# On the Least Cost For Proximity Searching in Metric Spaces ⋆

Karina Figueroa[1,2], Edgar Chávez[1], Gonzalo Navarro[2], and Rodrigo Paredes[2]

[1] Universidad Michoacana, México.
{karina,elchavez}@fismat.umich.mx
[2] Center for Web Research, Dept. of Computer Science, Universidad de Chile.
{gnavarro,raparede}@dcc.uchile.cl

**Abstract.** Proximity searching consists in retrieving from a database those elements that are similar to a query. As the distance is usually expensive to compute, the goal is to use as few distance computations as possible to satisfy queries. Indexes use precomputed distances among database elements to speed up queries. As such, a baseline is AESA, which stores all the distances among database objects, but has been unbeaten in query performance for 20 years. In this paper we show that it is possible to improve upon AESA by using a radically different method to select promising database elements to compare against the query. Our experiments show improvements of up to 75% in document databases. We also explore the usage of our method as a probabilistic algorithm that may lose relevant answers. On a database of faces where any exact algorithm must examine virtually all elements, our probabilistic version obtains 85% of the correct answers by scanning only 10% of the database.

## 1 Introduction

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, etc. In these applications there is a database (for example, a set of documents) and a similarity measure among its objects (for example, 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 distance function is normally considered quite expensive to compute, so that even I/O operations or the CPU cost of side computations are not considered. That is, the search complexity is taken as just the number of distance evaluations needed to answer a query, and thus the goal is to answer the queries by performing the minimum number of distance evaluations.

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.

⋆ Supported by CONACyT (Mexico) and Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, (Chile).

Different indexes store different information on the database distances [7]. 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 store just a range of distance values, and so on. 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 it requires $O(n^2)$ space, but it is applicable in some areas such as pattern recognition, as well as to index database subsets. In particular, using all the available information establishes a *baseline* on how good an index could be. Actually, all the development on metric space indexing can be regarded as the quest for maintaining good efficiency while reducing the amount of information stored [7].

The canonical algorithm that uses all the data is AESA [17]. For 20 years AESA has been the indexing technique requiring, by far, the least number of distance computations among all other indexes (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 specifically, AESA works by choosing a "pivot" from the remaining set of candidates and using it to prune more candidates. The closer the pivot to the query, the more effective the pruning is. We introduce a new technique called *iAESA* to choose the next pivot, which guesses better a close candidate and yields reductions in the number of distance evaluations of up to 75% in document databases.

In very high dimensions, even AESA and iAESA boil down to a sequential database scan. We explore the usage of iAESA as a probabilistic scheme that may lose some relevant answers, but could quickly find most of them. We show that, for example, on a database of face images where no exact algorithm can obtain any significant savings over a sequential scan, iAESA retrieves 85% of the correct answers by scanning just 10% of the database. This is 80% less than what would be needed to obtain the same result with probabilistic AESA.

## 2 Related Work

### 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 objects in $\mathbb{X}$. The distance function must satisfy the following properties: strict positiveness ($d(x,y) > 0 \iff x \neq y$), 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, $(q,r)_d = \{u \in \mathbb{U} \mid d(u,q) \leq r\}$

– $k$-nearest neighbor query, $k\text{NN}(q)_d = A$ 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 against 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 $q$ against 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 data [7]. When the dimension grows, the mean of a random distance increases and the variance diminishes. In high dimensions, there are no algorithms that can avoid sequential scan. AESA is also affected by dimension in spite of being the best proximity search algorithm in metric spaces.

### 2.2 AESA

The Approximating and Eliminating Search Algorithm (AESA) was introduced by E. Vidal in 1986 [17]. 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 against 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 with another technique and apply AESA on each partition [11].

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 [13] chooses $k$ elements of $\mathbb{U}$ as potential pivots, then reducing the space to $O(kn)$. An improved version of LAESA is Tree LAESA (TLAESA) [12] which achieves sublinear side computations at query time at the expense of doubling the number of distance computations on average. Reduced Overhead AESA (ROAESA) [18] strictly calculates the same distances as AESA but reduces the query processing time. Recently, graph $t$-spanner indexes [15] were used to simulate AESA, obtaining almost the same number of distance calculations and using much less memory. In fact, all the development on indexes for metric spaces can be seen as attempts to simulate the performance of AESA using less memory [7].

