Speeding up Spatial Approximation Search in Metric Spaces

KARINA FIGUEROA
Universidad Michoacana, Mexico
EDGAR CHAVEZ
Universidad Michoacana / CICESE, Mexico
GONZALO NAVARRO
RODRIGO PAREDES
University of Chile, Chile

Proximity searching consists in retrieving from a database those elements that are similar to a query object. The usual model for proximity searching is a metric space where the distance, which models the proximity, is expensive to compute. An index uses precomputed distances to speed up query processing. Among all the known indices, the baseline for performance for about twenty years has been AESA. This index uses an iterative procedure, where at each iteration it first chooses the next *promising* element ("pivot") to compare to the query, and then it discards database elements that can be proved not relevant to the query using the pivot. The next pivot in AESA is chosen as the one minimizing the sum of lower bounds to the distance to the query proved by previous pivots. In this paper we introduce the new index *iAESA*, which establishes a new performance baseline for metric space searching. The difference with AESA is the method to select the next pivot. In iAESA, each candidate sorts previous pivots by closeness to it, and chooses the next pivot as the candidate whose order is most similar to that of the query. We also propose a modification to AESA-like algorithms to turn them into probabilistic algorithms.

Our empirical results confirm a consistent improvement in query performance. For example, we perform as few as 60% of the distance evaluations of AESA in a database of documents, a very important and difficult real-life instance of the problem. For the probabilistic algorithm, we perform in a database of faces up to 40% of the comparisons made by the best alternative algorithm to retrieve the same percentage of the correct answer. Based on the empirical results we conjecture that the new probabilistic AESA-like algorithms will become, as AESA had been for exact algorithms, a reference point establishing in practice a lower bound on how good a probabilistic proximity search algorithm can be.

Categories and Subject Descriptors: E.1 [**Data Structures**]: ; H.3.3. [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Algorithms, Performance

Supported by CONACyT (Mexico) and Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, (Chile). The fourth author also acknowledges the Database Laboratory, University of A Coruña, Spain, Grant TIN2006-15071-C03-03.
Author's address: Karina Figueroa, Edgar Chávez, Ciudad Universitaria, Edificio B, Universidad Michoacana, Morelia, Michoacan, Mexico. Email {karina,elchavez}@fismat.umich.mx. Gonzalo Navarro, Rodrigo Paredes, Dept. of Computer Science, University of Chile, Blanco Encalada 2120, Santiago, Chile. Email: {gnavarro,raparede}@dcc.uchile.cl.
A preliminary partial version of this paper appeared in *Proc. 5th Workshop on Experimental Algorithms (WEA)*, LNCS v. 4007, pp. 279–290, 2006.

Additional Key Words and Phrases: Proximity or similarity searching, AESA

1. INTRODUCTION AND RELATED WORK

Proximity or similarity searching is nowadays an essential tool in a number of practical tasks such as vector quantization of signals, pattern recognition, retrieval of multimedia information, and so on. In these applications there is a database (e.g., a set of documents) and a similarity measure among its objects (e.g., the cosine distance). The similarity is modeled by a distance function defined by experts in each application domain, which tells how similar two objects are. The objects are seen as black boxes, so that the only operation permitted is to measure their distance towards another object. The distance function is usually quite expensive to compute and the CPU cost of side computations are not considered, and even in some cases the I/O cost is also neglected. The search complexity is taken as just the number of distance evaluations needed to carry out similarity searches, and thus the goal is to search by performing the minimum number of distance evaluations.

The most basic and common case of similarity search is when the query is an object, which is not necessarily in the database, and we ask to retrieve the k objects most similar to it in the database, or all database objects within certain distance to the query object. A brute force approach scans all the database to answer either of the above queries. To reduce the query cost, an index is built on the database before searching it. The index is a data structure that stores information on some distances among database elements. This information is used later to discard some elements without comparing them directly with the query object.

Different indices store different information about distances [Chávez et al. 2001]. Some store a subset of the distances, e.g. all the distances between k chosen pivots and all the rest, or all the distances between an element and its subtree, in a tree-structured index. Some indices store just a range of distance values, and in general, the more information an index stores, the lower query cost it achieves (although some use memory better than others). In this view, in a database of n objects the most information an index could store is the $n(n-1)/2$ distances among all element pairs. This is usually avoided because $O(n^2)$ space is unacceptable for realistic database applications. However, the space is affordable in some areas such as pattern recognition, as well as to index database subsets [Fredriksson 2007]. What is especially relevant of this approach is that the use of all the available information establishes a *baseline* on how good could an index possibly be. Actually, all the development on metric space indexing can be regarded as the quest for maintaining good efficiency while reducing the amount of storage used [Chávez et al. 2001].

We do not attempt to cover all the existing algorithms for metric space searching. The reader is referred to exhaustive surveys [Chávez et al. 2001; Hjaltason and Samet 2003] or books [Zezula et al. 2006; Samet 2006]. We will focus on the canonical algorithm that uses all the possible distances, AESA [Vidal 1986]. For 20 years AESA has been the indexing technique requiring, by far, the least number of distance computations among all other indices (which require much less space).

In this paper we show, for the first time, that it is possible to establish a new baseline on the number of distance evaluations for proximity searching. More specif-

ically, AESA uses a twofold procedure by firstly choosing a "pivot" from the remaining set of candidates and using it to prune other candidates. The closer the pivot to the query q, the more effective the pruning is. Each pivot enables a lower bound for the distance from any database object u to the query, as the difference of distances from u and q to the pivot p. AESA selects the next pivot as the one minimizing the sum of lower bounds to the distance towards the query proved by previous pivots. Then it prunes other candidates using the maximum of those lower bounds. In this paper we introduce a new technique to choose the next pivot that guesses better a close candidate, by choosing the candidate that orders previous pivots by closeness in the way most similar to how the query orders them. Our technique reduces the number of distance evaluations by up to 40% in document databases, which is a very difficult real-life instance of the proximity search problem.

A serious problem of all algorithms in metric spaces, even for AESA, is that when the intrinsic dimension of the space[1] grows, the whole database needs to be reviewed [Chávez et al. 2001]. In this case a probabilistic algorithm (which can miss some relevant answers) is a practical tool. Any exact algorithm can be turned into probabilistic, by letting it work until some predefined work threshold and measuring how many relevant answers it found.

Probabilistic algorithms have been proposed both for vector spaces [Arya et al. 1994; White and Jain 1996] and for general metric spaces [Clarkson 1999; Ciaccia and Patella 2002; Chávez and Navarro 2003; Bustos and Navarro 2003]. Bustos and Navarro [2003] define a probabilistic algorithm using a technique relevant to this work. They use different criteria to sort the database according to some *promise value*. As they traverse the database in such order they find relevant answers to the query. A good database ordering obtains most of the relevant answers by traversing a small fraction of the database. Given a limited amount of work allowed, the algorithm finds each correct answer with some probability, and it can refine the answer incrementally if more work is allowed. Thus, the problem of finding good probabilistic search algorithms translates into finding good database orderings.

We show how AESA and iAESA can be turned into probabilistic, where the traversal order is precisely given by the heuristic to choose the next pivot. Probabilistic iAESA performs better than probabilistic AESA, becoming also much stronger than existing probabilistic algorithms (which use less space, however). For example, in a database of faces, our algorithm performs less than 40% of the distance evaluations carried out by the best alternative probabilistic algorithm to achieve the same percentage of correct answers.

