Comparison of AESA and LAESA search algorithms using string and tree edit distances [1,2]

Juan Ramón Rico-Juan [a] Luisa Micó [b]

[a]*Universidad de Alicante, Departamento de Lenguajes y Sistemas Informáticos, E-03080 SPAIN*

[b]*Universidad de Alicante, Departamento de Lenguajes y Sistemas Informáticos, E-03080 SPAIN*

Abstract

Although the success rate of handwritten character recognition using a nearest neighbour technique together with edit distance is satisfactory, the exhaustive search is expensive. Some fast methods as `AESA` and `LAESA` have been proposed to find nearest neighbours in metric spaces. The average number of distances computed by these algorithms is very low and does not depend on the number of prototypes in the training set. In this paper, we compare the behaviour of these algorithms when string and tree edit distances are used.

Key words:
Metric Spaces; Pattern Recognition; Nearest Neighbour; Handwritten Character Recognition; Edit Distance

1 Introduction

Nearest neighbour search (NNS) is one of the best known and simplest techniques in Statistical Pattern Recognition. Given a set *P* of prototypes (*training set*) whose classification is known, the nearest neighbour technique classifies a *test sample x* in the class containing the prototype whose distance to *x* is minimal. For this purpose, some fast algorithms as `AESA` (Vidal (1994)) and `LAESA` (Micó et al. (1994)) have been proposed. The main features of these algorithms are: (a) they work in metric spaces (no representation of the data as a vector is necessary) and (b) experiments suggest that in these algorithms the number of distance computations

[1] The authors wish to thank Eva Gómez and Rafael C. Carrasco for useful comments
[2] Work partially supported by the spanish CICYT TIC2000-1599-C02 and TIC2000-1703-CO3-02

is nearly constant with the size of the training set. The first property allows one to use these methods in problems such as a chain representation for handwritten characters. The second property makes these methods faster when the cost of the distance computation is very expensive, as the Levenstein edit distance (Duda and Hart (1973)).

The `LAESA` algorithm (Micó et al. (1994)) is an evolution of the `AESA` algorithm (Vidal (1994)) and shares with it the above two properties. The major bottleneck of the `AESA` is its quadratic preprocessing time and memory requirements that limits its use for large data sets. In all the applications where both algorithms has been used, the `AESA` algorithm computes about one third of the distances computed by `LAESA`. This result does not necessarily mean that the `AESA` is faster than the `LAESA`. Indeed, depending on the type of the distance, in this work we will show that one method can be faster than the other. On the other hand, when the string edit distance was used in a real task as handwriting recognition (Gómez et al. (1995) and Micó and Oncina (1998)), speech recognition (Vidal et al. (1988)) or human banded chromosomes recognition (Juan and Vidal (2000)), a "looseness" on the triangle inequality allowed one to speed up the classification due to the algorithms computes less number of distances. In this work, we show experimentally that this is not needed when tree-edit-distance is used.

2 Feature extraction

In this section, we describe two different representations of handwritten characters. The first representation uses labelled trees as a basic structure and the second uses labelled strings.

2.1 Tree code

The Nagendraprasad-Wang-Gupta thinning algorithm (as modified in Carrasco and Forcada (1995)) was applied to eliminate redundant information in all binary images in the training set. The result image is transformed into a tree representation. The algorithm consists of the following steps:

(1) The first up and left pixel is marked and assigned the tree root with a special label "0".
(2) Every tree node has so many descendents as unmarked neighbours has the selected and marked pixel (clockwise).
(3) Each branch is extended following the neighbour until one of the following criteria is true:
    (a) the branch has the maximum fixed parameter size (*window*=6);

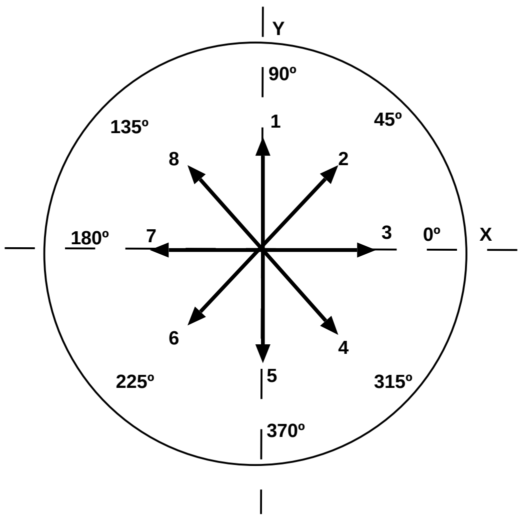


Figure 1. 2D labelled regions

(b) the pixel has no unmarked neighbours (terminal pixel);
(c) the pixel has more than one unmarked neighbour (intersection pixel).

(4) A new node is assigned to every end of branch pixel obtained after step 3. The starting pixel shall be placed at the origin of the axes. The label is assigned to the node depending on the final pixel relative position to the starting one. The 2D space is divided in 8 regions (figure 1): the north is labelled with **“1”** and clockwise, the rest is labelled. These labels are similar to codes used by Freeman (1961) except that the special code “0” is assigned to the root label and the rest of directions range between “1” and “8”.