### 2.3 Searching Using AESA

Like most indexing algorithms, AESA solves nearest neighbor queries by choosing a *pivot* $u \in \mathbb{U}$ to compare against $q$, then filtering out as many candidates of $\mathbb{U}$ as possible, and repeating until all candidates are compared or discarded. AESA proposes a specific method to select the pivots: The next pivot to compare against $q$ is chosen as the candidate $u$ minimizing

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

where $\mathbb{P}$ are those pivots already compared against $q$ (thus the $d(p,q)$ distances are known, whereas the $d(u,p)$ distances are stored in the matrix). The goal of minimizing $D(u)$ is to find a pivot as close as possible to $q$. The algorithm to answer a closest neighbor query 1-NN$(q)_d$ is summarized in five steps.

1. **Initialization**. The sets of pivots $\mathbb{P}$ and filtered elements $\mathbb{F}$ are empty. Let $D(u) \leftarrow 0$ for $u$, $D_{max_u} \leftarrow 0$ and $r \leftarrow \infty$. Steps 2-5 are repeated until $\mathbb{U} = \mathbb{P} \cup \mathbb{F}$.
2. **Approximating**. In this step a new pivot $p$ is selected according to Equation (1). That is $p \leftarrow \mathrm{argmin}_{u \in \mathbb{U}-\mathbb{F}-\mathbb{P}} \; D(u)$.
3. **Distance computation**. Element $p$ is compared against the query $q$ by computing $d(p,q)$. The new $p$ will be added to the set of used pivots $\mathbb{P}$.
4. **Updating the NN**. If $d(q,p) < r$, the current nearest neighbor and $r$ are updated. Every object in $\mathbb{U} - \mathbb{F} - \mathbb{P}$ updates its approximation criterion according to Equation (1), that is $D(u) \leftarrow D(u) + |d(u,p) - d(p,q)|$ and $D_{max_u} \leftarrow \max(D_{max_u}, |d(u,p) - d(p,q)|)$.
5. **Eliminating**. Those $u \in \mathbb{U} - \mathbb{F} - \mathbb{P}$ such that $D_{max_u} > r$ are discarded using the triangle inequality. The elements filtered in this step are added to $\mathbb{F}$. The process continues at step 2.

The nearest neighbor query process of AESA is presented in Fig. 1 (left). Range query process $(q,r)_d$ can be implemented similarly by keeping $r$ fixed and reporting every $p$ that $d(p,q) \leq r$.

These algorithms generalize to $k$-NN queries, where $k > 1$, by maintaining a pool with the $k$ closest elements $p^*$ found until now, so that $r$ is the distance to the current $k$-th nearest neighbor.

### 2.4 Proximity Preserving Order

We introduce some terminology needed to explain our technique [5].

Let $\mathbb{P} \subseteq \mathbb{U}$. Every element $u \in \mathbb{U}$ defines a *preorder* $\leq_u$ in $\mathbb{P}$ given by the distance to $u$. It is defined for $y, z \in \mathbb{P}$, as $y \leq_u z \Leftrightarrow d(u,y) \leq d(u,z)$. The relation $\leq_u$ is a preorder and not an order because some elements can be at the same distance of $u$, and then $\exists y \neq z$ such that $y \leq_u z \;\wedge\; z \leq_u y$.

