Using the Distance Distribution for Approximate Similarity Queries in High-Dimensional Metric Spaces

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

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

Abstract

We investigate the problem of approximate *similarity (nearest neighbor) search in high-dimensional metric spaces, and describe how the* distance distribution *of the query object can be exploited so as to provide probabilistic guarantees on the quality of the result. This leads to a new paradigm for similarity search, called PAC-NN (*probably approximately correct nearest neighbor*) queries, aiming to break the "dimensionality curse". 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. Analytical and experimental results obtained for sequential and index-based algorithms show that PAC-NN queries can be efficiently processed even on very high-dimensional spaces and that control can be exerted in order to tradeoff the accuracy of the result and the cost.*

1. Introduction

Answering similarity queries is a difficult problem in high-dimensional spaces [4, 15], and recent studies also show that this phenomenon, known as the "dimensionality curse", is not peculiar to vector spaces, but can also affect more generic metric spaces [12]. The dimensionality curse plagues modern database applications, such as multimedia, data mining, decision support, and medical applications, where similarity is usually evaluated by first extracting high-D *feature vectors* from the objects, and then measuring the *distance* between feature values, so that similarity search becomes a *nearest neighbor* (NN) query over the feature space. Dimensionality curse, which is strictly related to the distribution of distances between the indexed objects and the query object [4] – intuitively, if these distances are all similar, i.e. their *variance* is low, then searching is difficult – vanifies the usage of both *multi-dimensional* trees (such as the R*-tree [2] and the SR-tree [11]) and *metric* trees (e.g. the M-tree [6]), the latter only requiring that the distance is a *metric*, thus suitable even when no adequate vector representation for the features is possible.

To obviate this unpleasant situation, several *approximate* solutions have been proposed that allow errors in the result in order to reduce costs [1, 16]. In this paper we propose a *probabilistic* approach, in which a NN query can specify two additional parameters: the *accuracy* ϵ allows for a certain relative error, and the *confidence* δ guarantees, with probability at least $(1-\delta)$, that ϵ will not be exceeded. This generalizes both *correct* (C-NN) and *approximately correct* (AC-NN) NN queries, where the latter only consider ϵ. 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. We first evaluate the performance of a sequential PAC-NN algorithm which, although effective in many cases, has a complexity still linear in the dataset size. We then provide experimental evidence that the index-based PAC algorithm, which we have implemented in the M-tree, can lead to substantial performance improvement. 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.

1.1. Basic Background

In our generic scenario, 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 used to measure the distance between 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. Given a set $S \subset \mathcal{U}$ 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 $r^q \stackrel{\text{def}}{=} d(q, p(q)) \leq d(q,p), \forall p \in S$.

An *optimal* index-based *correct* nearest neighbor (C-NN) algorithm is described in [3]. The algorithm is termed optimal since it only accesses those nodes of the index whose region intersects the *NN ball* $\mathcal{B}_{r^q}(q)$. The algorithm can be used with *any* multi-dimensional and metric index tree which is based on a recursive and conservative decomposition of the space, as it is the case with the R*-tree, the M-tree, and many others. However, the algorithm is effective only when the dimensionality D of the feature space is low (i.e. ≤ 10), after which a sequential scan becomes competitive [15]. Intuitively, this is because in high-D spaces r^q is very likely to be "large", thus the probability that a data region intersects the NN ball $\mathcal{B}_{r^q}(q)$ approaches 1.

In order to reduce costs, *approximate* NN algorithms have been proposed. The *quality* of the result of such algorithms is typically evaluated by the *effective (relative) error*, ϵ_{eff}, defined as:

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

where $r \geq r^q$ is the distance between q and the approximate NN returned by the algorithm. *Approximately correct* NN (AC-NN) algorithms [1, 16] use an *accuracy* parameter (relative error) ϵ, to bound ϵ_{eff}, i.e. they return a point p' (called a $(1+\epsilon)$-approximate NN) for which:

$$d(q,p') \leq (1+\epsilon)r^q \tag{2}$$

surely holds. The optimal algorithm for C-NN queries can be easily adapted to support AC-NN queries, by pruning all those nodes N whose region, $Reg(N)$, does not intersect the ball $\mathcal{B}_{r/(1+\epsilon)}(q)$, with r being the distance from q of the "current" (approximate) NN.