(5) For each node with unmarked neighbours, go to step 2.

An example showing this feature extraction is presented in figure 2.

2.2 String code

An algorithm to extract the external contour of the characters is used to obtain the patterns from the images of the characters. It begins by searching the first black pixel in a left-to-right scan starting from the top. From this pixel going to the right, the border of the character is followed until this first pixel is reached again. During this route the algorithm builds a string with the directions that it follows to find the next pixel of the border. There are eight neighbouring pixels that can be found after a given pixel (figure 3), therefore, only eight symbols can appear in this chain-code. The numbers from 0 to 7 were used.

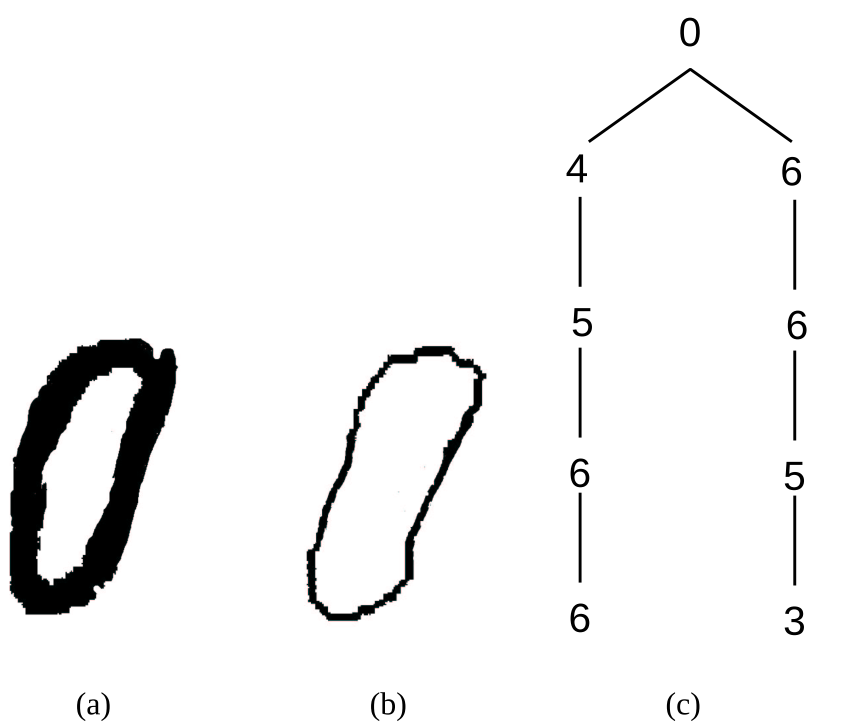
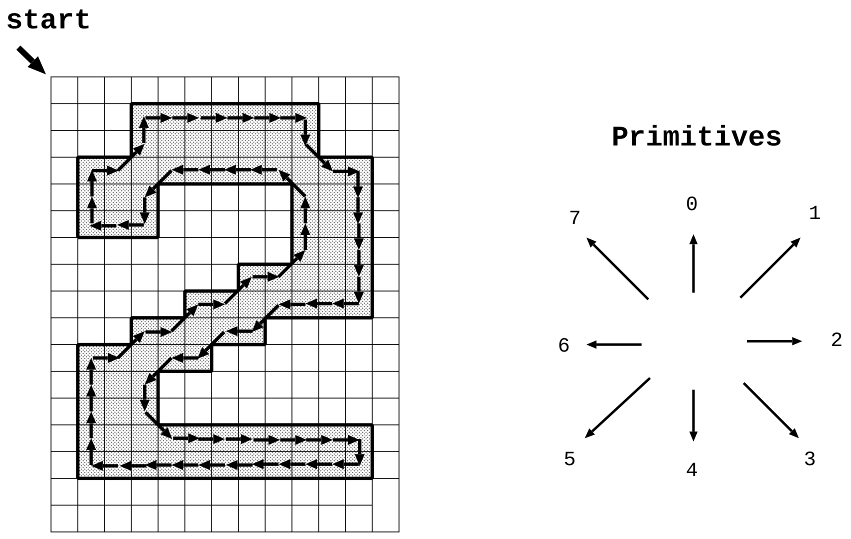


Figure 2. A digit example: (a) original image; (b) thinned image; (c) final labelled tree



```
"2"=2222224324444466656565432222222246
6666666660000212121210076666546600210
```

Figure 3. Example of string coded character.

3 Dissimilarity functions

As we have two structures we have two dissimilarity functions: the tree-edit-distance and the string-edit-distance.

$$w_{ij} = \begin{bmatrix} 0&1&2&3&4&3&2&1 \\ 1&0&1&2&3&4&3&2 \\ 2&1&0&1&2&3&4&3 \\ 3&2&1&0&1&2&3&4 \\ 4&3&2&1&0&1&2&3 \\ 3&4&3&2&1&0&1&2 \\ 2&3&4&3&2&1&0&1 \\ 1&2&3&4&3&2&1&0 \end{bmatrix}$$

Figure 4. Weight matrix for the substitution operation

3.1 The tree-edit-distance

A general tree-edit-distance, d_T, is described in Zhang and Shasha (1989). The distance between two ordered trees is considered to be the weighted number of edit operations (insertion, deletion and substitution) to transform one tree into another.