Every object $u$ can compute its preorder of $\mathbb{P}$ and associate it to a permutation, because the preorder induces a total order in the quotient set. Let us define

*AESA*

```
1.  Let $\mathbb{P} \leftarrow \emptyset$ set of pivots
2.  Let $\mathbb{F} \leftarrow \emptyset$ set of filtered elements
3.  $r \leftarrow \infty$
4.  For $u \in \mathbb{U}$, $D(u) \leftarrow 0$, $D_{max_u} \leftarrow 0$
5.  while $\mathbb{U} \neq \mathbb{P} \cup \mathbb{F}$ do
6.    $p \leftarrow \text{argmin}_{u \in \mathbb{U}-\mathbb{P}-\mathbb{F}} D(u)$
7.    $\mathbb{P} \leftarrow \mathbb{P} \cup \{p\}$
8.    if $d(q,p) < r$ then
9.        $r \leftarrow d(p,q)$
10.       $p^* \leftarrow p$
11.   for $u \in \mathbb{U} - \mathbb{P} - \mathbb{F}$ do
12.     $D_{max_u} \leftarrow \max(D_{max_u}, |d(q,p) - d(u,p)|)$
13.     if $D_{max_u} > r$ then
14.       $\mathbb{F} \leftarrow \mathbb{F} \cup \{u\}$
15.     else
16.       $D(u) \leftarrow D(u) + |d(q,p) - d(u,p)|$
17. return $p^*$
```

*iAESA*

```
1.  Let $\mathbb{P} \leftarrow \emptyset$ set of pivots
2.  Let $\mathbb{F} \leftarrow \emptyset$ set of filtered elements
3.  $r \leftarrow \infty$, $\Pi_q \leftarrow <>$
4.  For $u \in \mathbb{U}$, $F(u) \leftarrow 0, \Pi_u \leftarrow <>$
5.  For $u \in \mathbb{U}, D_{max_u} \leftarrow 0$
6.  while $\mathbb{U} \neq \mathbb{P} \cup \mathbb{F}$ do
7.    $p \leftarrow \text{argmin}_{u \in \mathbb{U}-\mathbb{P}-\mathbb{F}} F(u)$
8.    $\mathbb{P} \leftarrow \mathbb{P} \cup \{p\}$
9.    insert $p$ in $\Pi_q$
10.   if $d(q,p) < r$ then
11.       $r \leftarrow d(p,q)$
12.       $p^* \leftarrow p$
13.   for $u \in \mathbb{U} - \mathbb{P} - \mathbb{F}$ do
14.     $D_{max_u} \leftarrow \max(D_{max_u}, |d(q,p) - d(u,p)|)$
15.     if $D_{max_u} > r$ then
16.       $\mathbb{F} \leftarrow \mathbb{F} \cup \{u\}$
17.     else
18.       insert $p$ in $\Pi_u$
19.       $F(u) \leftarrow F(\Pi_q, \Pi_u)$
20. return $p^*$
```

**Fig. 1.** AESA and iAESA algorithms to retrieve the nearest neighbor (iAESA is described in Section 3).

$\Pi_u = p_1, p_2, \ldots, p_{|\mathbb{P}|}$ where $p_i \leq_u p_{i+1}$ the permutation of $u$. The elements at the same distance take an arbitrary but consistent order. We use $\Pi_u^{-1}(p_i)$ to identify the position of element $p_i$ in the permutation $\Pi_u$.

It is important to notice that two equal elements must have the same permutation, while two similar objects will hopefully have a similar permutation. So if $\Pi_u$ is similar to $\Pi_q$ we expect $u$ to be close to $q$.

Similarity between the permutations of $q$ and $u$ can be measured by Kendall Tau, Spearman Rho, or Spearman Footrule metric [10], among others. As all of these have a comparable predictive power [5], we choose Spearman Footrule because it is not expensive to compute. This measure is defined as follows:

$$F(u) = F(\Pi_u, \Pi_q) = \sum_{i=1}^{|\mathbb{P}|} |\Pi_u^{-1}(p_i) - \Pi_q^{-1}(p_i)|, \tag{2}$$

where $\mathbb{P}$ is the current set of pivots. For example, let $\Pi_q = p_1, p_2, p_3, p_4, p_5$ be the permutation of the query, and $\Pi_u = p_3,\ p_2, p_1, p_5, p_4$ the permutation of an element $u \in \mathbb{U}$. According to Equation (2), we have $F(\Pi_q, \Pi_u) = |1-3| + |2-2| + |3-1| + |4-5| + |5-4| = 6$.

## 3 Our Proposal: iAESA

From Section 2.2, we notice that the only way to improve the performance of AESA seems to be by modifying the approximation criterion, that is, by proposing a different method to select the next pivot.