2. METRIC SPACES AND AESA

2.1 Notation and Basic Concepts

Let $(\mathbb{X}, d)$ be a metric space, where $\mathbb{X}$ is the universe of objects and d the distance function among the objects in $\mathbb{X}$. The distance function $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}^+$ is defined by experts in the application domain and expresses the dissimilarity between

[1]In a strict sense, the *intrinsic dimension* is the minimum dimensionality of a vector space to which the database can be mapped without distortion, preserving the distance matrix. The term is also used in a broader sense, as a measure of the search difficulty of a metric space, and there are several estimators of it based on the histogram of distances, see e.g., Chávez et al. [2001].

objects in $\mathbb{X}$. The distance function must satisfy the following properties: strict positiveness ($d(x,y) > 0 \Leftrightarrow x \neq y$), symmetry ($d(x,y) = d(y,x)$) and triangle inequality ($d(x,z) \leq d(x,y) + d(y,z)$).

Let $\mathbb{U} \subseteq \mathbb{X}$ be our database of size n, $q \in \mathbb{X}$ the query, and $r \geq 0$. The similarity queries can be classified into two basic types:

—Range query, $(q,r) = \{u \in \mathbb{U},\ d(u,q) \leq r\}$

—k-nearest neighbor query, k-NN$(q) = A$ such that $\forall\ u \in A, v \in \mathbb{U} - A$, $d(u,q) \leq d(v,q)$, and $|A| = k$.

The naive approach to these kind of queries is to compare the whole database with the query. This solution, however, requires n distance computations. An index is a data structure on $\mathbb{U}$ that solves queries of either type trying to use less than n distance evaluations. As the objects are black boxes, the search always proceeds by comparing q with some element of $\mathbb{U}$, discarding candidates using that distance and the help of the index, and so on until every element is either discarded or reported.

The performance of the algorithms in metric spaces is affected by the intrinsic dimension of the data, as described by Chávez et al. [2001]. When the dimension grows, the mean of a distance between random objects increases and the variance diminishes. This implies very low selectivity and prevents an effective use of the triangle inequality: In high dimensions no algorithm can avoid a sequential scan. AESA is also affected by dimension in spite of being the best proximity search algorithm in metric spaces.

2.2 AESA: Approximating and Eliminating Search Algorithm

AESA was introduced by Vidal [1986]. AESA needs to compute and store a matrix as an index, recording every distance $d(u,v)$, $\forall\ u, v \in \mathbb{U}$, that is $O(n^2)$ distances. During the search process, an element from the remaining candidates, called a "pivot", is chosen and compared with the query. AESA uses the matrix of distances to discard remaining candidates using the triangle inequality. The algorithm is described in Section 2.3.

Although $O(n^2)$ space can be a large amount of memory, there are applications with small enough databases (up to few thousand objects) where managing all the $O(n^2)$ distances is possible. For this kind of applications, AESA is still a practical solution and the one performing least distance computations.

In the case of larger databases, where $O(n^2)$ distances cannot be stored, it is still possible to partition the database using a hierarchical index and then apply AESA on the last level of the hierarchy [Fredriksson 2007].

AESA has been for 20 years the algorithm that computes the least number of distance evaluations to answer proximity queries. There have been some algorithms aimed at reducing its preprocessing time or space used. LAESA [Micó et al. 1994] chooses k elements of $\mathbb{U}$ as potential pivots, then reducing the space to $O(kn)$. An improved version of LAESA is Tree LAESA [Micó et al. 1996] which achieves sublinear side computations at query time with just approximately twice the average number of distance evaluations. Reduced Overhead AESA [Vilar 1995] strictly calculates the same distances as AESA but reduces the query processing time. Recently, graph t-spanner indices [Navarro et al. 2007] were used to simulate AESA, almost reaching its number of distance calculations using less memory. In fact, all

the development on indices for metric spaces can be seen as attempts to simulate the performance of AESA using less memory [Chávez et al. 2001].

2.3 Searching Using AESA

Like most indexing algorithms, AESA solves nearest neighbor queries by choosing a subset $\mathbb{P} \subseteq \mathbb{U}$ (called the *pivots*) to compare with q, and then filtering out as many elements of $\mathbb{U}$ as possible using precomputed distances. By the triangle inequality, if p is a pivot for which we have computed $d(p, q)$, and the index stores distance $d(p, u)$, then we can deduce that $d(q, u) > r$ without evaluating $d(q, u)$ if $|d(u, p) - d(p, q)| > r$. As described by Chávez et al. [2001], this can be regarded as using a contractive mapping Φ from the original metric space into a vector space. In this mapping each element of the original metric space is represented as a point in the target vector space. Let $\mathbb{P} = \{p_1 \dots p_k\}$, then $u \in \mathbb{U}$ is mapped to $\Phi(u) = (d(u, p_1), \dots, d(u, p_k))$. If we set the target space to be $(\mathbb{R}^k, L_\infty)$, where L_∞ is the well-known Minkowski metric on k dimensions, then the mapping is contractive, and thus one can safely discard projected elements using the same radius r. By analogy we use $D_\infty(q, u) = L_\infty(\Phi(q), \Phi(u))$ in Eq. (1), and similarly D_1 in Eq. (2).

Considering all the pivots, we can discard u from the query outcome without evaluating $d(q, u)$ if $D_\infty(q, u) > r$, where

$$D_\infty(q, u) \;=\; \max_{p \in |\mathbb{P}|} |d(u, p) - d(p, q)| \tag{1}$$

is the best lower bound one can prove on $d(q, u)$ using pivots $\mathbb{P}$. We remark that $D_\infty(q, u)$ is computed without any distance evaluation once the pivots have been compared to q and the distances of elements u to pivots p are stored in the index.

The key aspect of AESA is that the subset $\mathbb{P}$ to compare with the query is not fixed, but determined online one at a time. That is, every element (not yet compared nor discarded) is a candidate to pivot, and thus all the distances between elements of $\mathbb{U}$ are precomputed and stored in the index. Because pivots closer to the query are better at filtering out other elements, AESA attempts to choose the next pivot as a candidate u that appears to be close to q. Of course this must be done without measuring the actual distance $d(q, u)$, as otherwise we have already paid the cost to convert the candidate into pivot.

AESA uses the information given by the previous pivots (the set $\mathbb{P}$, which now grows as each new pivot is chosen and compared with the query) to estimate how promising are the remaining candidates. More precisely, the next pivot to compare with the query q is chosen as the candidate u minimizing

$$D_1(q, u) \;=\; \sum_{p \in |\mathbb{P}|} |d(u, p) - d(p, q)|. \tag{2}$$

The heuristic states that the *sum* of lower bounds is a good guess of how far is the candidate u from p. This essentially completes the description of the range search algorithm using AESA. The algorithm to answer a nearest neighbor query, 1-NN(q), is slightly more complicated and can be summarized in five steps.

(1) **Initialization**. The sets of pivots $\mathbb{P}$ and filtered elements $\mathbb{F}$ are empty. Let $D_1(u) \leftarrow 0$ for all $u \in \mathbb{U}$ and $r \leftarrow \infty$. Steps 2-5 are repeated until $\mathbb{U} = \mathbb{P} \cup \mathbb{F}$.

(2) **Approximating**. A new pivot p is selected as the one minimizing $D_1(u)$ (Eq. (2)) among the remaining candidates, that is, $p \leftarrow \text{argmin}_{u \in \mathbb{U}-\mathbb{F}-\mathbb{P}} D_1(u)$.

(3) **Updating the NN**. Distance $d(p, q)$ is computed. If $d(q, p) < r$, then the current nearest neighbor and the distance r are updated.

(4) **Updating the Estimations**. The new p is added to the set of pivots $\mathbb{P}$. Every object in $\mathbb{U} - \mathbb{F} - \mathbb{P}$ updates its approximation criterion according to Eq. (2), that is $D_1(u) \leftarrow D_1(u) + |d(u, p) - d(p, q)|$.

