ELSEVIER

Pattern Recognition Letters 17 (1996) 731-739

Pattern Recognition Letters

A fast branch & bound nearest neighbour classifier in metric spaces *

Luisa Micó *, Jose Oncina, Rafael C. Carrasco

Departamento de Lenguajes y Sistemas Informáticos, Ap. Correos 99, Universidad de Alicante, E-03080 Alicante, Spain

Received 13 October 1995; revised 1 February 1996

Abstract

The recently introduced algorithm LAESA finds the nearest neighbour prototype in a metric space. The average number of distances computed in the algorithm does not depend on the number of prototypes but it shows linear space and time complexities. In this paper, a new algorithm (TLAESA) is proposed which has a sublinear time complexity and keeps the other features unchanged.

Keywords: Metric spaces; Nearest-neighbour; Pattern recognition

1. Introduction

One of the most popular techniques in Pattern Recognition is the nearest neighbour search. In this method, the sample is classified in the same class as the prototype which minimizes a predefined distance function with respect to the sample. For instance, in handwritten character recognition the set of prototypes consists of many (classified) examples of letters and numerals, the sample is the character to be classified and the edit distance has often been used in order to estimate the similarity between the sample and every prototype. However, computing all the distances between the sample and the prototypes becomes usually unfeasible, as the distance function is often highly time-expensive.

For three decades, a wide variety of algorithms to face this problem has been proposed (Dasarathy, 1991). Some methods (Friedman et al., 1977; Ramasubramanian and Paliwal, 1990) make use of properties which are intrinsic to the representation of the data and therefore are not general. However, those algorithms using only the intrinsic properties of the distance function (reflexivity, symmetry and the triangular inequality) can always be applied. Examples of such methods can be found in (Fukunaga and Narendra, 1975; Kalantari and McDonald, 1983; Vidal, 1986, 1994, Micó et al., 1994; Shasha and Wang, 1990). Depending on their complexity, they can be be categorized into two groups:

1. algorithms (Fukunaga and Narendra, 1975; Kalantari and McDonald, 1983), with a sublinear time complexity, and
2. algorithms (Vidal, 1986, 1994; Micó et al., 1994) such that the average number of distance computations is bounded by a constant which does not depend on the size of the prototype set.

Up to now, no algorithm has been developed which

* Work partially supported under grant TIC93-0633-C02-02.

* Corresponding author. E-mail: mico@dtic.ua.es.

PII S0167-8655(96)00032-3

has simultaneously both properties 1 and 2. In this work, such an algorithm is proposed. This algorithm (TLAESA or Tree LAESA) is based on the LAESA (Linear AESA) in (Micó et al., 1994) which, in turn, is based on the AESA (Approximating Eliminating Search Algorithm) (Vidal, 1986, 1994). In the AESA, the expected number of distance computations is bounded by a constant, but its time complexity is linear and the space complexity is quadratic. This last feature makes AESA impractical for large prototype sets. In order to overcome this drawback the LAESA algorithm was introduced. LAESA has a linear space complexity while keeping the other features unchanged.

The algorithm presented here (TLAESA) is an improvement over the LAESA and reduces its average time complexity to sublinear. The TLAESA algorithm consists of two parts: (1) preprocessing and (2) search. In the preprocessing algorithm, two structures are built: a binary search tree storing all prototypes and a table of distances. The search algorithm is essentially a traversal of the binary tree where the table of distances is used in order to avoid the exploration of some branches of the tree.

2. The preprocessing

During preprocessing the algorithm builds two different structures: the distance table and the prototype tree. The distance table M stores the distances between every prototype in the set of prototypes P and a selected subset of them called *base prototypes*. Base prototypes will serve as reference points during the pruning process and therefore, it is advisable to choose them as much separated as possible. One of the best strategies for selecting a subset of n base prototypes $B = \{b_1, \ldots, b_n\}$ is proposed in (Micó et al., 1992, 1994). The first one, b_1, is randomly selected and then, for $i = 2, 3, \ldots, n$,

$$b_i = \text{argmax}_{p \in (P - B_i)} \sum_{k=1}^{i-1} d(p, b_k), \tag{1}$$

where $B_i = \{b_1, \ldots, b_{i-1}\}$.