We propose to select as the new pivot the element whose permutation is the most similar to the permutation of the query. We describe this process next.

### 3.1 Searching Using iAESA

The algorithm consists basically in modifying the approximation criterion, which will be replaced by the similarity between permutations. The permutations will be formed by the pivots already used.

Instead of $D(u)$, we will use $F(u)$, which is initialized at 0 and updated upon choosing a new pivot $p$ according to Equation (2). The new pivot will be the one with smallest $F(u)$. Fig. 1 (right) gives the algorithm.

### 3.2 Comparing AESA with iAESA

In Fig. 2 we compare iAESA with AESA, using the same example shown in [17]. The example retrieves the nearest neighbor. 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 (1); 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 which elements are to be chosen given the approximation criterion.

AESA initially selected $p_1$. The next pivot was $p_2$ because it minimizes Equation (1) (step-1). The next pivot is $p_3$ (step-2) and finally $p_4$ (step-4) according to the approximation criterion.

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 of the elements at this point. Note that $p_4$ has the same permutation as $\Pi_q = \{p_2, p_1\}$, 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 $O(|\mathbb{P}| \cdot n)$, as $|\mathbb{P}|$ is the number of iterations over the elements not yet discarded and $D(u)$ is updated in constant time. iAESA complexity is higher because we need $O(|\mathbb{P}|)$ time to update $\Pi_u$ and $F(u)$. This yields a total complexity of $O(|\mathbb{P}|^2 \cdot n)$.

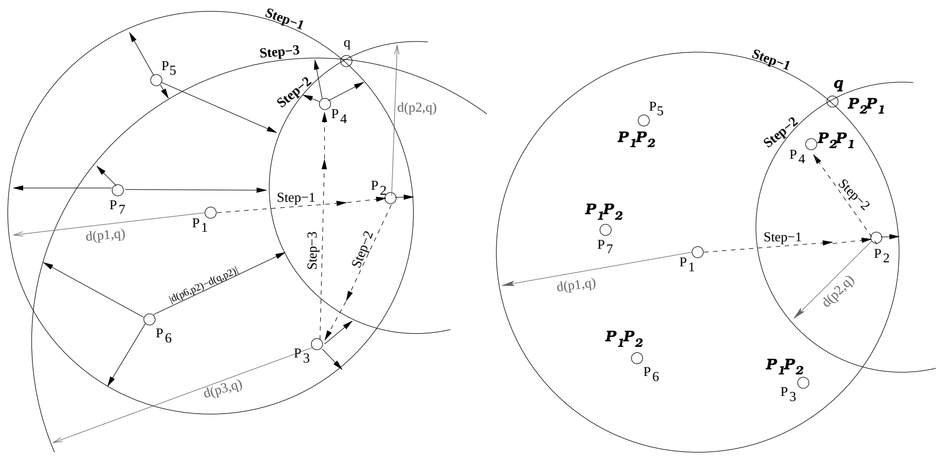


**Fig. 2.** On the left, the example shown in [17] to explain AESA. On the right, iAESA process for the same set of elements. The order of selection is step-1, step-2 and step-3. Note that iAESA uses one pivot less than AESA in this example.

### 3.3 Combining AESA and iAESA

AESA and iAESA criteria can be combined into an algorithm that we call *iAESA2*. The idea is to modify the approximation criterion of iAESA (i.e., the similarity between permutations) using AESA approximation criterion $D(u)$ to break ties in $F(u)$. These ties are common when there are few pivots. The CPU time complexity of iAESA2 is also $O(|\mathbb{P}|^2 \cdot n)$.

In other words, iAESA2 uses two approximation criteria: a primary one given by the least value of Spearman Footrule metric (i.e. the most similar permutation) and a secondary one given by the smallest $D(u)$.

## 4 Probabilistic iAESA

A serious problem of all algorithms in metric spaces, even for AESA, is that when the dimension of the space grows [7], the whole database needs to be reviewed. 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 did it find.