Even if performance of AC-NN algorithms can in principle be tuned by varying ϵ, two problems arise. First, as results in [1] (for the BBD-tree) and [16] (for the M-tree) show, $\epsilon_{\mathit{eff}} \ll \epsilon$ usually holds, with ratios typically in the range $[0.01, 0.1]$. This implies that users cannot directly control the actual quality of the result, rather only a much-higher upper bound. Second, since the cost of AC-NN is still exponential in D, performance improvements are possible only in low-D [1] but not in high-D [16] spaces.

An alternative approach to AC-NN queries [16] considers to use the *relative distance distribution of* q, formally defined as:

$$F_q(x) = \Pr\{d(q,p) \leq x\} \tag{3}$$

where p is distributed according to a measure of probability over $\mathcal{U}$, denoted as μ. This leads us to consider *random metric spaces*, $\mathcal{M} = (\mathcal{U}, d, \mu)$ [7]. To help intuition, we slightly abuse terminology and also call μ the *data* distribution over $\mathcal{U}$. Since F_q depends on q, different query objects can have different "views" of the space [5].

Example 1 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 L_∞ "max" metric is used, $L_\infty(p_i,p_j) = \max_k\{| p_i[k] - p_j[k] \ |\} \leq 1$. When the query point coincides with the "center" of the space, $q^{cen} = (0.5, \ldots, 0.5)$, it is easy to derive that $F_{q^{cen}}(x) = (2x)^D$, whereas when the query point is one of the 2^D corners of the hypercube, it is $F_{q^{cor}}(x) = x^D$. □

The F_q-based algorithm in [16] stops the search when it finds a point p' whose distance $r = d(q,p')$ from q is such that

$$F_q(r) \leq \rho \tag{4}$$

where ρ is an input parameter. The idea is that when Eq. 4 holds for, say, $\rho = 0.01$, one obtains an approximate NN which is among the best 1% cases. Although interesting, this approach does not provide guarantees on the quality of the result, since ϵ_{eff} is not bounded by any function of ρ. Furthermore, it is not clear how ρ has to be chosen and how it affects the tradeoff between cost and accuracy.

2. Probably Approximately Correct Similarity Queries

The new approach we propose considers a *probabilistic* framework, and can be regarded as an extension of AC-NN queries where the error bound ϵ can be exceeded with a certain probability δ.

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 $\mathcal{B}_{(1+\epsilon)r^q}(q)$ ball is at least $1-\delta$, that is,*

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

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

The characterization of PAC-NN algorithms relies on information on the distance distribution. However, unlike [16], we make use of the *distribution of the distance of the nearest neighbor of* q with respect to a dataset of size n, which is given by [7]:

$$G_q(x) \stackrel{\text{def}}{=} \Pr\{r^q \leq x\} = 1 - (1 - F_q(x))^n \tag{5}$$

For instance, by referring to Example 1, it is $G_{q^{cen}}(x) = 1 - (1-(2x)^D)^n$ and $G_{q^{cor}}(x) = 1 - (1-x^D)^n$.

Given G_q, the basic idea of PAC-NN search is to avoid to search in a region which is "too close" to the query point, since, in high-D spaces, it is unlikely that any data point will be found therein because r^q is usually "large".

Definition 2 *Given a dataset S of n points, a query point q with nearest neighbor distance distribution G_q, 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, $r^q_\delta \stackrel{\text{def}}{=} \sup\{r \mid G_q(r) \leq \delta\}$. If G_q is invertible, then:*

$$r^q_\delta = G_q^{-1}(\delta) \tag{6}$$

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

$$r^{q^{cen}}_\delta = G^{-1}_{q^{cen}}(\delta) = \frac{1}{2}\left(1 - (1-\delta)^{1/n}\right)^{1/D} \tag{7}$$

When $D = 50$, $n = 10^6$, and $\delta = 0.01$, then $r^{q^{cen}}_{0.01} \approx 0.346$ results, that is, with probability 0.99 the hypercube centered on q^{cen} with side 2×0.346 is empty.

The definition of δ-radius immediately leads to a *stopping condition* with probabilistic guarantees. Let p' be the closest point to q discovered so far by a PAC-NN algorithm, and let $r = d(q, p')$. If

$$r \leq (1+\epsilon) r^q_\delta \stackrel{\text{def}}{=} r^q_{\delta,\epsilon} \tag{8}$$

then p' is a $(1+\epsilon; \delta)$-approximate nearest neighbor of q. This holds since $\Pr\{\epsilon_{\mathit{eff}} > \epsilon\} \leq \delta$ iff $\Pr\{r/r^q - 1 > \epsilon\} = \Pr\{r^q < r/(1+\epsilon)\} \leq \delta$. Since the last probability equals $G_q(r/(1+\epsilon))$ and $r/(1+\epsilon) \leq r^q_\delta = G_q^{-1}(\delta)$, it follows that $G_q(r/(1+\epsilon)) \leq G_q(G_q^{-1}(\delta)) = \delta$.