The prototype tree T defines a recursive binary partition of P. Every node $t \in T$ represents a subset $S_t \subset P$. If t is a leaf then t contains exactly one prototype. Otherwise, $S_t = S_{t_l} \cup S_{t_r}$, where t_l and t_r are the left and right child of t. Every node t stores a representative $m_t \in S_t$ and the radius r_t of S_t. The *radius* is defined as follows:

$$r_t = \max_{p \in S_t} d(p, m_t). \tag{2}$$

Note that $S_t = \{m_t\}$ if t is a leaf and then $r_t = 0$.

The prototype tree T is (recursively) built as follows: the root (ρ) represents $S_\rho = P$ and m_ρ is randomly selected in P. The right child t_r of a node t has $m_{t_r} = m_t$ as representative prototype. However, for the left child t_l:

$$m_{t_l} = \text{argmax}_{p \in S_t} d(p, m_t). \tag{3}$$

Moreover, $S_{t_r} = \{p \in S_t : d(p, m_{t_r}) < d(p, m_{t_l})\}$ and $S_{t_l} = S_t - S_{t_r}$. The recursion stops at leaves which represent subsets with a single element. There exist other methods for building the tree (Duda and Hart, 1973), but they have larger time complexities.

3. The search algorithm

The search algorithm follows a branch and bound technique. The algorithm performs a guided traversal of the prototype tree pruning those branches which cannot contain the nearest neighbour. Distances are only evaluated whenever a leaf in T is reached. Initially, the nearest base prototype to the sample x is taken as candidate for nearest neighbour $p_{\min}$.

In order to reduce the number of distance calculations, a lower bound of the distance, $g : P \times P \to \mathbb{R}$ is defined:

$$g(x, y) = \max_{b \in B} \{|d(x, b) - d(b, y)|\}. \tag{4}$$

The recursive search follows a depth-first strategy combining root-left-right and root-right-left traversals depending on the values $g(x, m_{t_r})$ and $g(x, m_{t_l})$.

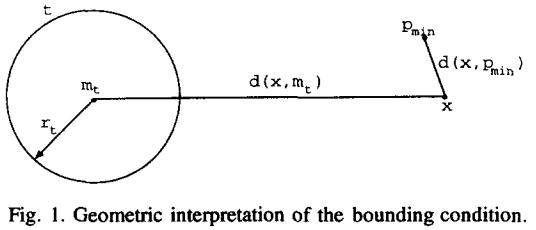

Fig. 1. Geometric interpretation of the bounding condition.

```
algorithm TLAESA
input: T (prototype tree with root ρ)
       B (subset of base prototypes)
       M[b,p] (distances between base prototypes
              and prototypes)
       x (sample)
output: p_min, d_min (nearest neighbour and distance)
begin algorithm
  d_min = ∞, g_ρ = 0
  for every b ∈ B do
      D[b] = d(x,b)
      if D[b] < d_min then
         p_min = b
         d_min = D[b]
      endif
      g_ρ = max(g_ρ, |M[b,ρ] − D[b]|)
  end for
  search(ρ, g_ρ, p_min, d_min)
end algorithm
```

Fig. 2. Algorithm TLAESA.

The child with a more promising g-value is first explored (if it is not pruned). Whenever a leaf q is reached the distance $d(x,q)$ is evaluated and compared with the smallest distance found so far. If $d(x,q) < d(x,p_{\min})$, then $p_{\min}$ is updated with q.

Functions $g(x,m_t)$ can be calculated without evaluating any distance during the search process as:

- distances of type $d(b,m_t)$ are computed during preprocessing (when base prototypes and class representatives are selected) and stored in M, and
- distances of type $d(x,b)$ do not depend on the structure of the partition and can be computed and stored as a vector D before the search in T starts.