(5) **Eliminating**. Those $u \in \mathbb{U} - \mathbb{F} - \mathbb{P}$ such that $D_\infty(q, u) > r$ are discarded using the triangle inequality, by adding them to $\mathbb{F}$. The process continues at Step 2.

A range query process (q, r) can be implemented similarly by keeping r fixed and reporting every p such that $d(p, q) \leq r$. In the case of k-NN(q) queries for $k > 1$ it is necessary to keep the k nearest neighbor candidates in a heap data structure. Whenever a distance is computed the object and the corresponding distance will be inserted in the heap. The first k elements correspond to the current k nearest neighbors, and r will be the distance from q to the current k-th neighbor.

Step (2) of the algorithm is the heuristic part, as it determines the next pivot to use. A small improvement in the approximating step may yield a large improvement in performance. Our aim is to propose a better technique to select the next pivot.

3. OUR PROPOSAL

3.1 A New Database Ordering

Our goal is to define an order such that the first element is very close to the query, without actually computing real distances. This order depends on the query and must be computed just in time, when the query arrives. A natural way to define such an ordering is to use another distance δ (easier to compute than d itself), and choose the next pivot as the element u minimizing $\delta(q, u)$.

Note, incidentally, that probabilistic algorithms as defined at the end of the Introduction build on the same idea: sort and traverse the database according to an order where one expects that the first elements reviewed will be close to the query.

In the metric proximity search literature, especially that related to AESA and probabilistic algorithms, there are two main examples of distances used to sort the database: D_∞ (Eq. (1)) and D_1 (Eq. (2)). The former has the additional benefit of being a lower bound to the distances to q, so it permits terminating the search when the most promising element is sufficiently far away from q in terms of D_∞. In turn, the second has been preferred in AESA for its good heuristic performance.

Our main contribution is the definition of another way to sort the candidates, which obtains better results in terms of choosing better pivots for AESA. Unlike the two previous examples, where the numeric values of previously computed distances are used, we use the relative order in which the pivots are perceived by the candidates. The precise definition needs some formalism.

Every element $x \in \mathbb{X}$ defines a *preorder* $\leq_x$ in $\mathbb{P}$ given by the distance to x. The preorder is defined for $y, z \in \mathbb{P}$, as

$$y \leq_x z \iff d(x, y) \leq d(x, z).$$

The relation $\leq_x$ is a preorder and not an order because some elements can be at the same distance from x, and then $\exists\ y \neq z$ such that $y \leq_x z \ \wedge \ z \leq_x y$. Since this distinction is not essential for our construction we will convert $\leq_x$ into an order by breaking ties arbitrarily (but consistently for all x). Let us fix $\mathbb{P} = \{p_1,\ p_2, \ldots,\ p_{|\mathbb{P}|}\}$, then we redefine $\leq_x$ as

$$p_i \leq_x p_j \iff d(x,p_i) < d(x,p_j) \ \vee \ (d(x,p_i) = d(x,p_j) \ \wedge \ i < j).$$

Now it turns out that $\leq_x$ is a *total order* (that is, every pair of pivots is comparable), thus we can associate a permutation Π_x to each $x \in \mathbb{X}$. We define permutation $\Pi_x = (i_1, i_2, \ldots, i_{|\mathbb{P}|})$ of $(1 \ldots |\mathbb{P}|)$ so that

$$p_{i_1} \leq_x p_{i_2} \leq_x \cdots \leq_x p_{i_{|\mathbb{P}|}}.$$

We also use $\Pi_x^{-1}(i)$ to identify the position of element i in permutation Π_x.

Our intuition is that two close elements will have a similar permutation (indeed, if the two elements coincide, they must have the same permutation). Therefore, we propose to sort, up to ties, the candidates u by dissimilarity between Π_u and Π_q. Tie breaking in permutations will be discussed shortly.

There are several choices in the literature for measuring dissimilarities between permutations. In previous work [Chávez et al. 2008] we experimented with a few of them, such as Kendall Tau, Spearman Rho, or Spearman Footrule [Fagin et al. 2003]. We found that their performance, for our purposes, were similar. Since Spearman Footrule is the least expensive to compute among those, we choose it as our dissimilarity measure. Spearman Footrule measures how much the indexes are displaced in the respective permutations and is defined as follows:

$$F(\Pi_u, \Pi_q) = \sum_{i=1}^{|\mathbb{P}|} |\Pi_u^{-1}(i) - \Pi_q^{-1}(i)|. \tag{3}$$

For example, let $\Pi_q = (42153)$ be the permutation associated to the query, and $\Pi_u = (32154)$ the permutation associated to an element $u \in \mathbb{U}$. According to Eq. (3), we have $F(\Pi_u, \Pi_q) = |3-3| + |2-2| + |5-1| + |1-5| + |4-4| = 8$. In this case objects 3 and 4 exchanged positions, counting four positions each.

3.2 iAESA: Improved AESA

After the previous discussion we can establish the core of our proposal. It is simply to substitute the use of D_1 by F (Eq. (3)) in the approximating step of AESA (Section 2.3). The rest of the algorithm remains the same.

Note that, since $\mathbb{P}$ grows as the algorithm progresses, the permutations and the F measures for the remaining candidates u change on the fly. Figure 1 illustrates the changes made on AESA to obtain iAESA.

In Figure 2 we compare iAESA with AESA, using the same example shown by Vidal [1986]. The example retrieves the nearest neighbor to the query q. In the figure (left side) the objects are $p_1, \ldots, p_7$ and q is the query; the solid lines are the terms of Equation (2); the dashed lines indicate the process to select the next pivot (labeled step-1, step-2 and step-3); the circle and the semicircles are the distances from a pivot to the query, and the semicircles are labeled by the order of execution, step-1, step-2 and step-3. They help viewing the ordering induced by D_1.

AESA

```
1.  Let ℙ ← ∅ set of pivots
2.  Let 𝔽 ← ∅ set of filtered elements
3.  r ← ∞
4.  for u ∈ 𝕌, D(u) ← 0, D_u^max ← 0
5.  while 𝕌 ≠ ℙ ∪ 𝔽 do
6.    p ← argmin_{u∈𝕌−ℙ−𝔽} D(u)
7.    ℙ ← ℙ ∪ {p}
8.    if d(q,p) < r then
9.      r ← d(p,q)
10.     p* ← p
11.   end if
12.   for u ∈ 𝕌 − ℙ − 𝔽 do
13.     D_u^max ← max(D_u^max, |d(q,p) − d(u,p)|)
14.     if D_u^max > r then
15.       𝔽 ← 𝔽 ∪ {u}
16.     else
17.       D(u) ← D(u) + |d(q,p) − d(u,p)|
18.     end if
19.   end for
20. end while
21. return p*
```

iAESA

```
1.  Let ℙ ← ∅ set of pivots
2.  Let 𝔽 ← ∅ set of filtered elements
3.  r ← ∞, Π_q ←<>
4.  for u ∈ 𝕌, F(u) ← 0, Π_u ←<>
5.  for u ∈ 𝕌, D_u^max ← 0
6.  while 𝕌 ≠ ℙ ∪ 𝔽 do
7.    p ← argmin_{u∈𝕌−ℙ−𝔽} F(u)
8.    ℙ ← ℙ ∪ {p}
9.    insert p in Π_q
10.   if d(q,p) < r then
11.     r ← d(p,q)
12.     p* ← p
13.   end if
14.   for u ∈ 𝕌 − ℙ − 𝔽 do
15.     D_u^max ← max(D_u^max, |d(q,p) − d(u,p)|)
16.     if D_u^max > r then
17.       𝔽 ← 𝔽 ∪ {u}
18.     else
19.       insert p in Π_u
20.       F(u) ← F(Π_q, Π_u)
21.     end if
22.   end for
23. end while
24. return p*
```

Fig. 1. AESA and iAESA algorithms to retrieve the nearest neighbor.