Figure 1 provides a graphical intuition on how parameters of a PAC-NN algorithm are used. Given a value of δ, the algorithm first determines the δ-radius r^q_δ, then stops the search according to the condition in Equation 8, i.e. as soon as a point p' is found whose distance from q does not exceed $(1+\epsilon) r^q_\delta = r^q_{\delta,\epsilon}$, which is conveniently called the (δ, ϵ)*-radius* of q.

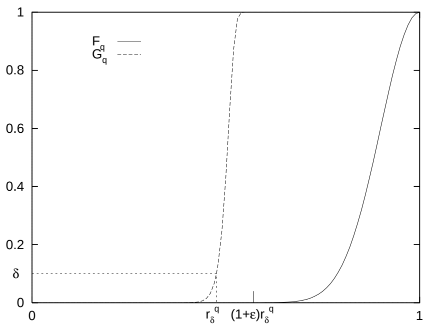


Figure 1. How G_q, ϵ, and δ interact in PAC-NN search.

2.1. Analysis of the PAC-NN Sequential Search

When the dataset S is stored as a sequential file, an AC-NN search ($\delta = 0$) would necessarily scan the whole file. In order to estimate the cost, measured as the number of distance computations, of a PAC-NN query, we can consider a *random sampling process with repetitions* (i.e. 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.

With the above assumptions, the cost M is a *geometric* random variable, where the probability of success of a "trial" is $F_q(r^q_{\delta,\epsilon})$, thus its expected value is simply the inverse of the trial success probability, $E[M] = 1/F_q(r^q_{\delta,\epsilon})$. It follows that *the cost does not change as long as $r^q_{\delta,\epsilon}$ is kept constant*. As an example, given the value of $r^{q^{cen}}_\delta$ in Eq. 7, it is obtained:

$$E[M] = \frac{1}{(1+\epsilon)^D (1 - (1-\delta)^{1/n})} \tag{9}$$

Thus, as long as $\epsilon = \text{const.}/(1 - (1-\delta)^{1/n})^{1/D} - 1$ the cost will not change. Results in Table 1 are in line with the analysis (this, as expected, breaks down when $E[M] \ll n$ does not hold).[1]

As to the effective error, its distribution is derived to be:

$$\Pr\{\epsilon_{\mathit{eff}} \leq x\} = 1 - G_q(r^q_{\delta,\epsilon}/(1+x)) + \\ + \int_0^{r^q_{\delta,\epsilon}/(1+x)} \frac{F_q((1+x)y) - F_q(y)}{F_q(r^q_{\delta,\epsilon}) - F_q(y)}\, g_q(y)\, \mathrm{d}y \tag{10}$$

where g_q is the density of G_q and $1 - G_q(r^q_{\delta,\epsilon}) = \Pr\{\epsilon_{\mathit{eff}} = 0\}$. The denominator "normalizes" the possible distances to those that can result when $r^q = y \leq r^q_{\delta,\epsilon}$, that is $[y, r^q_{\delta,\epsilon}]$. This, together with $E[M] = 1/F_q(r^q_{\delta,\epsilon})$, completely characterizes the tradeoff between accuracy and cost. Table 2 shows some statistics on the effective error distribution.

δ	ϵ_{eff} (avg)	ϵ_{eff} (max)	$\epsilon_{\mathit{eff}} > \epsilon$ (% 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 2. Statistics on the effective error. $\epsilon = 0.2, n = 10^5, D = 40$.

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

$\epsilon \downarrow \quad \delta \rightarrow$	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)

Table 1. Expected and (actual) costs of the PAC-NN sequential algorithm. $n = 10^6, D = 100$.

□

3. The Index-based PAC-NN Algorithm

From Eq. 9 it can be derived that the PAC-NN sequential algorithm has complexity at least $O(n\delta^{-1}(1+\epsilon)^{-D})$, thus linear in n and unsuitable for (very) large datasets, especially when ϵ and δ have both small values. The index-based PAC-NN algorithm in Figure 2 follows the outline of the optimal algorithm in [3], where a priority queue containing references to the tree nodes is used. The queue PQ is ordered on increasing values of $d_{min}(q, Reg(N))$, i.e. the minimum distance between q and the region of node N. The stopping condition (Eq. 8) is at line 8. The ϵ parameter is also used in lines 5 and 11 to prune tree nodes, as it happens in AC-NN search.

