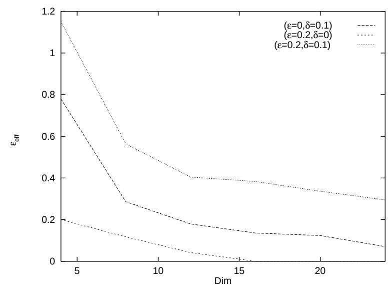


Fig. 7. Cost of AC-NN and PAC-NN queries in high-D spaces. $n = 10^5$. (a) As a function of space dimensionality when $\epsilon = 0.1$; (b) As a function of ϵ when $D = 40$.

In low-D spaces both PAC-NN and AC-NN algorithms can be profitably used. Figure 8 (a) shows that ϵ alone has a minimal influence on the cost,[17] and Figure 8 (b) confirms that PAC-NN search can exceed the error bound, the average amount depending on the choice of δ. The conclusion we can draw from our experience is that in low-D spaces the two algorithms can be made to run so as to obtain a similar tradeoff between cost and accuracy of the result.



(a)



(b)

Fig. 8. Low-D spaces. (a) Cost; (b) Effective error.

Tuning PAC-NN Search. The following graphs aim to provide some guidelines on how parameters of PAC-NN queries can be chosen in order to achieve a certain tradeoff between the actual quality of the result, i.e. ϵ_{eff}, and the cost. In Figure 9 we plot "iso-cost" lines, each line joining pairs of (ϵ, δ) values that lead to approximately the same cost, which provide a first intuition on how parameters have to be chosen in order to obtain a given performance level. Figure 10 (a) relates the effective error to the cost, with each curve referring to a different δ value. The most important observation is that ϵ_{eff} *is almost insensitive to the specific choice of* ϵ *and* δ *values*, provided the two parameters are chosen so as to yield the desired cost (i.e. they belong to the given "iso-cost" curve). For convenience, the values of δ which guarantee to have $\epsilon_{eff} \approx \epsilon$ are given in Figure 10 (b), for several values of the ϵ parameter.

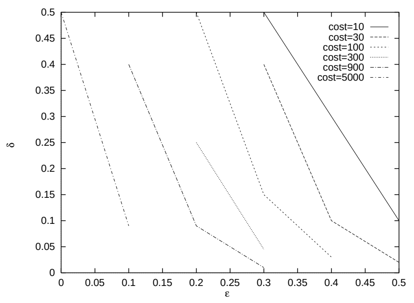
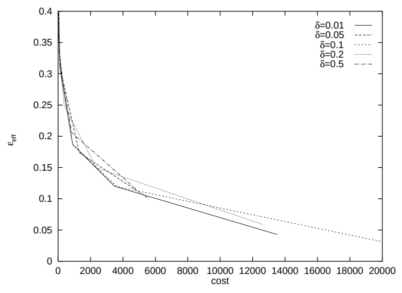
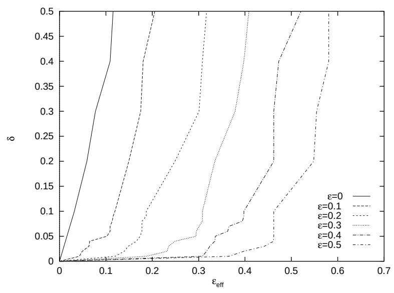


Fig. 9. "Iso-cost" curves. $D = 40$.

[17] This does not contradict results in [AMN$^+$], where much higher values of ϵ are considered, up to $\epsilon = 10$.



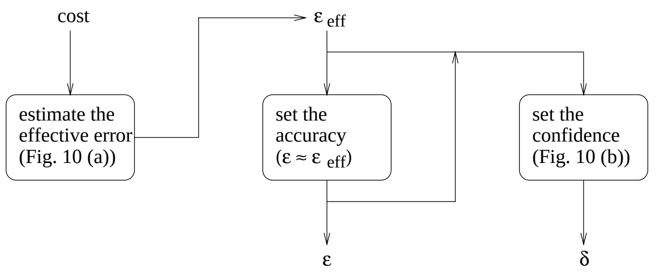
(a)



(b)

Fig. 10. (a) Effective error vs. cost; (b) δ vs. ϵ_{eff}. In both cases it is $D = 40$.

A realistic scenario for an user issuing PAC-NN queries on a dataset for which are available statistics of above kind is depicted in Figure 11. 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 10 (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 10 (a). As an example, in order to have $\epsilon_{eff} = 0.2$, Figure 10 (a) predicts a cost in the range 800..1400, and Figure 10 (b) suggests to use $\delta \approx 0.1$.



cost
ε eff
estimate the effective error (Fig. 10 (a))
set the accuracy (ε ≈ ε eff)
set the confidence (Fig. 10 (b))
ε
δ

Fig. 11. Flow diagram showimg how ϵ and δ values can be chosen to yield a given performance level (effective error or cost).

Searching an Image Database. We have experimented the PAC-NN algorithm on a real-world collection of 11,648 color images, represented as 45-dimensional vectors obtained using the method described in [SO95] and compared using the Euclidean distance. In general, for all the queries we tried, costs were reduced up to 50% by using the PAC-NN algorithm. As to the quality of the result, the general trend was that, even using quite high values of ϵ and δ, the correct NN was retrieved also by the PAC-NN algorithm. Figure 12 presents two sample cases, with the query image shown in the left column and the NN in the middle column. For the *fox* query the NN is also retrieved by the PAC-NN algorithm as long as $\epsilon < 1$ and $\delta < 0.5$. For higher values of the parameters the PAC-NN search retrieves the image shown on the right, which however is still semantically related to the query image. This is not the case for the *turtle* query, even if now the correct result is more "stable", staying unchanged up to $(\epsilon, \delta) = (1.5, 0.5)$.