AESA initially selected p_1. The next pivots are p_2 (step-1), p_3 (step-2), and p_4 (step-3), as they are, successively, the ones minimizing the D_1 distance to q.

On the other hand, in the process of iAESA, p_1 and p_2 are selected in the same way as AESA. We have drawn the permutation associated to the elements at this point. Note that Π_{p_4} is the same $\Pi_q = (21)$, therefore p_4 is the next (and last) pivot. In this example iAESA uses the pivots (p_1, p_2, p_4), one less than AESA.

The CPU time complexity of AESA is $O(|\mathbb{P}| \cdot n)$: $|\mathbb{P}|$ is the number of iterations over the elements not yet discarded, and $D(u)$ is updated in constant time per candidate in each iteration. The CPU time complexity of iAESA is higher because we need $O(|\mathbb{P}|)$ time to update Π_u and the corresponding value of $F(\Pi_u, \Pi_q)$, totalizing $O(|\mathbb{P}|^2 \cdot n)$. In metric space searching, this increase in the CPU time complexity is tolerated when the algorithm reduces the number of distance evaluations computed, as we show is the case of iAESA in Section 4.

iAESA2: A Combined Criterion. The similarity among permutations is effective only when we have added sufficient pivots to $\mathbb{P}$. Before that, it discriminates little among different zones of the space. In particular, permutations give no clue on how to select the second pivot, whereas D_1 gives an effective criterion. In general, permutations formed with few pivots will have many ties.

We propose a variant of iAESA, called iAESA2, that uses Spearman Footrule as

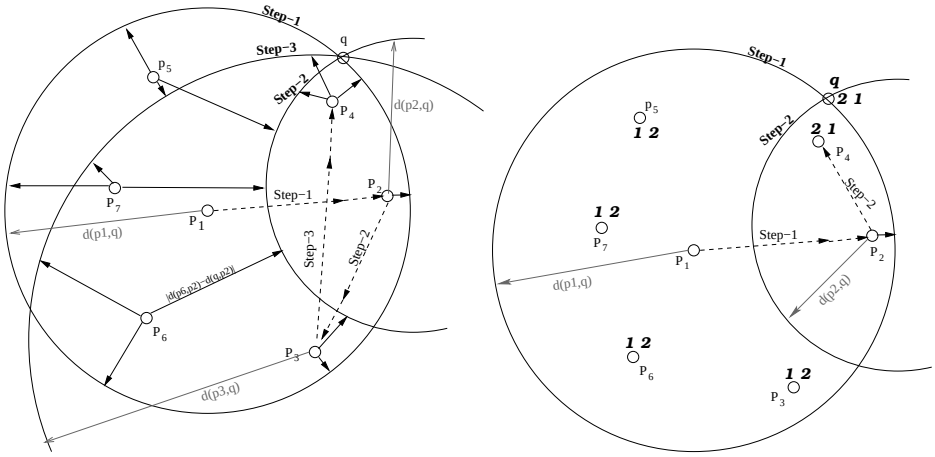


Fig. 2. On the left, the example shown in [Vidal 1986] to explain AESA. On the right, iAESA process for the same set of elements. The order of selection is step-1, step-2 and step-3. Note that iAESA uses one pivot less than AESA in this example.

its primary criterion, and D_1 as a secondary criterion to break ties in F.

The time complexity of iAESA2 is also $O(|\mathbb{P}|^2 \cdot n)$.

Probabilistic iAESA. As described at the end of the Introduction, every exact algorithm can be turned into probabilistic by preempting it after some amount of work has been carried out. In particular, a probabilistic version of AESA, iAESA and iAESA2 for nearest neighbor search consists in reviewing objects up to some fraction of the database and report the closest object found so far.

4. EXPERIMENTAL RESULTS

We experimented on different synthetic and real-life metric databases. The real-life metric spaces are TREC-3 documents under the cosine distance [Baeza-Yates and Ribeiro-Neto 1999], and a database of feature vectors of face images under Euclidean distance [Navarrete and Ruiz-del-Solar 2002]. The synthetic metric space examples are random vectors in the unitary cube under the Euclidean distance. Each point in each plot is obtained as an average over 100 different runs.

4.1 Exact iAESA

4.1.1 *Unitary Cube.* The performance of state-of-the-art proximity searching algorithms when answering both range and k-nearest neighbor queries worsens as the dimension of the space grows [Chávez et al. 2001; Böhm et al. 2001]. Therefore, it is interesting to experiment in spaces with different dimensions. A way to control the dimension of the space is to generate synthetic sets uniformly distributed in the unitary cube, and use these sets as abstract metric spaces.

We experimented with dimensions 4 to 14, for databases of size from 5,000 to 20,000 elements. The performances of AESA and iAESA for k-NN searching are compared in Figure 3. As expected, increasing the dimension of the data makes the problem more difficult. However iAESA performs consistently (up to 17%) better

than AESA for small k. For larger k our technique worsens compared to AESA. For example, in dimension 14, AESA takes over iAESA for $k > 5$. iAESA2 had the same performance as iAESA, hence it is omitted in this experiment. In any case, the differences are not so significant as in the real-life metric spaces considered in the next sections.

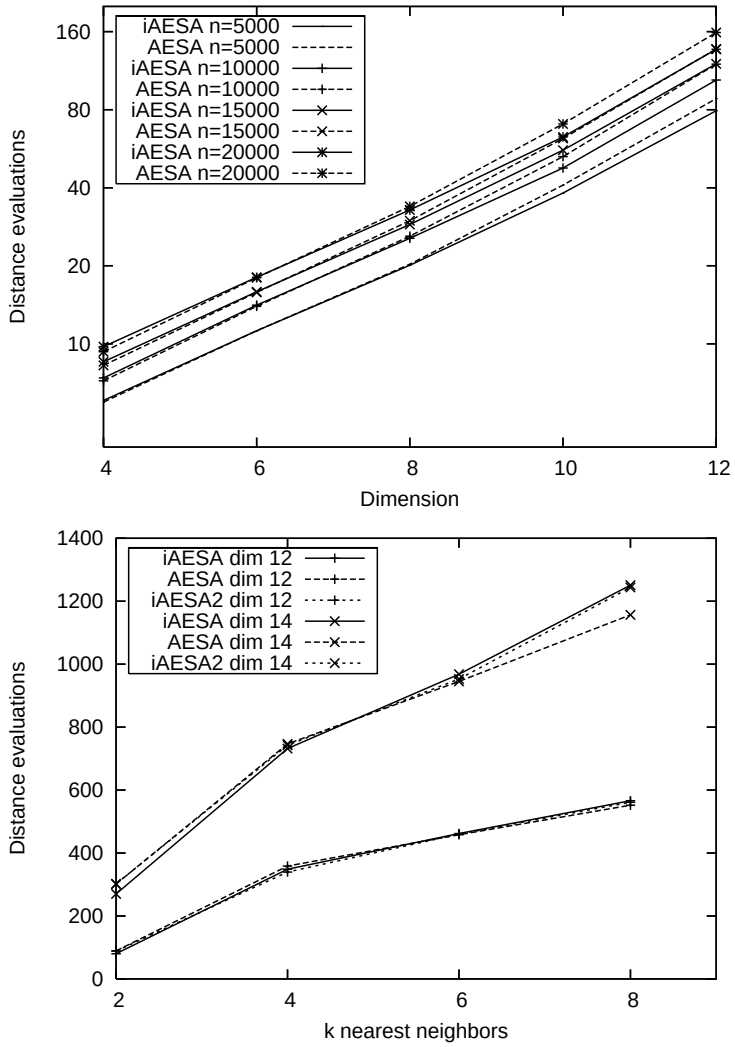


Fig. 3. Performance of our technique against AESA for different dimensions (top) for 2-NN queries. On the bottom, the dimension is 12 and 14 and $n = 5,000$, and we retrieved different number of nearest neighbors.