We describe now how the pruning works. Let x be the sample, t a node in T and $p_{\min}$ the candidate for nearest neighbour. It is not difficult to prove (see Fig. 1)

$$d(x,p_{\min}) + r_t < d(x,m_t) \Rightarrow d(x,p) > d(x,p_{\min})\ \forall p \in S_t. \quad (5)$$

In such case, no prototype in S_t is closer to x than $p_{\min}$ and the x nearest neighbour cannot be in S_t. Therefore, the branch t can be pruned.

Due to the triangle inequality, $g(x,y) \leqslant d(x,y)$ for all $x, y \in P$ and the condition in Eq. (5) can be replaced with

$$d(x,p_{\min}) + r_t < g(x,m_t). \quad (6)$$

A schematic representation of the algorithm is plotted in Figs. 2 and 3.

```
algorithm search
input: t ∈ T (node)
       g_t (value for g(x,t))
output: p_min, d_min (nearest neighbour and distance)
begin algorithm
  if is_leaf(t) then
     if g_t < d_min then
        d = d(x,m_t)
        if d < d_min then
           p_min = m_t
           d_min = d
        endif
     endif
  else
     t_r = right_child(t)
     t_l = left_child(t)
     g_l = g(x,t_l)
     if g_l < g_t then
        if d_min + r_l > g_l then
           search(t_l, g_l, p_min, d_min)
        endif
        if d_min + r_r > g_r then
           search(t_r, g_t, p_min, d_min)
        endif
     else
        if d_min + r_r > g_r then
           search(t_r, g_t, p_min, d_min)
        endif
        if d_min + r_l > g_l then
           search(t_l, g_l, p_min, d_min)
        endif
     endif
  endif
end algorithm
```

Fig. 3. The search procedure.

4. Experiments

The behaviour of the algorithm was checked by means of experiments on artificial data. Prototype sets consisted of random points in the d-dimensional hy-

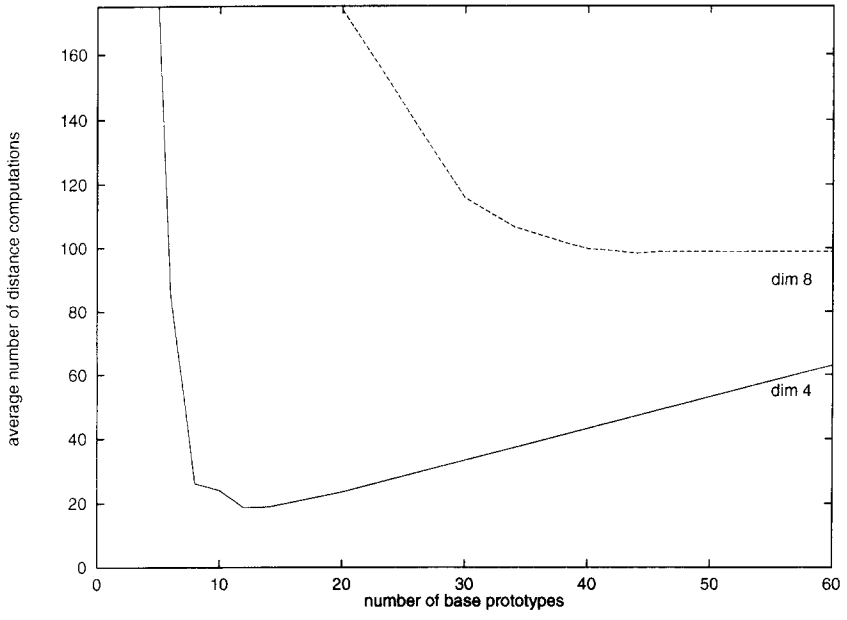

Fig. 4. Average number of distance computations in TLAESA as a function of the number n of base prototypes for different dimensions ($|P| = 2^{13}$).

percube generated according to a uniform probability distribution. All experiments were repeated with ten different prototype sets and 1000 different samples. Therefore, results were obtained as an average over 10000 experiments. Deviations were always below one per cent.