Algorithm PAC-NN

Input: index tree $\mathcal{T}$, query object q, ϵ, δ, G_q;
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 T;
2.  Let r_δ^q = G_q^{-1}(δ); Let r = ∞;
3.  While PQ ≠ ∅ do:
4.      Extract the first entry from PQ, referencing node N;
5.      If d_min(q, Reg(N)) ≥ r/(1+ε) 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 ≤ (1+ε) r_δ^q 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+ε):
12.                 Update PQ performing an ordered insertion
                    of the pointer to N_c;
13. End.
```

Figure 2. The index-based PAC-NN algorithm.

Example 2 Refer to Figure 3, where the metric space is $(\Re^2, L_2)$, points are indexed by an M-tree, whose regions are *balls*, $Reg(N) = \mathcal{B}_{r_N}(p_N)$, and $d_{min}(q, Reg(N)) = \max\{d(q, p_N) - r_N, 0\}$. In Figure 3 (a) p' is the current NN, $r = d(q, p')$, and the queue contains pointers to nodes A, B, C, and D. Node A is first accessed, since $d_{min}(q, Reg(A)) = \max\{d(q, p_A) - r_A, 0\} = 0 < r/(1+\epsilon)$, object p'' becomes the new current NN, and $r = d(q, p'')$ is set. Then (Figure 3 (b)) the search is immediately stopped, since $r \leq r^q_{\delta,\epsilon}$, and point p'' is returned.

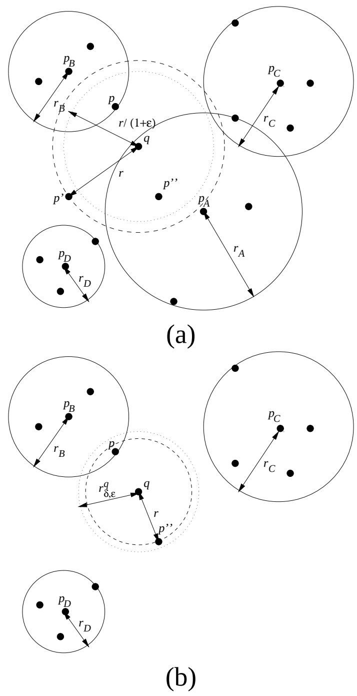
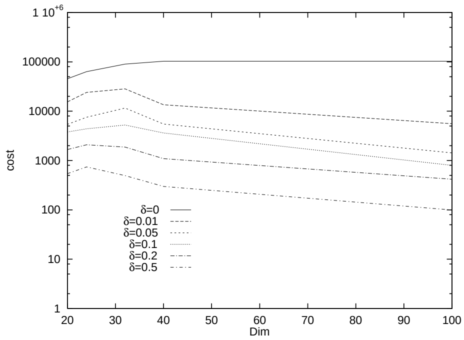


Figure 3. PAC-NN search in the metric space ($\Re^2$,L_2).

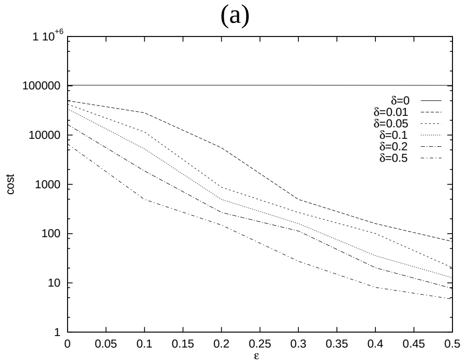
In the experiments we present, each dataset is indexed by an M-tree and results are averaged over 100 queries. We concentrate on uniform datasets (with *clustered* datasets both costs and effective errors are (much) lower, as expected). For simplicity, we approximate the query distance distribution, F_q, with the *overall* distance distribution, F, obtained by sampling the dataset at hand. From a practical point of view estimation errors are minimal, as demonstrated in [7].[2] 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, up to a scale factor which depends on the average number of entries in each node.

[2]Alternatively, a better approximation of F_q can be obtained by using the techniques described in [5].

PAC-NN vs AC-NN Search. Figure 4 (a) contrasts PAC-NN and AC-NN search costs in high-D spaces. In such spaces, where the *intrinsic* dimensionality of the dataset is high, AC-NN algorithms ($\delta = 0$) are not able to speed up the search with respect to a sequential scan and even dimensionality reduction techniques [13] fail, whereas the cost of PAC-NN queries remains quite low. Figure 4 (b) presents a more detailed analysis for the case $D = 40$.



(a)



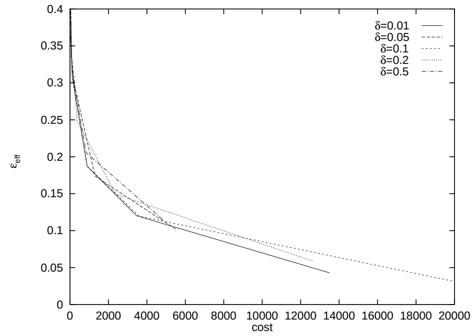
(b)

Figure 4. 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 $D = 40$.

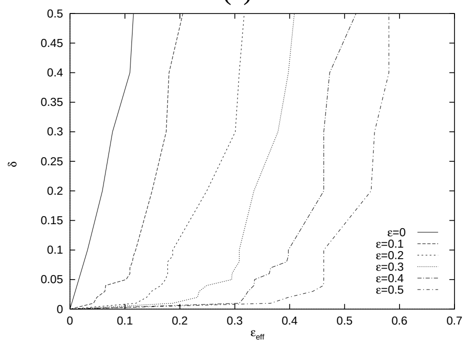
Tuning PAC-NN Search. Figure 5 (a) relates ϵ_{eff} to the cost, and confirms that ϵ_{eff} is almost independent of the specific ϵ and δ values, provided they are chosen to yield a given cost level. This is consistent with our statement of Section 2.1, i.e. both ϵ_{eff} and the search cost only depend on $r^q_{\delta,\epsilon}$.

A realistic scenario for an user issuing PAC-NN queries on a dataset for which are available such kind of statistics is as follows. 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 5 (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 5 (a).