Probabilistic algorithms have been proposed both for vector spaces [1, 19] and for general metric spaces [9, 8, 6, 4]. In [4] they use a technique to obtain probabilistic algorithms that is relevant to this work. They use different techniques to sort the database according to some *promise value*. As they traverse the database in such order, they obtain more and more relevant answers to the query. A good database ordering is one that obtains most of the relevant answers by traversing a small fraction of the database. In other words, given a limited amount of work, 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 a good ordering of the database given a query $q$.

Under this model, a probabilistic version of $k$-NN AESA, iAESA and iAESA2 consists in reviewing objects up to some fraction of the database and reporting the $k$ closest object found until then. In Fig. 1, we should replace the **while** condition (line 5) by **while** $|\mathbb{P}| < \mathit{percentage\_of\_database}$. For range queries we would simply report any relevant element found until the scanning is stopped.

## 5 Experimental Results

We conducted experiments on different synthetic and real-life metric databases. The real-life metric spaces are TREC-3 documents under cosine distance [2], and a database of feature vectors of face images under Euclidean distance [14]. The synthetic metric spaces are random vectors in the unitary cube.

### 5.1 Exact iAESA: Unitary Cube

The performance of the existing algorithms, to answer both range and $k$-nearest neighbor queries, worsens as the dimension of the space grows [3]. Therefore, it is interesting to experiment with 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 this set as an abstract metric space. We experimented with 4 to 14 dimensions, for databases of size from 5,000 to 20,000 elements. The performance of our technique can be seen in Fig. 3. Notice that, as we increase the dimension of the data, the problem becomes more difficult. Nevertheless, iAESA retains its (slight) advantage over AESA when the dimension grows. In the best case, iAESA requires 17% less distance evaluations than AESA. iAESA2 had the same performance as iAESA, so we omitted it in this experiment. On the other hand, we note that iAESA loses its advantage over AESA as the number $k$ of nearest neighbors sought grows. For example, in dimension 14 AESA takes over for $k > 5$.

### 5.2 Exact iAESA: Documents

A set of 1265 English documents obtained from the Wall Street Journal 87-89 collection from TREC-3 was indexed. We compare the documents under the vector space, using the cosine distance [2].

Fig. 4 shows the results on this space. This time iAESA improves upon AESA as the number $k$ of nearest neighbors retrieved grows over 8. On the left we show distance computations. It can be seen that iAESA2 is clearly better than both AESA and iAESA in all cases, improving upon AESA by up to 75%, when $k = 1$. Fig. 4 (right) displays overall CPU time. It can be seen that, even though iAESA and iAESA2 suffer from a higher number of side CPU computations, they are still preferable over AESA.

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

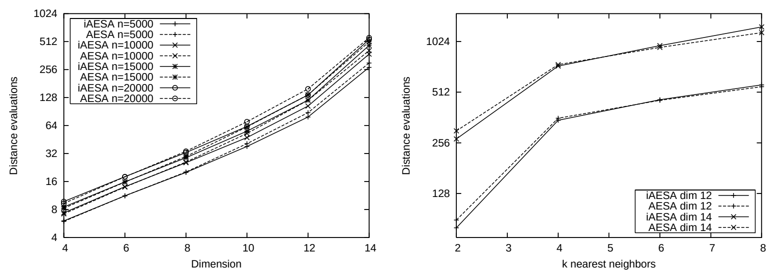


**Fig. 3.** Performance of our technique against AESA for different dimensions and $k = 2$ (left side). On the right, we retrieve different numbers $k$ of nearest neighbors, on dimensions 12 and 14 and $n = 5,000$. Note the logscale.