Just to confirm that AESA is by far the best performing index, we compare in Figure 4 AESA and iAESA with List of Clusters [Chávez and Navarro 2005],

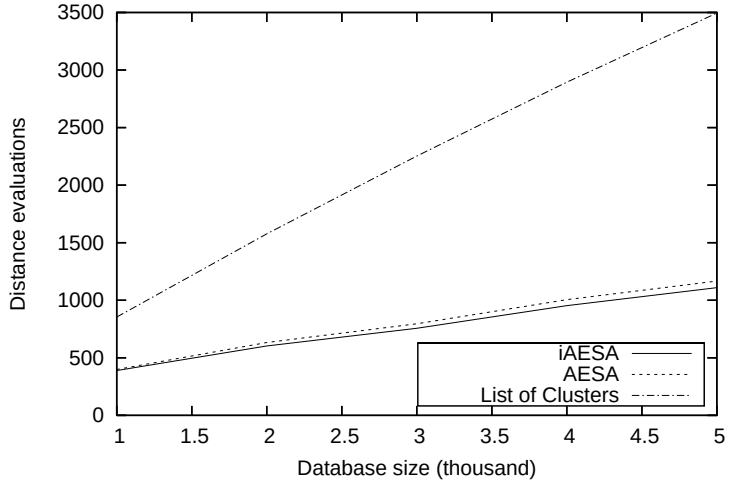


Fig. 4. Comparing our technique with AESA and List of Clusters on the unitary cube of dimension 16. Retrieving 5 nearest neighbors.

which is reported as an extremely efficient (linear-space) index for dimensions 14–20. AESA and iAESA are at least twice as fast in terms of distance computations.

4.1.2 *Documents.* Collection TREC-3, formed by 1,265 English files obtained from different sources, was indexed. We compare the documents using the vector space model, taking the angle between vectors as the distance measure (this is inversely proportional to the cosine similarity [Baeza-Yates and Ribeiro-Neto 1999]).

Figure 5 (top) shows the results for k-NN searching. The number of distance computations of iAESA grows slower than AESA as k grows, and for $k > 6$ iAESA takes over. It can also be seen that iAESA2 is clearly better than both AESA and iAESA. iAESA2 takes as low as 60% of the distance computations made by AESA.

As we have mentioned, iAESA and iAESA2 have higher asymptotic upper bound on CPU cost compared with AESA. In Figure 5 (bottom) we show that iAESA2 is, in practice, better than AESA even in terms of CPU time.

As a sanity check, we compared the results of AESA against the heuristic of choosing the next pivot at random. This turned out to make four times more evaluations than AESA.

4.2 Probabilistic AESA and iAESA

With a probabilistic algorithm we can handle much higher dimensions, in exchange for losing a few elements. A new parameter related to the quality of the answer has to be taken into account. This is measured in two ways. The first is the number of queries for which the probabilistic algorithm finds the correct k nearest neighbors after scanning a percentage of the database. The second is, for the queries where the correct answer is not obtained, the *relative error*: distance to the k-th NN found by the algorithm divided by the distance to the correct k-th NN.

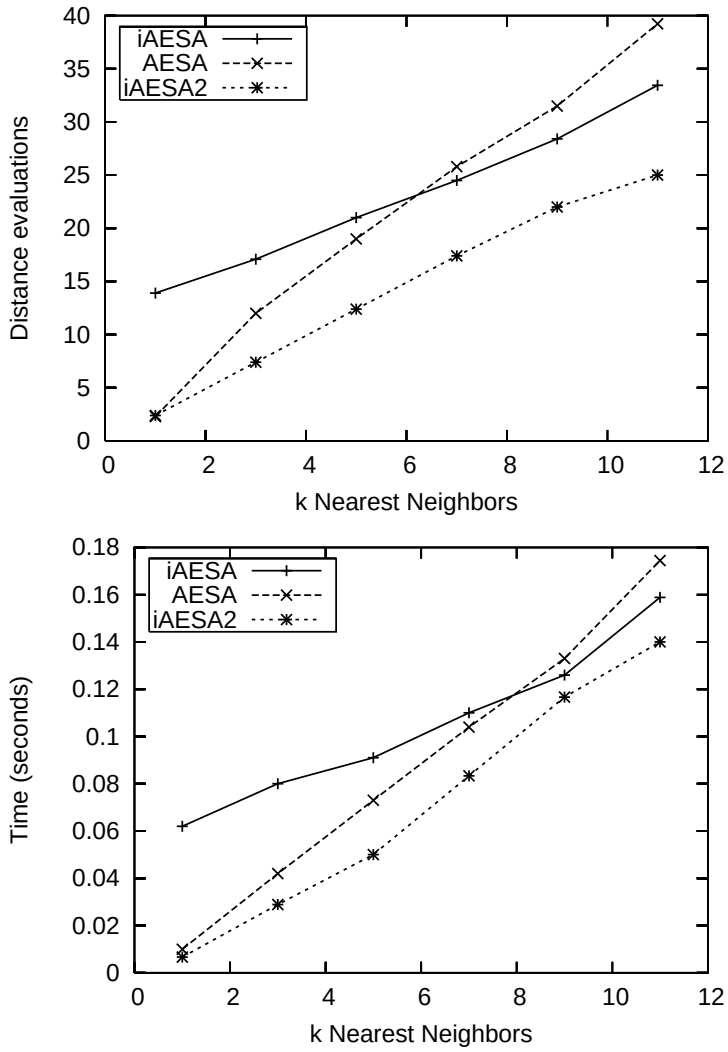


Fig. 5. Comparing the performance of our technique against AESA on a document database (1,265 documents). We show both number of distance computations (top) and CPU time (bottom).

4.2.1 *Unitary Cube.* We generated 3,000 synthetic random vectors in the unitary cube of 128 dimensions. No exact algorithm is able to avoid a full sequential scan in 128 dimensions. Figure 6 (top) shows the percentage of successful queries when the number of objects compared with the query is limited. iAESA and iAESA2 perform similarly, solving more correct queries than AESA as soon as we traverse a very small fraction of the database. On the bottom we show the relative error in unsuccessful queries, which are very low in all cases.