A preliminary set of experiments was performed in order to fit the only parameter in the algorithm: the number of base prototypes n. The results in Fig. 4 show that there is an optimal value of n which makes the average number of distance calculations minimal. When n is low, adding more base prototypes considerably enlarges the effect of pruning, and the total number of distance evaluations becomes reduced. However, once n is large enough, a linear increase in the number of distance calculations dominates. This is due to the fact that all distances from the sample to the base prototypes are evaluated in the algorithm (and stored in vector D). Fig. 4 also shows that the optimal n depends on the dimensionality d of the space. The experiments in (Micó, 1996) showed that this optimal value does not depend on the number of prototypes.

Another set of experiments studied how the average number of distance calculations depended on the number of prototypes. The parameter n was always set to its optimal value for the dimensionality of the experiment. As shown in Fig. 5, this dependence is rather flat. Indeed, every curve is bounded by a constant. This is checked in Fig. 6, where the average number of distance calculations in TLAESA and in the *kd*-tree algorithm (Friedman et al., 1977) are compared. This is not a general algorithm, as it needs the coordinates of the prototypes, but we will use it for comparison purposes. The number of distance calculations in the *kd*-tree algorithm is bounded by a constant which depends on the dimensionality of the space. The experiments plotted in Fig. 6 show a smaller number of distance calculations in our algorithm, indicating (compared to the *kd*-tree algorihtm) that an upper bound should also apply to our case.

Finally, comparison of TLAESA with AESA and LAESA is shown in Fig. 7. Even if AESA computes fewer distances, it can only be used with small sets (below a few thousands prototypes), due to its quadratic space complexity.

On the other hand, the pruning in TLAESA is not so effective as in LAESA but the time complexity is smaller in TLAESA. The choice between both algorithms depends on the number of prototypes and the cost of the distance computations. Small prototype

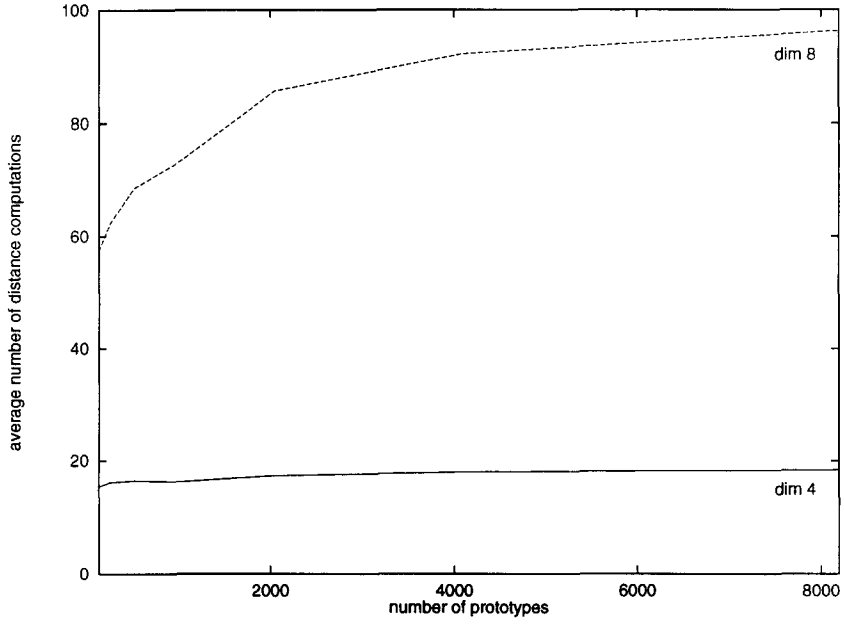

Fig. 5. Average number of distance computations in TLAESA as a function of the number of prototypes for $d = 4$ and $d = 8$.