Substituting a node n means changing the label in n. When deleting a node n, the children of n become the children of the parent of n and n is removed. Insertion is complementary of deletion. This means that inserting n as the child of n' will make n the parent of a consecutive subsequence of the current children of n'.

A dynamic programming algorithm is implemented to compute the distance between two trees, T_1 and T_2 whose complexity is in space $O\left(|T_1| \times |T_2|\right)$ and time $O\left(|T_1| \times |T_2| \times \min\left(\text{depth}\left(T_1\right),\ \text{leaves}\left(T_1\right)\right) \times \min\left(\text{depth}\left(T_2\right),\ \text{leaves}\left(T_2\right)\right)\right)$.

Each basic operation has an associated weight. We use the weights proposed in Rico-Juan (1999). Substitution weights w_{ij} are $\min(|i-j|, 8-|i-j|)$. This equation assigns smaller weights to closer directions and larger weights to distant directions as shown in figure 4. Both insertion and deletion have a weight $w_I = w_D = 2$. The effect of choosing a different set of weights is relatively small with respect to accuracy, provided that $w_I + w_D \geq w_{ij}$.

This distance is finally normalised with the sum of the number of nodes in each tree:

$$d'_T(T_1, T_2) = \frac{d_T(T_1, T_2)}{|T_1| + |T_2|}$$

3.2 The string-edit-distance

The edit distance, d_S, is defined as the minimum-cost set of transformations to turn a string into the other. The basic transformations are deletion, insertion and substitution of a single symbol in the string. This distance can be defined recursively as follows:

$$d_S(\lambda,\lambda) = 0$$

$$d_S(xa,yb) = min \begin{cases} d_S(x,yb) + w_D \\ d_S(xa,y) + w_I \\ d_S(x,y) + w_{ab} & \text{if } a \neq b \\ d_S(x,y) & \text{if } a = b \end{cases}$$

where w_D, w_I and w_{ab} are, respectively, the cost of a deletion, an insertion and a substitution, and λ is the empty string. The cost values are equal as those used in tree-edit-distance, so $w_D = w_I = 2$ and w_{ij} is shown in figure 4.

This distance can be computed in time in $O(|x|,|y|)$ using a standard dynamic-programming technique (Masek and Paterson (1980) and Wagner and Fischer (1974)).

As in the tree-edit-distance case, this final measure is normalised by the sum of length of the strings:

$$d'_S(x,y) = \frac{d_S(x,y)}{|x|+|y|}$$

4 The algorithms

The LAESA algorithm requires a preprocessing procedure that selects a subset of *base prototypes* B from the *training set* P, and computes and stores the distances between the prototypes of P and B. This preprocessing is the main difference in relation to the AESA algorithm, since AESA requires the computation and storage of all the distances between all the prototypes belonging to P. In the figures 5 and 6 are detailed the AESA and LAESA algorithms respectively.

The search procedure in these algorithms uses a *branch and bound* technique for finding the nearest prototype to a sample x (*test sample*). The two main steps of these algorithms are approximation and elimination. Instead of computing the distance between the prototypes belonging to P and the sample x to search the nearest neighbour (exhaustive search), a lower bound is used to avoid some distance computations. This lower bound uses the information obtained in the preprocessing step and in the previous iterations of the search procedure. For every $p \in P$, a lower bound $g_x(p)$ of the distance from x to p can be easily derived from the triangle

```
Input: $P$ (training set),
       $x$ (sample)
Output: $min \in P$, $D_{\min} \in \mathbb{R}$
 (1) Initialization.
     $D_{\min} = \infty$
     for every $p \in P$, $g(p) = 0$
     $s$ = arbitrary_element($P$)
 (2) Distance computation, $d(x,s)$, $P = P - \{s\}$
 (3) Update $min$ and $D_{\min}$
 (4) For every $p \in P$ do
       • Update function $g(p)$
       • Elimination. If $g(p) \geq D_{\min}$, $p$ is eliminated from $P$.
       • Approximation. Select as new candidate $s$. The prototype with the mini-
         mum value of $g$. Return to step 2 if $P \neq \emptyset$.
```