Sequential vs Index-based PAC-NN Search. In Table 3 we present some results which contrast sequential and index-based PAC-NN algorithms on a 40-dimensional dataset with 10^5 uniformly distributed points. The improvement obtainable through indexing is always consistent (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).



(a)



(b)

Figure 5. (a) Effective error vs. cost; (b) δ vs. ϵ_{eff}. In both cases it is $D = 40$.

4. 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 [7], 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 [14].

Other approaches, besides those in [1, 16], exist to support approximate NN search. Indik and Motwani [10]

$\epsilon\downarrow \quad \delta\rightarrow$	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 3. Costs of index-based and (sequential) PAC-NN algorithms. $n = 10^5$**,** $D = 40$**.**

consider a hash-based technique able to return a $(1+\epsilon)$-approximate NN with *constant* probability. However, 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 [9], which applies to exact NN search over generic metric spaces, but whose space requirements depend on the error probability.

In the future, we plan to extend our approach to k-nearest neighbors queries and to develop a cost model for predicting the performance of the index-based PAC-NN search. 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 [8].

References

[1] 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. To appear on *Journal of the ACM*. A preliminary version has appeared in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[3] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 78–86, Tucson, AZ, May 1997.

[4] K. Beyer, J. Goldstein, R. Ramakhrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, pages 217–235, Jerusalem, Israel, Jan. 1999.

[5] P. Ciaccia, A. Nanni, and M. Patella. A query-sensitive cost model for similarity queries with M-tree. In *Proceedings of the 10th Australasian Database Conference (ADC'99)*, pages 65–76, Auckland, New Zealand, Jan. 1999.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, Aug. 1997.

[7] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 59–68, Seattle, WA, June 1998.

[8] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pages 9–23, Valencia, Spain, Mar. 1998.

[9] K. L. Clarkson. Nearest neighbor queries in metric spaces. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97)*, pages 609–617, El Paso, TX, May 1997.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 604–613, Dallas, TX, May 1998.

[11] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 369–380, New York, NY, May 1997.

[12] 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, Jan. 1999. `http://xxx.lanl.gov/abs/cs.IR/9901004`.

[13] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 154–165, Seattle, WA, June 1998.

[14] N. Shivakumar, H. Garcia-Molina, and C. Chekuri. Filtering with approximate predicates. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB'98)*, pages 263–274, New York, NY, Aug. 1998.

[15] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 194–205, New York, NY, Aug. 1998.

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