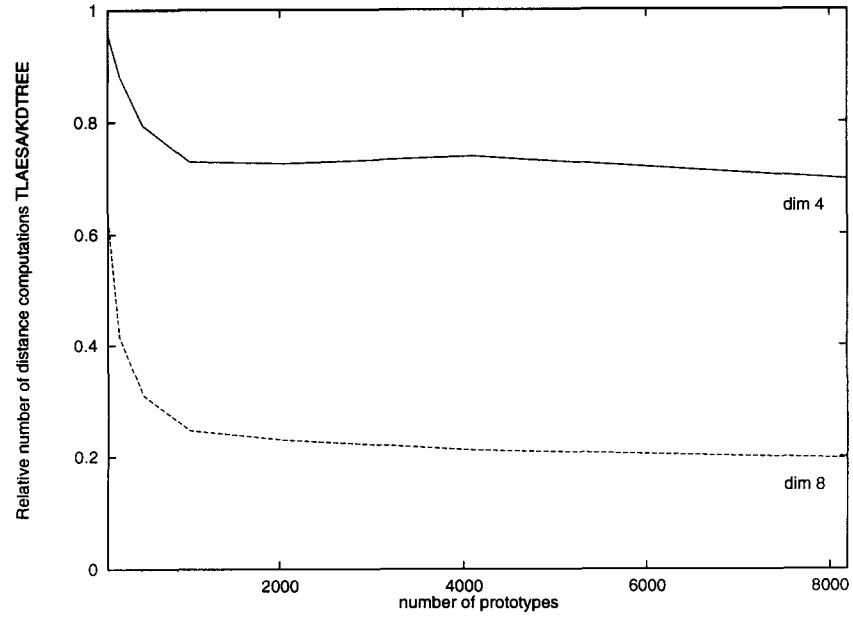

Fig. 6. Relation between the number of distances calculated by TLAESA and the number of distances calculated by the *kd*-tree algorithm.