Figure 7 compares probabilistic iAESA and iAESA2 against a linear-space probabilistic algorithm based on the permutation technique described here. This algorithm was shown by Chávez et al. [2008] to be by far better than existing alterna-

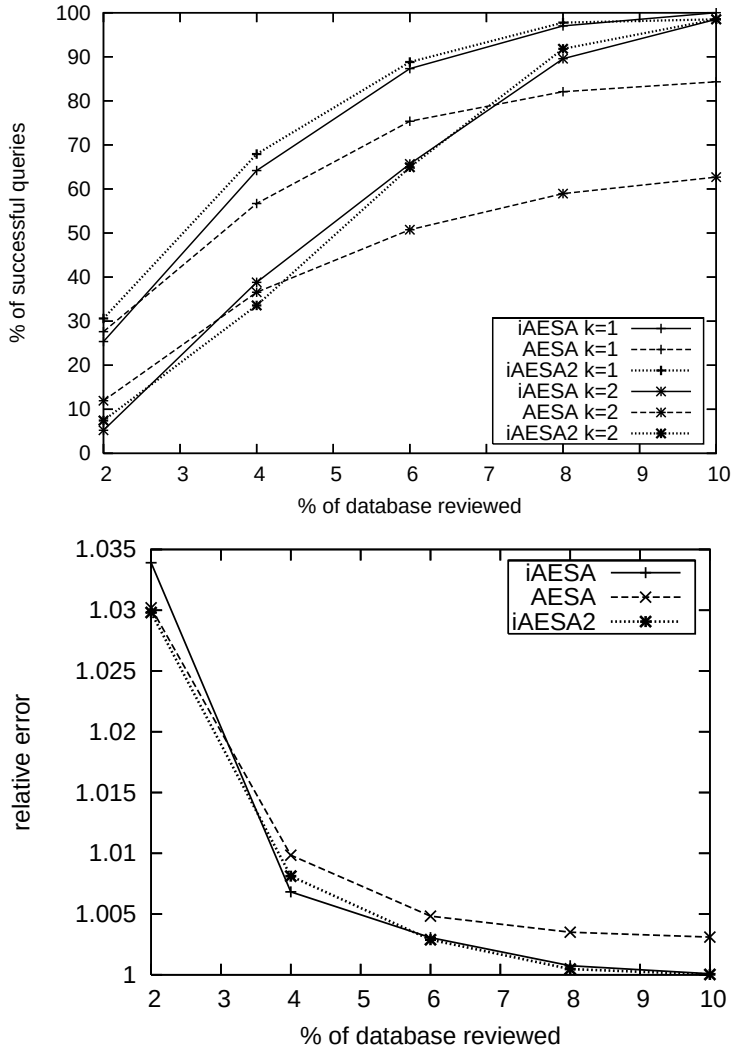


Fig. 6. Searching for k-NN queries on synthetic random vectors uniformly distributed in the unitary cube with 128 dimensions and 3,000 objects. On top the percentage of successful queries (higher is better) and on the bottom the relative error for unsuccessful queries.

tives, e.g. Bustos and Navarro [2003]. It consists in choosing a *fixed* set of pivots, sorting the database according to the permutations of that set, and traversing a percentage of the sorted database. The figure shows that the alternative probabilistic method is very far from achieving the same performance of iAESA or iAESA2.

Approximate proximity search has been recently surveyed by Patella and Ciaccia [2009]. This is a more general model where a relaxation on the accuracy of the answer is allowed. With the experimental result shown above, we conjecture that probabilistic AESA-like algorithms can serve as a baseline for approximate proximity searching algorithms.

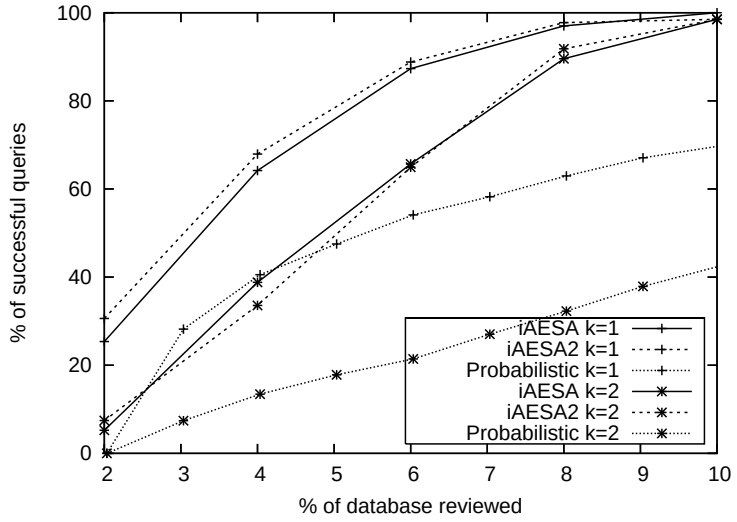


Fig. 7. Comparing iAESA with a linear-space version of the probabilistic algorithm using random vectors on dimension 128, 60 pivots and n=3000.

4.2.2 *FERET Face Images Database.* Many real databases are composed of few objects, each of them with very high intrinsic dimension. This is the case of the FERET database of face images [Phillips et al. 1998]. We used a target set with 762 images of 254 different classes (three images per class), and a set of 254 queries (1 image per class). Here each class is a person, the three images in the class are photos taken in different angles, and the corresponding query is a fourth photo of the same person. The intrinsic dimension of the database is around 40 when measured according to Chávez et al. [2001] or Navarrete and Ruiz-del-Solar [2002]. This is considered intractable for indexing because a search with or without index takes about the same time: Using the exact version of AESA ends up comparing 90% of the database. The performance of probabilistic AESA, iAESA and iAESA2 on this database is presented in Figure 8. It can be seen that, as k grows, the number of distance computations to achieve a given percentage of correct answers grows accordingly. Yet, in all cases, iAESA2 performs consistently better than the others. This time the relative errors are a bit higher, although still acceptable.

5. REDUCING CPU TIME

We have established a new baseline, both for exact and probabilistic algorithms, in terms of the number of distance computations. In this section we turn our attention to the extra CPU time involved in the process. In many metric spaces the extra CPU time is irrelevant because distance computations are very expensive (e.g., we have shown in Figure 5 that iAESA was faster than AESA even considering its higher CPU complexity per pivot processed). Yet, there are cases where the extra CPU time deserves attention. In this section we present a couple of ideas in the aim of reducing extra CPU time.

The core of the cost in iAESA is, for every new pivot $p = p_j$ introduced in $\mathbb{P}$,

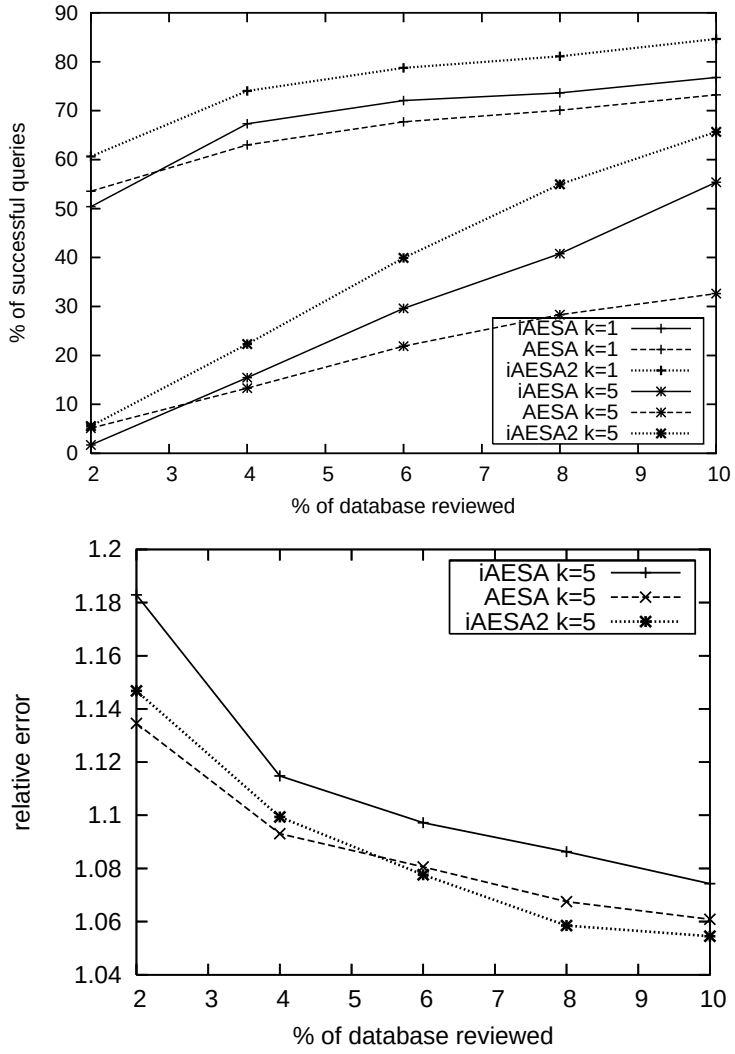
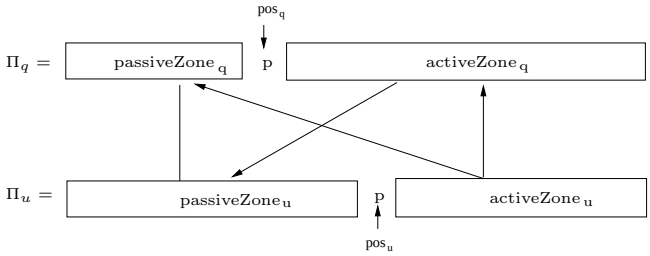


Fig. 8. Searching for k-NN queries in a real-life image database of 762 faces.