Figure 5. Scheme of the AESA algorithm.

```
Input: $P$ (training set),
       $B \subseteq P$ (base prototypes),
       $x$ (sample)
Output: $min \in P$, $D_{\min} \in \mathbb{R}$
 (1) Initialization.
     $D_{\min} = \infty$
     for every $p \in P$, $g(p) = 0$
     $s$ = arbitrary_element($B$)
 (2) Distance computation, $d(x,s)$, $P = P - \{s\}$
 (3) Update $min$ and $D_{\min}$
 (4) For every $p \in P$ do
       • Update function $g(p)$ // only when $s \in B$
       • Elimination. If $p \notin B$ and $g(p) \geq D_{\min}$, $p$ is eliminated from $P$.
       • Approximation. Select as new candidate $s$. The prototype with the mini-
         mum value of $g$ (where $s \in B$ if $B \neq \emptyset$). Return to step 2 if $P \neq \emptyset$.
```

Figure 6. Scheme of the LAESA algorithm.

inequality:

$$g_x(p) = \max_{b \in U} \{|d(x,b) - d(b,p)|\} \tag{1}$$

where U is the subset of prototypes whose distance to the test sample has been previously computed. For the AESA algorithm, all prototypes in P are allowed to be in U (provided that their distance to x has been computed). However, in the LAESA algorithm, only base prototypes are added to the set U.

Both methods can be implemented as an iterative procedure that ends when the list of prototypes is empty. At each iteration, a new candidate is selected and those prototypes that cannot be closer to the sample than the actual candidate are eliminated (using the triangle inequality). In the LAESA algorithm the new candidate is selected among base prototypes.

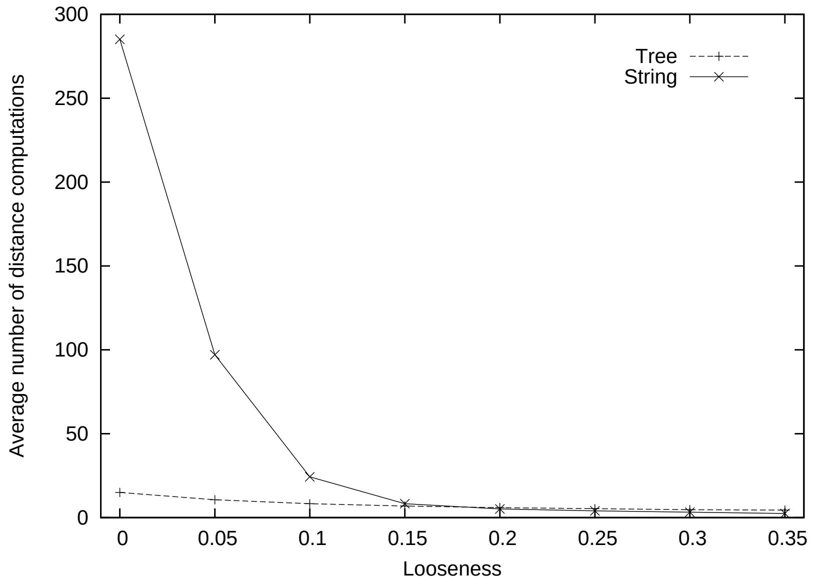


Figure 7. Average number of distance computations as a function of the looseness for `AESA` algorithm using 4000 prototypes as training set. The standard deviation for all estimations was below 3.5%.

In the approximation step, the prototype with a minimum value of the lower bound function is selected. Details of the algorithm can be found in Vidal (1994) and Micó et al. (1994).

Some methods for selecting base prototypes from the training set *P* are proposed in previous works as Micó et al. (1994). In this paper, the *maximum minimum distances* method (*MMD*) is used. This is the recommend method in Micó (1996) and uses an accumulator array that stores the minimum distance to the preselected base prototypes. The prototype with a maximum value in the array in each iteration is selected as a new base prototype. In this paper, `LAESA` was implemented using the elimination condition where the base prototypes never are pruned. This criterion means that all the distances to the test sample from the base prototypes are computed before selecting non-base prototypes in the approximation step.

5 Experiments

In order to explore the application of this algorithms to the handwritten digit recognition task, we apply these classifiers to the NIST SPECIAL DATABASE 3 of National Institute of Standards and Technology also used in other works as Rico-Juan and Calera-Rubio (2002), Gómez et al. (1995), Micó and Oncina (1998) and López and Piñaga (2000).

The increasing-size training samples for the experiments were built by taking 50 writers and selecting the samples of digits randomly. The figures show the results