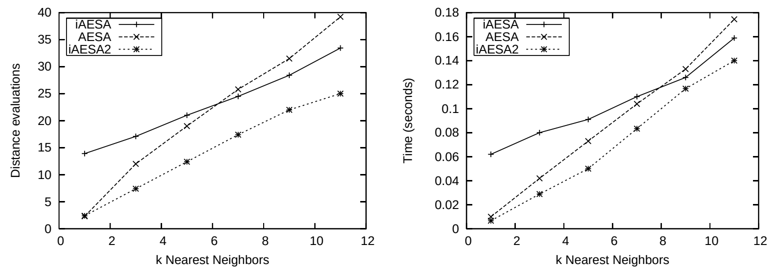


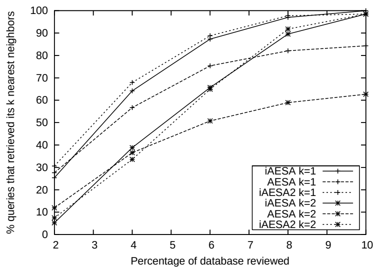
**Fig. 4.** Comparing the performance of our technique against AESA on a document database (1265 documents).

## 5.3 Probabilistic iAESA: Unitary Cube

We experimented with 3,000 synthetic random vectors in the unitary cube of 128 dimensions. Any exact algorithm is forced to compare every element in such a high-dimensional space. Fig. 5 (left) shows the percentage of successful queries when the number of objects compared against the query is limited. That is, we plot the percentage of queries that retrieved all their correct $k$ nearest neighbors after scanning a fraction of the database. We can see that iAESA finds the $k$ nearest neighbors faster than AESA, and that iAESA2 is the fastest for large $k$. AESA, on the other hand, needs to compare almost 80% of the database in order to retrieve 95% of the answer.

On the right we show the ratio among the distance to the $k$-th NN found by the algorithm versus the distance to the true $k$-th NN. In this computation we exclude the queries where the algorithm finds all the $k$ correct neighbors, that is, on queries considered unsuccessful on the left plot.

Note that we have defined a query as unsuccessful even if it finds $k-1$ out of the $k$ correct elements. Fig. 6 plots the fraction of the correct nearest neighbors found as we scan the database. For example, for with iAESA2 we need to scan about 7% of the database to find 90% of the answers.



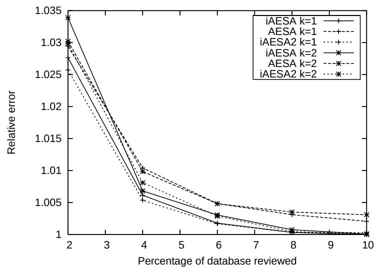


**Fig. 5.** Searching for $k$ nearest neighbors on 3,000 random vectors in 128 dimensions. On the left, fraction of correct queries. On the right, distance approximation ratio for the unsuccessful queries.

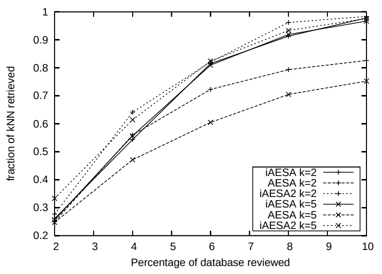


**Fig. 6.** Fraction of the answer retrieved as the scanning progresses. Random vectors on dimension 128 and $n = 3000$.

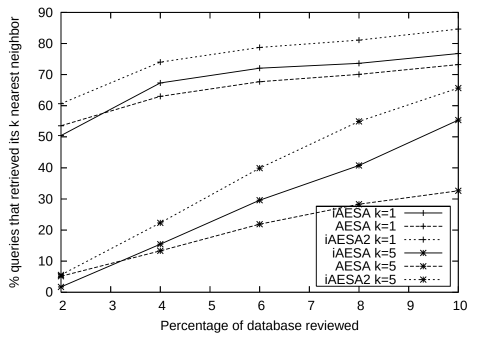
### 5.4 Probabilistic iAESA: Face Images

Many real databases are composed of few objects, each of very high intrinsic dimension. This is the case the FERET database of face images [16]. We use a target set with 762 images of 254 different classes (3 similar images per class), and a query set of 254 images (1 image per class). The intrinsic dimension of the database is around 40, which is considered intractable: exact AESA and iAESA must scan 90% of the database in order to answer $k$-NN queries on this space.

The performance of the probabilistic algorithms is compared in Fig. 7. It can be seen that larger fractions of the database must be scanned in order to satisfactorily solve queries with larger $k$. Again, iAESA2 is faster than the others.