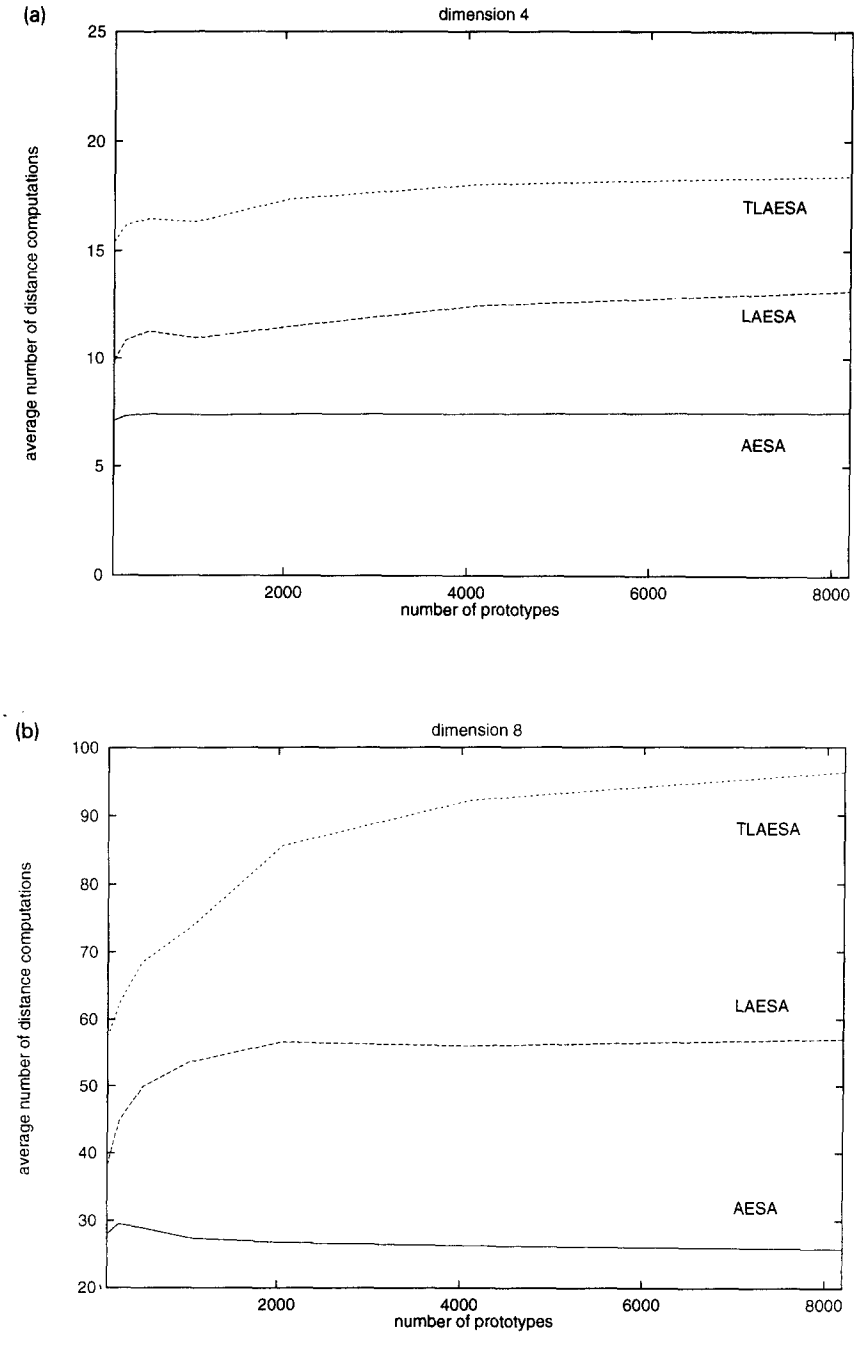

Fig. 7. Comparison of the results with TLAESA in Fig. 5 with the corresponding results using AESA and LAESA.