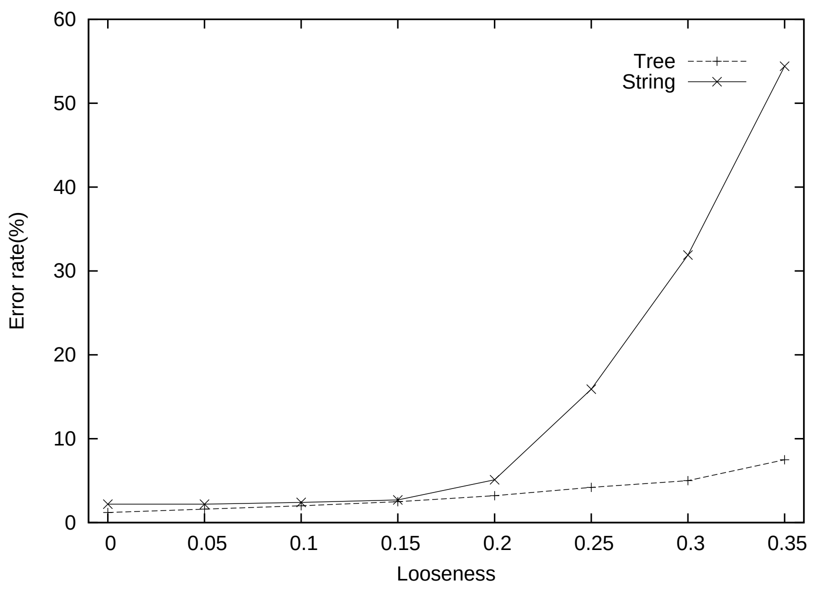


Figure 8. Error rate as a function of the looseness for `AESA` algorithm using 4000 prototypes as training set. The standard deviation for all estimations was below 3.5%.

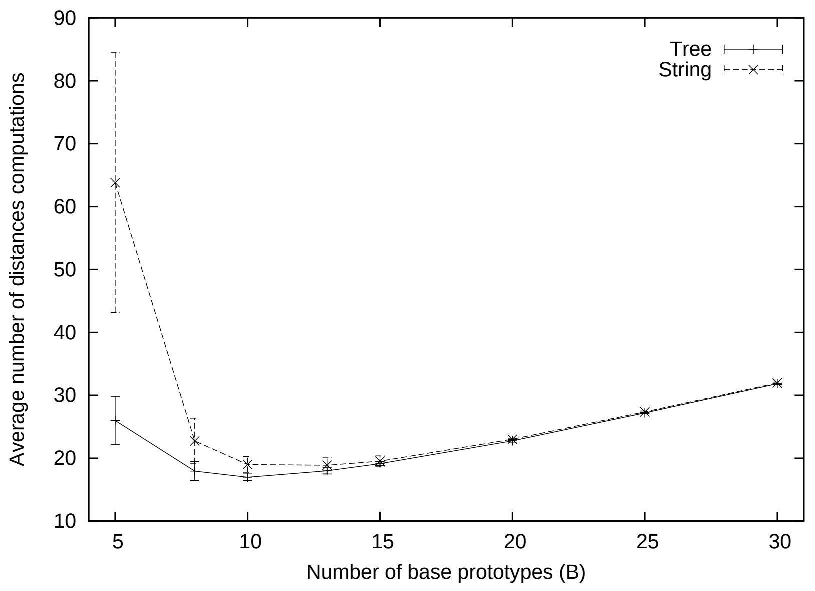


Figure 9. Average number of distance computations for `LAESA` algorithm with $H = 0.15$ as a function of the number of base prototypes, for a training set of 4000 prototypes.

averaged for all combinations.

The application of the `AESA` and `LAESA` algorithms in previous works as Vidal and Lloret (1988), Vidal et al. (1988) and Micó and Oncina (1998) shows that a “looseness”, H, in the triangle inequality reduces the number of distance computations.

Given a representation space E, the *looseness* is defined for each $x,y,z \in E$ as $h(x,y,z) = d(x,y) + d(y,z) - d(x,z)$. If a histogram of the distributiion of $h(x,y,z)$ is

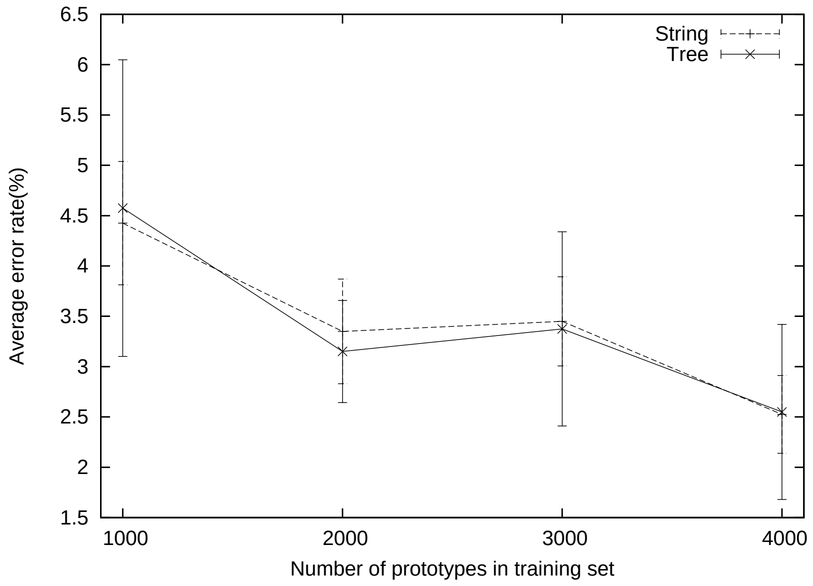


Figure 10. Average error rate obtained with different sizes of handwritten digits in the NIST database with $H = 0.15$.

computed, this histogram can be used to estimate of the probability that the triangle inequality is satisfied with a *looseness* smaller than H. (Vidal et al. (1985)).