## 6 Conclusions and Future Work

Proximity searching in metric spaces consists in retrieving the elements from the database that are relevant to a given query. The similarity between objects is measured by a distance function that is usually expensive to compute. AESA [17] 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. We present a new technique, called iAESA, able to improve upon AESA by up to 75% over different metric databases.



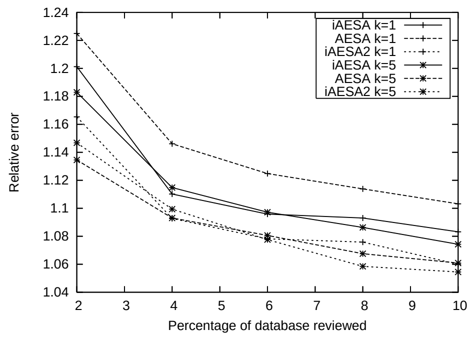


**Fig. 7.** Searching for the $k$ nearest neighbors in a real-life image database of 762 faces. On the left, fraction of correct queries. On the right, ratio of distances to the $k$-th NN found versus the real $k$-th NN.

In very high dimensions there are no exact algorithms able to avoid sequential scan. We propose a new probabilistic algorithm based on iAESA, which is able to solve a large fraction of queries by scanning a small fraction of the database. For example, on a faces image database, iAESA solves 85% of the queries by checking just 10% of the database.

The only weak point of our approach is the extra CPU time required to reduce the distance computations. This can be significant if the distances are not very expensive to compute. We plan to address this issue in two ways. One is to avoid, upon the insertion of a new pivot, the full recomputation of the permutation of each element as well as its distance to the query permutation. We are exploring a scheme that reduces this work by about 50%, and further reductions could be possible by using smarter data structures. Another idea is based on the fact that the distances to the query permutation can only grow as more pivots are inserted. Thus we can delay the updating of the permutation of every element until it would become the next pivot. At this point the pivot insertions delayed are carried out on the candidate's permutation and its distance to the query permutation is updated. This may push the candidate behind on the priority queue and forces us to choose the next best candidate, until we get an up-to-date next candidate. Thus many elements could be removed from the candidate set without ever having updated their permutations, thus saving CPU time.

## References

1. S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 573–583, 1994.
2. R. Baeza-Yates and B. Ribeiro. *Modern Information Retrieval*. Addison-Wesley, 1999.

3. C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces-index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
4. B. Bustos and G. Navarro. Probabilistic proximity search algorithms based on compact partitions. *Journal of Discrete Algorithms (JDA)*, 2(1):115–134, 2003.
5. E. Chávez, K. Figueroa, and G. Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *MICAI 2005: Advances in Artificial Intelligence*, volume 3789 of *LNCS*, pages 405–414, 2005.
6. E. Chávez and G. Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters*, 85(1):39–46, 2003.
7. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
8. P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. on Database Systems*, 27(4):398–437, 2002.
9. K. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry*, 22(1):63–93, 1999.
10. R. Fagin, R. Kumar, and D. Sivakumar. Comparing top $k$ lists. *SIAM J. Discrete Math*, 17(1):134–160, 2003.
11. K. Fredriksson. Parallel and memory adaptive metric indexes. *Pattern Recognition Letters*, to appear.
12. L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.
13. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
14. P. Navarrete and J. Ruiz-Del-Solar. Analysis and comparison of eigenspace-based face recognition approaches. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 16(7):817–830, 2002.
15. G. Navarro, R. Paredes, and E. Chávez. $t$-spanners as a data structure for metric space searching. In *SPIRE 2002: 9th International Symposium on String Processing and Information Retrieval*, LNCS 2476, pages 298–309, 2002.
16. P. Phillips, H. Wechsler, J. Huang, and P. Rauss. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing Journal*, 16(5):295–306, 1998.
17. E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.
18. J. Vilar. Reducing the overhead of the AESA metric-space nearest neighbor searching algorithm. *Information Processing Letters*, 56:256–271, 1995.
19. D. White and R. Jain. Algorithms and strategies for similarity retrieval. Technical Report VCL-96-101, Visual Computing Laboratory, U. of California, 1996.