(i) to recompute the new permutations Π_u for all the remaining candidates u, and (ii) to recompute the new values $F(\Pi_q, \Pi_u)$ as well, using Eq. (3).

Incremental updating of F. A first idea is to update F incrementally as we insert the index j of the new pivot p_j at its correct position in Π_u. Let $pos_u = \Pi_u^{-1}(j)$ be that position. Similarly, we call $pos_q = \Pi_q^{-1}(j)$ the position where $p = p_j$ has been inserted into Π_q. We can divide the permutation into an *active* and a *passive* zone. The passive zone P_q is the part of the permutation Π_q before pos_q and the active zone A_q is the part after pos_q, as illustrated below.

Likewise, let us call P_u and A_u the passive and active zones of Π_u. The new value of F can be computed from the current one by comparing the interactions of active and passive zones. We have four interactions to analyze, as illustrated next.



Note that indices that are in the same zone (active or passive) in both permutations will not alter the value of F. Thus we have to focus on the other two cases. We precompute Π_q^{-1} and traverse each Π_u backwards while searching for the insertion point of j. The idea is to update F while we insert j in Π_u.

We first assume that all the objects that are in A_q increase F (that is, they are in P_u and their positions precede those in A_q), and later correct the cases where this is wrong. We now traverse Π_u backwards. As long as we shift the positions to make room for j, we are in A_u. With Π_q^{-1} we know whether each element we shift is in A_q or P_q. If it is in P_q, we determine whether the positions are closer or farther after the shift and update F accordingly. If it is in A_q, we reverse the increment made by the default assumption that all the elements A_q were in P_u. Once we arrive at the insertion point of j, the computation of F is almost correct, except that it is possible that some default increments should have been decrements. That is, if an element in A_q was in P_u but its position in A_q preceded that in P_u, the shift in A_q decreases rather than increasing F. This is corrected by traversing P_u backwards, yet we only have to review until position pos_q.

Let us see an example. Let 12 be the index of the new pivot, and let the permutations before inserting 12 be:

$$\begin{array}{l} \Pi_q = (\ 7\ \ 9\ \ 8\ \ 5\ \ 4\ \ 6\ 10\ 11\ \ 3\ \ 2\ \ 1\ ) \\ \Pi_u = (\ 2\ 10\ \ 6\ 11\ \ 9\ \ 4\ \ 7\ \ 3\ \ 8\ \ 5\ \ 1\ ) \end{array}$$

we have $F(\Pi_q, \Pi_u) = 44$. After 12 is inserted, we have:

$$\begin{array}{l} \Pi_q = (\ 7\ \ 9\ \ 8\ \ 5\ \underline{12}\ \ 4\ \ 6\ 10\ 11\ \ 3\ \ 2\ \ 1\ ) \\ \Pi_u = (\ 2\ 10\ \ 6\ 11\ \ 9\ \ 4\ \ 7\ \ 3\ \underline{12}\ \ 8\ \ 5\ \ 1\ ) \end{array}$$

We start by increasing F by $|A_q| = 7$. Now we traverse A_u backwards. Index 1 is in A_q, so we decrement F to reverse the incorrect increment. Index 5 is in P_q and the distance increases, so we increment F. The same goes for index 8. We now have to correct the mistakes in P_u. Index 3 is in A_u and the default increment was correct, index 7 is in P_q so we do nothing. Index 4 was counted as an increment but it should have been a decrement, so we decrease F by 2. Finally we add 4 (the distance between both positions of 12) to F, to get its updated value $F = 54$.

We compared this incremental algorithm versus the brute-force one to recompute F in the unitary cube with 5,000 objects. Figure 9 shows the performance ratio in terms of number of operations for both algorithms, as well as total CPU times.

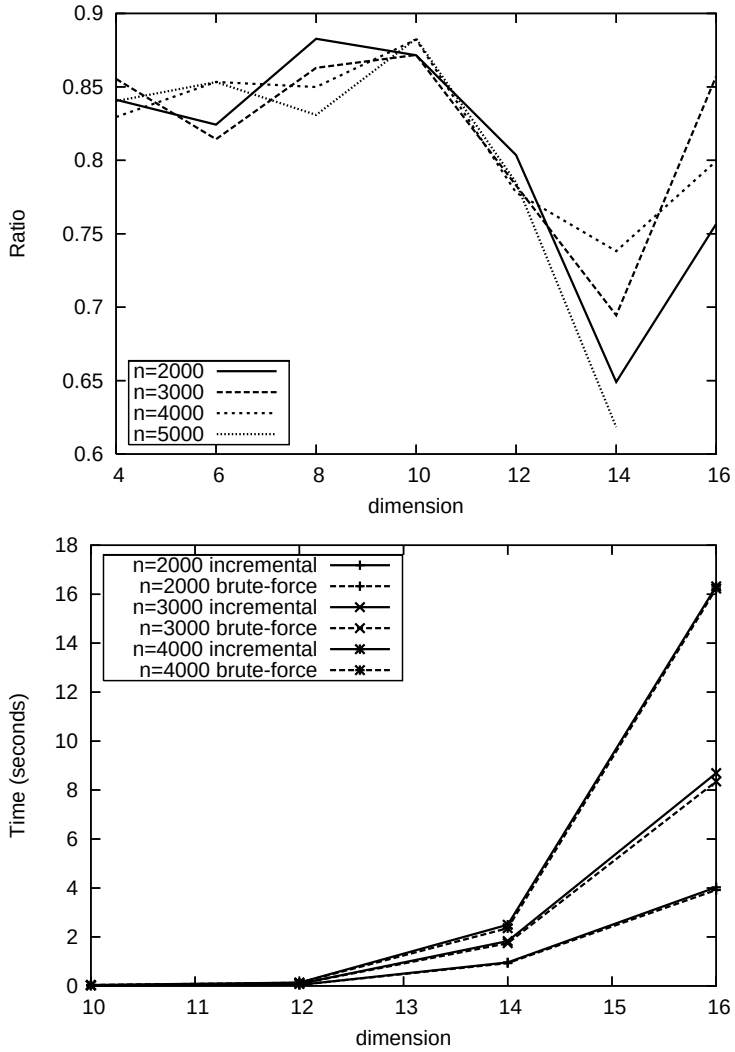


Fig. 9. On top, the cost ratio of incremental versus brute-force algorithm, in terms of number of operations. A smaller ratio implies fewer operations in the incremental algorithm. On the bottom, absolute CPU times.

The experiments show that it is possible to reduce the number of operations by almost 60%, but this does not heavily impact the CPU time. That is, although we traverse slightly more than half of the permutations on average, the more complex logic cancels the gains.