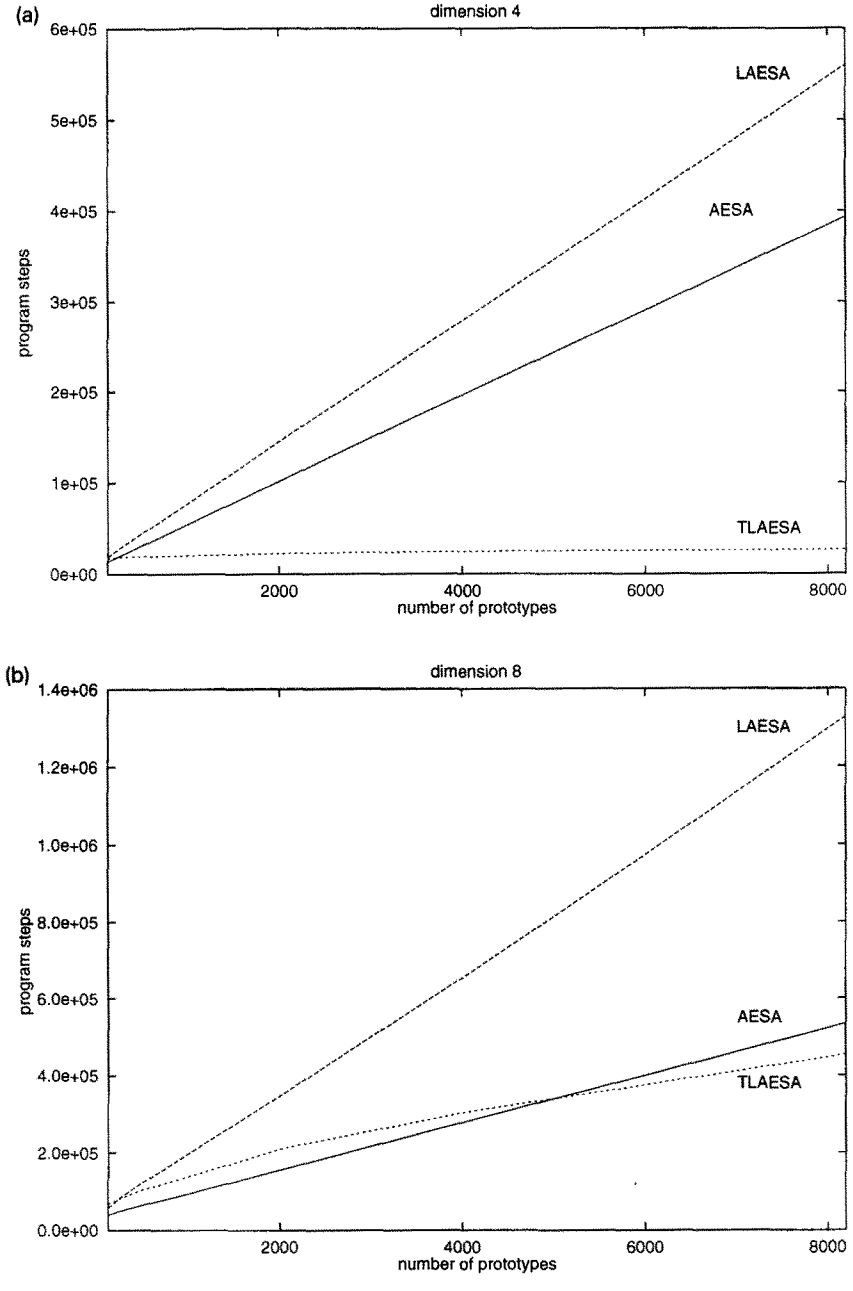

Fig. 8. Overhead for the AESA, LAESA and TLAESA, for a fixed number of steps of the distance equal to 1000.