A prototype is eliminated in both algorithms if $g_x(p) \geq D_{min} - H$, where D_{min} is the distance between the sample and the nearest neighbour found at this moment.

We have checked that AESA and LAESA algorithms using the tree-edit-distance and the “looseness” is not needed to obtain very low computing time.

A first experiment using AESA was made to select H (using 4000 prototypes as a training set). The experiments in Micó (1996) showed that this optimal number does not depend on the number of prototypes in the training set. We can see in figure 7 that these algorithms compute a few number of distances, even when the looseness is not used with tree-edit-distance. Another interesting result obtained when looseness is used for tree-edit-distance (to compare with the strings case) is that the classification error rate is only slightly increased in contrast to the large increase in the strings case.

In view of figures 7 and 8, we have decided to select $H = 0.15$ as value to use for pruning. This value reduces the number of computed distances but does not increases significantly the classification error rate. If $H = 0$ the number of computed distances doubles in the trees case but, in the strings case, this number is 30 times bigger.

In the following experiment, the optimal size of the set of base prototypes, B, for the LAESA is calculated. As shown in figure 9, the optimal size of B that minimises the number of computed distances per sample is 10 and 13 for trees and strings,

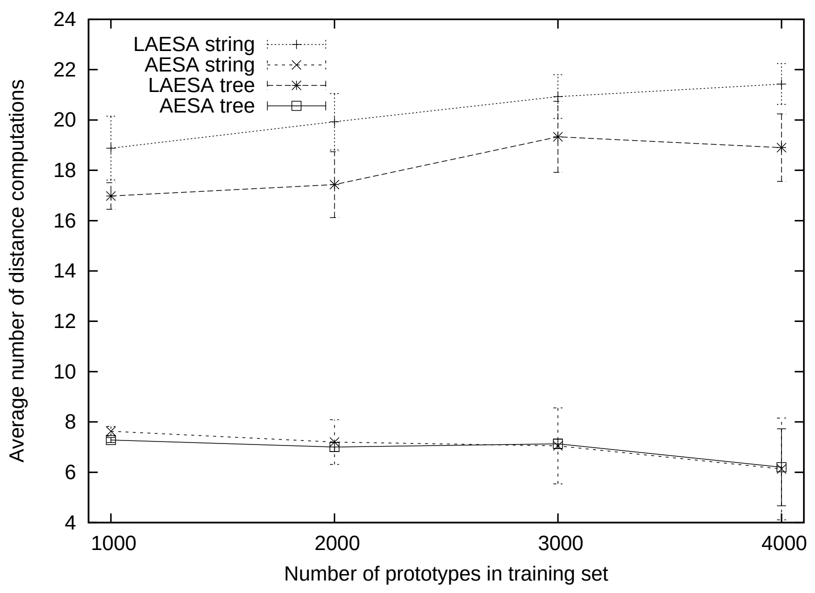


Figure 11. Average number of distances computed by the algorithms as a function of the numbers of prototypes in training set.

respectively. These sizes are fixed for the rest of the `LAESA` experiments.

The accuracy of classification, as well as, the number of computed distances per sample when $H = 0.15$ are similar when using trees or strings (figure 10).

The number of distance computations needed to classify a sample is very important when the cost of the distance is high. The figure 11 shows that this number computed by `AESA` (`LAESA`) tends to constants 6 (19), in the tree case; 6 (21), in the string case. So, this fact suggests that `AESA` is faster than `LAESA`. However, we will see that this is not necessarily true.

Note that the time to classify a sample depends both on the algorithm and on the dissimilarity function. As the figure 12 shows, `LAESA` is faster than `AESA` when the tree-edit-distance is used. Both algorithms are faster when the tree-edit-distance is used instead of the string-distance.

In order to study this behaviour, we have considered the time used to classify a sample by the algorithms as a function of the number $n = |P|$ of prototypes,

$$T(n) = T_o(n) + T_1(n) \qquad (2)$$

where $T_o(n)$ is the time consumed if the cost of the dissimilarity function was null and $T_1(n)$ is the time used by dissimilarity function. In other words, $T_1(n) = N(n) \times t$, where $N(n)$ is the number of computed distances and t the average time used to compute the dissimilarity function.

An approximate way to obtain $T_o(n)$ is precompute all the distances between the samples in training set and test set and store the results in a matrix (see figure 13).