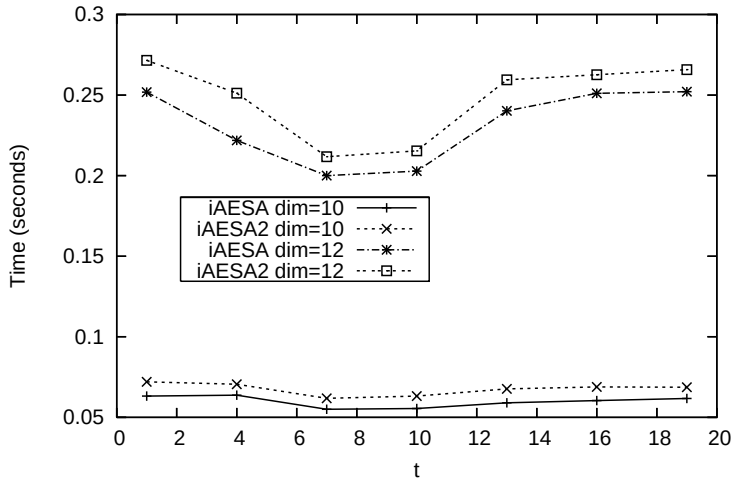


Fig. 10. Tradeoff in CPU time for iAESA and iAESA2, on 5,000 random points and retrieving 2 nearest neighbors.

Delayed recomputation. An alternative way to improve the performance of iAESA is to avoid recomputing all the Π_u and $F(\Pi_q, \Pi_u)$ values every time a new pivot is added to $\mathbb{P}$. Instead, we recompute those values every t pivots. In this case, instead of choosing the next pivot using the current F values, we choose the next t pivots at once, using algorithm IQS [Paredes and Navarro 2006]. A similar idea could be applied to AESA.

Parameter t yields a tradeoff in CPU time. As it grows, the quality of the pivots chosen decreases and thus the candidate sets are larger and the total number of distance computations may increase, but CPU time per processed candidate is reduced. The optimum value depends on the metric space and on the cost of computing the distance. If the distance is too expensive, a smaller value of t is preferable. Figure 10 shows experiments on uniformly distributed vectors, where the distance is relatively cheap to compute. In this case the best t values are in the range of 7 to 10, where reductions of up to 25% in the CPU time are achieved.

6. CONCLUSIONS AND FUTURE WORK

Proximity searching in metric spaces consists in retrieving the elements from a database that are close enough to a given query. The similarity between objects is measured by a distance function that is usually expensive to compute. AESA [Vidal 1986] has been without question, for 20 years, the most successful algorithm to solve similarity queries, because it computes the least number of distance evaluations to answer them.

The good performance of AESA is based on the elements that it chooses to compare with the query, called *pivots*. In this paper we proposed a better way to iteratively select the pivots. Our method establishes a new performance baseline for both exact and approximate proximity searching algorithms.

Our key result is a better predictor of the real distance between objects without evaluating it explicitly. This predictor is an ordering based on permutations that we proved to be superior to distance D_1, which had been the best choice for 20 years. Yet, one line of work is trying to find even better estimators, as this is the key to obtain improved performance.

Another related issue is the extra CPU cost to compute this estimator. Our new estimator is slightly more expensive than that of AESA but, since it achieves a reduced number of distance evaluations, it obtaines better overall performance compared to AESA when the distance is sufficiently expensive to compute. We have presented some techniques that achieve improved CPU time, yet we have not solved the basic data structuring problem that arised. This problem, which can be of independent interest, can be stated as follows: Design a data structure that stores two permutations (those are our Π_q and Π_u) supporting the following operations: (1) create a pair of empty permutations; (2) insert element $j+1$ anywhere in each of the two permutations of $(1 \dots j)$; (3) compute F between both permutations (Eq. (3)). Is it possible to execute m operations in $o(m^2)$ time?

Finally, it is always interesting to achieve versions of AESA-like structures that approach their performance using less space, albeit, as we have already mentioned, all the research in metric indices can be regarded under this light.

REFERENCES

Arya, S., Mount, D., Netanyahu, N., Silverman, R., and Wu, A. 1994. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA'94)*. ACM Press, New York, 573–583.

Baeza-Yates, R. and Ribeiro-Neto, B. 1999. *Modern Information Retrieval*. Addison-Wesley, New York.

Böhm, C., Berchtold, S., and Keim, D. A. 2001. Searching in high-dimensional spaces—Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR) 33,* 3, 322–373.

Bustos, B. and Navarro, G. 2003. Probabilistic proximity search algorithms based on compact partitions. *J. of Discrete Algorithms (JDA) 2,* 1, 115–134.

Chávez, E., Figueroa, K., and Navarro, G. 2008. Effective proximity retrieval by ordering permutations. *IEEE Trans. on Patt. Analysis and Machine Intelligence (TPAMI) 30,* 9, 1647–1658.

Chávez, E. and Navarro, G. 2003. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Inf. Process. Lett. (IPL) 85,* 1, 39–46.

Chávez, E. and Navarro, G. 2005. A compact space decomposition for effective metric indexing. *Patt. Recognition Lett. (PRL) 26,* 9, 1363–1376.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. 2001. Proximity searching in metric spaces. *ACM Computing Surveys (CSUR) 33,* 3, 273–321.

Ciaccia, P. and Patella, M. 2002. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst. (TODS) 27,* 4, 398–437.

Clarkson, K. 1999. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry 22,* 1, 63–93.

Fagin, R., Kumar, R., and Sivakumar, D. 2003. Comparing top k lists. *SIAM J. Discrete Math. (SIDMA) 17,* 1, 134–160.

Fredriksson, K. 2007. Engineering efficient metric indexes. *Patt. Recognition Lett. (PRL) 1,* 28, 75–84.

Hjaltason, G. R. and Samet, H. 2003. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst. (TODS) 28,* 4, 517–580.

Micó, L., Oncina, J., and Carrasco, R. 1996. A fast branch and bound nearest neighbour classifier in metric spaces. *Patt. Recognition Lett. (PRL) 17*, 731–739.

Micó, L., Oncina, J., and Vidal, E. 1994. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Patt. Recognition Lett. (PRL) 15*, 9–17.

Navarrete, P. and Ruiz-del-Solar, J. 2002. Analysis and comparison of eigenspace-based face recognition approaches. *Intl. J. of Patt. Recognition and Artificial Intelligence (IJPRAI) 16,* 7, 817–830.

Navarro, G., Paredes, R., and Chávez, E. 2007. t-spanners for metric space searching. *Data & Knowledge Engineering (DKE) 63,* 3, 820–854.

Paredes, R. and Navarro, G. 2006. Optimal incremental sorting. In *Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX'06)*. SIAM Press, 171–182.

Patella, M. and Ciaccia, P. 2009. Approximate similarity search: A multi-faceted problem. *J. of Discrete Algorithms (JDA) 7,* 1, 36–48.

Phillips, P., Wechsler, H., Huang, J., and Rauss, P. 1998. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing 16,* 5, 295–306.

Samet, H. 2006. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, New York.

Vidal, E. 1986. An algorithm for finding nearest neighbors in (approximately) constant average time. *Patt. Recognition Lett. (PRL) 4*, 145–157.

Vilar, J. 1995. Reducing the overhead of the AESA metric-space nearest neighbor searching algorithm. *Inf. Process. Lett. (IPL) 56*, 256–271.

White, D. and Jain, R. 1996. Algorithms and strategies for similarity retrieval. Tech. Rep. VCL-96-101, Visual Computing Laboratory, U. of California.

Zezula, P., Amato, G., Dohnal, V., and Batko, M. 2006. *Similarity search: the metric space approach*. Advances in Database Systems, vol. 32. Springer, Berlin.