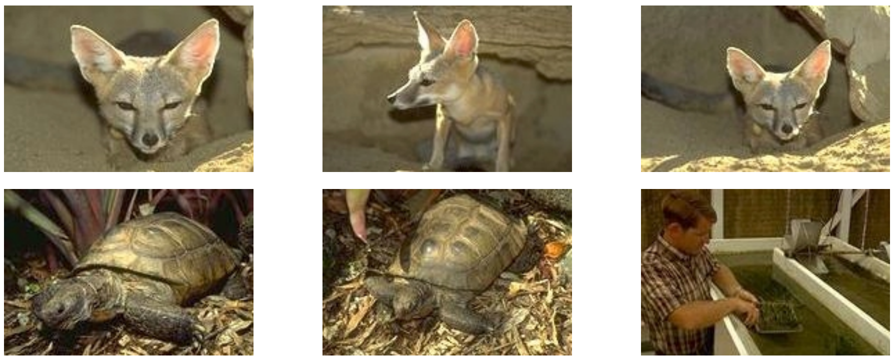


Fig. 12. (a) Query image; (b) C-NN and "good" PAC-NN; (c) "Bad" PAC-NN obtained with $(\epsilon, \delta) = (1, 0.5)$ (*fox* query) and $(1.5, 0.5)$ (*turtle* query).

Sequential vs Index-based PAC-NN Search. We conclude by exhibiting some results which contrast sequential and index-based PAC-NN algorithms. Since, as discussed at the beginning of this Section, the index uses the overall distance distribution (rather than the one specific of 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 2 presents results for a 40-dimensional dataset 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 of interest, as discussed in Sections 3.1 and 4.

$\epsilon \downarrow \quad \delta \rightarrow$	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 2. Costs of index-based and (sequential) PAC-NN algorithms. $n = 10^5$, $D = 40$.

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, where both ϵ and δ can be chosen on a per-query basis. We have 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 necessitate of some prior information on the *distance distribution* of the query point, which, using results in [CPZ98a], can be however reliably approximated by the *overall* distance distribution of the dataset. We have also shown that it is indeed possible to exert an effective control on the quality of the result, thus trading off between accuracy and cost. This is an important issue which has gained full relevance in recent years [SGMC98].

Other approaches, besides the one proposed in [AMN$^+$] and that we have somewhat taken as a starting point, exist to support approximate NN search. Indik and Motwani [IM98] consider a hash-based technique able to return a $(1 + \epsilon)$-approximate NN with *constant* probability. Although interesting, this technique is limited to vector spaces and L_p norms, its preprocessing costs are exponential in $1/\epsilon$, and 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 in [Cla97], which applies to exact NN search over generic metric spaces, but whose space requirements depend on the error probability.

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 [BGRS99], 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.

We plan to extend our approach to k-nearest neighbors queries and to develop a cost model for predicting the performance of PAC-NN queries. Another interesting research issue would be to extend 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 an object [Fag96, CPZ98b].

References

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. In *Proc. of 4th FODO*, pages 69–84, 1993. Springer-Verlag, LNCS, Vol. 730.

[AMN$^+$] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Jou. of the ACM* (to appear).

[BBKK97] S. Berchtold, C. Böhm, D.A. Keim, and H.-P. Kriegel. A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space. In *Proc. of the 16th PODS*, pages 78–86, 1997.

[BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is "Nearest Neighbor" Meaningful? In *Proc. of the 8th ICDT*, pages 217–235 1999.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. of the ACM SIGMOD*, pages 322–331, 1990.

[BÖ97] T. Bozkaya and M. Özsoyoglu. Distance-Based Indexing for High-Dimensional Metric Spaces. In *Proc. of the ACM SIGMOD*, pages 357–368, 1997.

[Cla97] K.L. Clarkson. Nearest Neighbor Queries in Metric Space. In *Proc. of the 29th STOC*, pages 609–617, 1997.

[CNP99] P. Ciaccia, A. Nanni, and M. Patella. A Query-sensitive Cost Model for Similarity Queries with M-tree. In *Proc. of the 10th ADC*, pages 65–76, 1999.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the 23rd VLDB*, pages 426–435, 1997.

[CPZ98a] P. Ciaccia, M. Patella, and P. Zezula. A Cost Model for Similarity Queries in Metric Spaces. In *Proc. of the 17th PODS*, pages 59–68, 1998.

[CPZ98b] P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-based Access Methods. In *Proc. of the 6th EDBT*, pages 9–23, 1998.

[Fag96] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proc. of the 15th PODS*, pages 216–226, 1996.

[FEF$^+$94] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and Effective Querying by Image Content. *Jou. of Int. Inf. Sys.*, 3(3/4):231–262, July 1994.

[Gut84] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. of the ACM SIGMOD*, pages 47–57, 1984.

[IM98] P. Indik and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. of the 30th STOC*, 1998.

[KS97] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of the ACM SIGMOD*, pages 369–380, 1997.

[Pes99] V. Pestov. On the Geometry of Similarity Search: Dimensionality Curse and Concentration of Measure. Report RP-99-01, School of Math. and Comp. Sci., Victoria University of Wellington, NZ, 1999. URL: `http://xxx.lanl.gov/abs/cs.IR/9901004`.

[SGMC98] N. Shivakumar, H. Garcia-Molina, and C.S. Chekuri. Filtering with Approximate Predicates. In *Proc. of the 24th VLDB*, pages 263–274, 1998.

[SO95] M. Stricker and M. Orengo. Similarity of Color Images. In *SPIE*, pages 381–392, 1995.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. of the 24th VLDB*, pages 357–367, 1998.