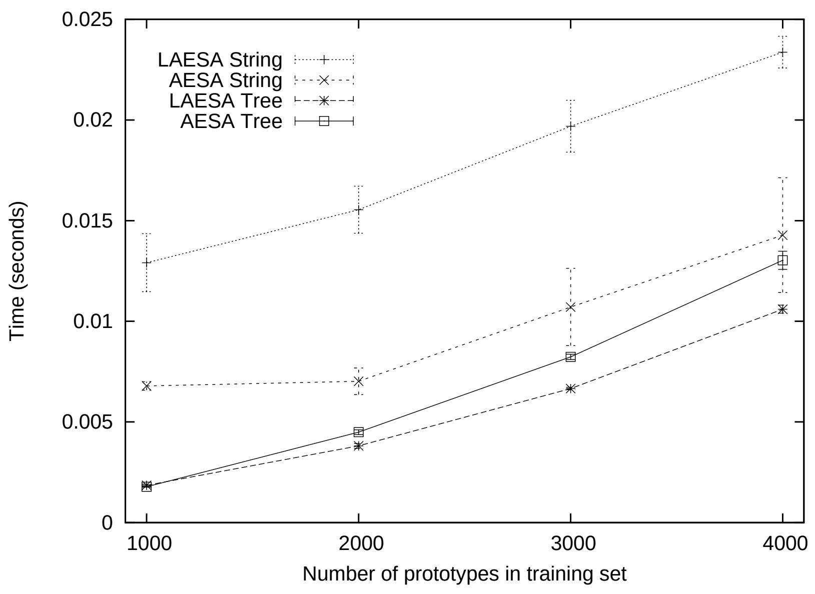


Figure 12. Average time needed to classify a sample as a function of the number of prototypes in training set.

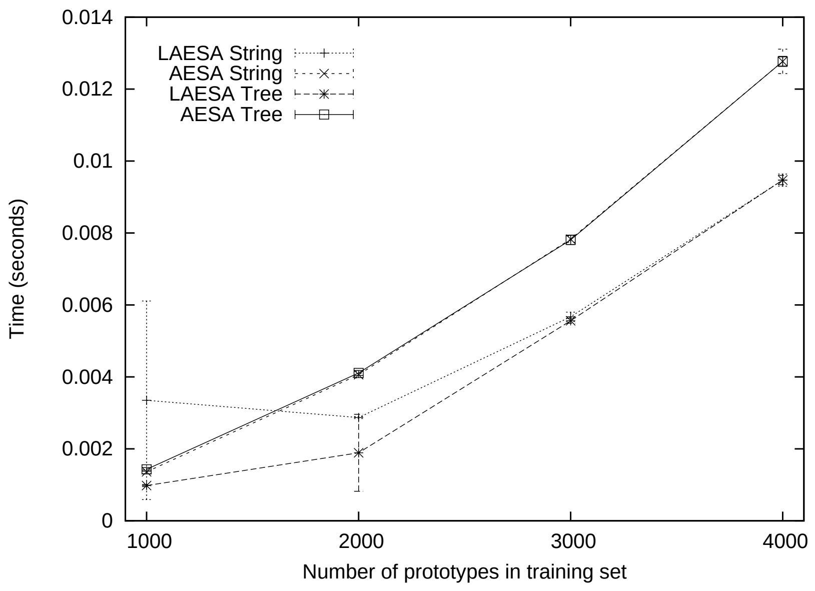


Figure 13. An approximation to compute $T_o(n)$ as a function of the number of prototypes in training set.

The `LAESA` is faster than the `AESA` in most cases. This reflects that the bounds update in `AESA` is more expensive than in `LAESA`.

If we know $T_o(n)$ and $N(n)$, we can estimate the value of t where `LAESA` becomes faster than `AESA`

$$
\begin{aligned}
T^{\mathrm{L}}(n) &= T_o^{\mathrm{L}}(n) + N^{\mathrm{L}}(n)\widehat{t}(n) \\
T^{\mathrm{A}}(n) &= T_o^{\mathrm{A}}(n) + N^{\mathrm{A}}(n)\widehat{t}(n)
\end{aligned}
\tag{3}
$$

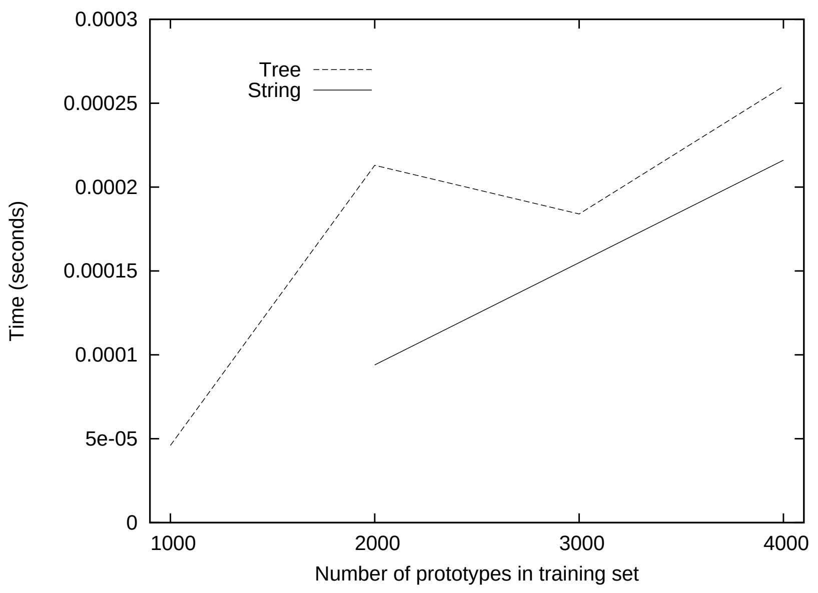


Figure 14. Representation of $\widehat{t}(n)$ when $T^{\mathrm{L}}(n) = T^{\mathrm{A}}(n)$ as a function of the numbers of prototypes in training set.