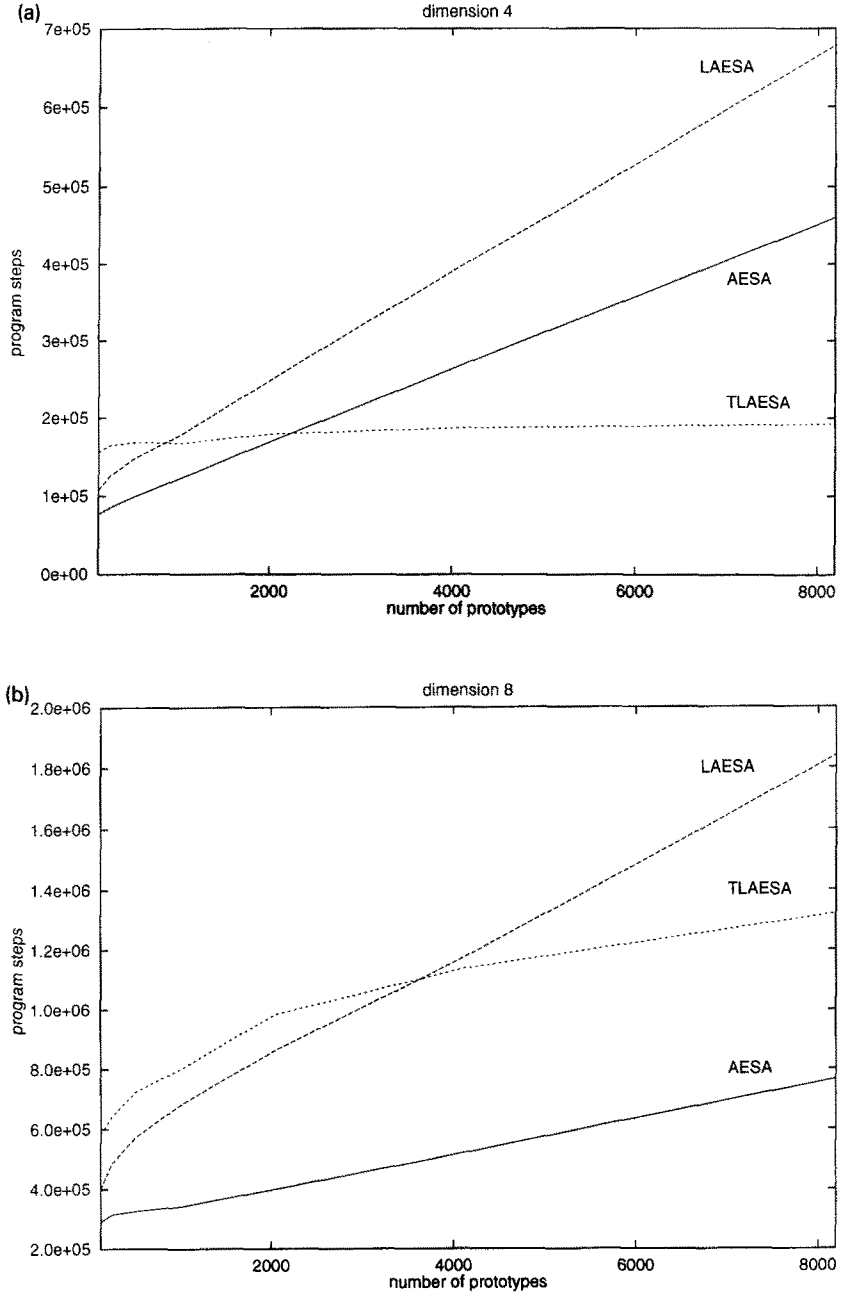

Fig. 9. Overhead for the AESA, LAESA and TLAESA algorithms in dimension 4 and 8, when the computation of the distance has a fixed value of 10000.

sets or very expensive distances will make LAESA preferable, but as the prototype set becomes larger, the TLAESA will outperform LAESA. This can be seen in Figs. 8 and 9, where the applicability range is studied. The experiments count how many steps the algorithm needed in order to find the nearest neighbour. Every single high-level sentence in the program is counted as one step every time it is executed. In addition, every time the distance function is called the number of steps is increased by a large fixed number (1000 and 10000, respectively) in order to simulate realistic experimental conditions. Figs. 8 and 9 study the influence of two characteristics of the distance functions for real data: (1) computational cost and (2) geometric properties that can be characterized for the dimensionality. In the following experiments these two properties are studied independently. The figures show that for low cost distances (as is the case for the Euclidean distance with $d = 4$), TLAESA shows a better behaviour than LAESA. However, when the distance is expensive, TLAESA becomes better at larger prototype sets. For instance, this happens for prototype sets larger than 4000 prototypes with $d = 8$.

5. Conclusions

Although AESA and LAESA are algorithms that compute very few distances in order to find nearest neighbours, their linear time complexity is a serious bottleneck when the number of prototypes becomes large. The algorithm presented here (TLAESA) makes only use of the metric properties of the distance in order to avoid the calculation of some distances. It works in sublinear time, the number of distance computations is bounded by a constant and has a linear space complexity. Although this algorithm computes, in average, more distances than AESA and LAESA, its computing time is lower when the number of prototypes increases.

References

Dasarathy, B. (1991). *Nearest Neighbour (NN) norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Silver Spring, MD.

Duda, R. and P. Hart (1973). *Pattern Classification and Scene Analysis.* Wiley, New York.

Friedman, J., J. Bentley and R.A. Finkel (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Software* 3, 209–226.

Fukunaga, K. and M. Narendra (1975). A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.* 24, 750–753.

Kalantari, I. and G. McDonald (1983). A data structure and an algorithm for the nearest point problem. *IEEE Trans. Software Engrg.* 9, 631–634.

Micó, L., J. Oncina and E. Vidal (1992). An algorithm for finding nearest neighbours in constant average time with a linear space complexity. *Proc. 11th ICPR*, The Hague, Vol. II, 557–560.

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

Micó, L. (1996). Nearest neighbour search algorithms in metric spaces (in Spanish). Ph.D. dissertation, Universidad Politécnica de Valencia, 1996.

Ramasubramanian, V. and K. Paliwal (1990). An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation. In: L. Torres et al., Eds., *Signal Processing V: Theories and Applications.* Proc. EUSIPCO-90. North-Holland, Amsterdam, 1323–1326.

Shasha, D. and T. Wang (1990). New techniques for best-match retrieval. *ACM Trans. Inform. Syst.* 8, 140–158.

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

Vidal, E. (1994). New formulation and improvements of the nearest-neigbour Approximating and Eliminating Search Algorithm (AESA). *Pattern Recognition Lett.* 15, 1–7.