where the superscripts L and A denote `LAESA` and `AESA`, respectively. When $T^{\mathrm{L}}(n) = T^{\mathrm{A}}(n)$ we obtain the crossing point $\widehat{t}(n)$

$$\widehat{t}(n) = \frac{T_o^{\mathrm{A}}(n) - T_o^{\mathrm{L}}(n)}{N_o^{\mathrm{L}}(n) - N_o^{\mathrm{A}}(n)} \tag{4}$$

represented in figure 14. If the t is between 0 and $\widehat{t}(n)$ the `LAESA` is faster than the `AESA`, otherwise `AESA` is faster. Note that in the strings case if the size of training set is below than 2000, `AESA` is always the faster. In contrast, in the trees case the `LAESA` is always faster than `AESA`.

6 Conclusions

In this paper we have studied the situations where `LAESA` is faster and cheaper (in terms of memory cost) than `AESA`. In previous works as Micó et al. (1994) and Micó and Oncina (1998), it seemed that `AESA` was the best algorithm, because it computes less distances. However, the cost of the dissimilarity function is very important to the final result.

Our experiments show that the tree-edit-distance is a suitable choice in order to use `AESA` and `LAESA` algorithms in a handwritten recognition task. Although the error rate is similar using strings or trees, the average time needed to classify a sample becomes smaller when the tree-edit-distance is used.

This paper also shows that the *looseness* can be avoided when the tree-edit-distance is used, in contrast with previous works as Juan and Vidal (2000) and Micó and Oncina (1998) where the looseness was needed. However, further work must be done in order to understand the *looseness* effect in the string and the tree case.

References

Carrasco, R. C. and Forcada, M. L. (1995). A note on the Nagendraprasad-Wang-Gupta thinning algorithm. *Pattern Recognition Letters*, 16:539–541.

Duda, R. O. and Hart, P. E. (1973). *Pattern Recognition and Scene Analysis*. John Wiley and Sons, New York.

Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computer*, 10:260–268.

Gómez, E., Micó, L., and Oncina, J. (1995). Testing the linear approximating eliminating search algorithm in handwritten character recognition tasks. In *Actas del VI Simposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, pages 212–217, Córdoba (Spain).

Juan, A. and Vidal, E. (2000). On the use of edit distances and an efficient k-NN search technique (k-AESA) for fast and accurate string classification. In *Proc. of the 15th Int. Conf. on Pattern Recognition (ICPR 2000)*, volume 2, pages 680–683. Spain.

López, D. and Piñaga, I. (2000). Syntactic pattern recognition by error correcting analysis on tree automata. In Ferri, F. J., Iñesta, J. M., Amin, A., and Pudil, P., editors, *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 133–142, Berlin. Springer-Verlag.

Masek, W. J. and Paterson, M. S. (1980). A faster algorithm for computing string edit distances. *J. of Computer and System Sciences*, 20:18–31.

Micó, L. (1996). *Algoritmos de búsqueda de vecinos más próximos en espacios métricos*. PhD thesis, Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación.

Micó, L. and Oncina, J. (1998). Comparison of fast nearest neighbour classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19:351–356.

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

Rico-Juan, J. R. (1999). Off-line cursive handwritten word recognition based on tree extraction and an optimized classification distance. In Torres, M. I. and Sanfeliu, A., editors, *Pattern Recognition and Image Analysis: Proceedings of the VII Symposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, volume 3, pages 15–16, Bilbao (Spain).

Rico-Juan, J. R. and Calera-Rubio, J. (2002). Evaluation of handwritten character recognizers using tree-edit-distance and fast nearest neighbour search. In Iñesta, J. M. and Micó, L., editors, *Pattern Recognition in Information Systems*, pages 326–335, Alicante (Spain). ICEIS PRESS.

Vidal, E. (1994). New formulation and improvements of the Nearest-Neighbour approximating and eliminating search algorithm(AESA). *Pattern Recognition Letters*, 15(1):1–7.

Vidal, E., Casacuberta, F., and Rulot, H. (1985). Is the DTW distance really a metric? an algorithm reducing the number of dtw comparisons in isolated words. *Speech Communication*, 4:333–344.

Vidal, E. and Lloret, M. J. (1988). Fast speaker independent DTW recognition of isolated words using a metric-space search algorithm (AESA). *Speech Communication*, 7:417–422.

Vidal, E., Rulot, H., Casacuberta, F., and Benedí, J. (1988). On the use of a metric-space search algorithm (AESA) for fast DTW-bassed recognition of isolated words. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36:651–660.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *J. ACM*, 21:168–173.

Zhang, K. and Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262.