Speeding up the cyclic edit distance using LAESA with early abandon

Vicente Palazón-González[a,*], Andrés Marzal[a]

[a] *Universitat Jaume I, Dept. Llenguatges i Sistemes Informàtics and Institute of New Imaging Technologies, Castellón de la Plana, Spain*

Abstract

The cyclic edit distance between two strings is the minimum edit distance between one of this strings and every possible cyclic shift of the other. This can be useful, for example, in image analysis where strings describe the contour of shapes or in computational biology for classifying circular permuted proteins or circular DNA/RNA molecules.

The cyclic edit distance can be computed in $O(mn \log m)$ time, however, in real recognition tasks this is a high computational cost because of the size of databases. A method to reduce the number of comparisons and avoid an exhaustive search is convenient. In this work, we present a new algorithm based on a modification of LAESA (Linear Approximating and Eliminating Search Algorithm) for applying pruning in the computation of distances. It is an efficient procedure for classification and retrieval of cyclic strings. Experimental results show that our proposal considerably outperforms LAESA.

Keywords:
Cyclic strings, cyclic edit distance, AESA, LAESA, shape recognition, circular permuted proteins, circular DNA/RNA.

1. Introduction

Comparison of strings is an important problem in pattern recognition employed in applications that range from speech and image analysis to molecular biology [20]. A widely used measure between two strings is the edit distance (ED). This distance is defined as the cost of the best sequence of edit operations for transforming one string into the other [24].

There are situations in which the objects are better described using cyclic strings. They are strings or sequences where the last element is followed by the first element, that is, there is no beginning or end [17, 18]. For example, contours of shapes can be represented by cyclic chain-codes [1, 26]. We can also find them in computational biology. Inherited information can be stored on circular DNA or RNA molecules [15, 13, 5] and there are also cyclic sequences in proteins [14, 25].

If we want to measure distances between cyclic strings, we can use the cyclic edit distance (CED), which is defined as the cost of the best sequence of edit operations for transforming a string into any cyclic shift of the other. This can be computed in $O(m^2 n)$ time (where m and n are the sizes of the strings). In [8], Maes introduced an algorithm that reduces the time cost to $O(mn \log m)$. In [10, 19], the authors improved the Maes' algorithm using a branch and bound procedure achieving a significant speeding up.

In classification or retrieval systems running-time is a very important issue. For this reason, an exhaustive search of databases is not appropriate. We need to avoid it by means of indexing methods. AESA (Approximating and Eliminating Search Algorithm) [22, 23] is characterised by a drastic reduction of the number of distances computed. It is then specially interesting when the distance has a high cost. Although AESA offers a good performance, it has a quadratic space complexity with respect to the size of the database, then, it is not suitable when our database begins to grow. In that case, linear AESA (LAESA) [11] offers a better alternative because its space complexity is linear. However, its performance is lower than AESA.

In this paper, we present a modification of LAESA for including pruning in the computation of the distances (LAESA with early abandon) that improves the performance of LAESA and AESA (in some cases) in the recognition of cyclic strings.

The remainder of this document is organized as follows. In Sections 2 and 3, CED, AESA and LAESA are revisited. The modification of LAESA, LAESA with early abandon (LAESAEA) applied to the CED is presented in Section 4. In Section 5, experimental results in classification and retrieval tasks compare the different methods. Finally, conclusions and future work are commented in Section 6.

2. Cyclic edit distance

Let Σ be a set of symbols and let Σ^* be the set of all finite strings over Σ. Let $x = x_1 x_2 \dots x_m$ and $y = y_1 y_2 \dots y_n$ be two strings in Σ^*. Let u, v, z be symbols in Σ and let λ be the empty string. An edit sequence is a sequence $e = e_1 e_2 \dots e_q$ where e_i, for $1 \le i \le q$, is one of the following edit operations: deletion $(u \to \lambda)$, insertion $(\lambda \to v)$, substitution $(u \to v)$, and matching $(u \to u)$. Let $\gamma(e) = \Sigma_{1 \le i \le q} \gamma(e_i)$ be the cost of the edit sequence e, being $\gamma(e_i)$ the cost of the edit operation e_i (satisfying $\gamma(u \to v) + \gamma(v \to z) \ge \gamma(u \to z)$). The edit distance,

*Corresponding author: Tel.: +34-964-72-8335; fax: +34-964-72-8435;
Email address: palazon@lsi.uji.es (Vicente Palazón-González)

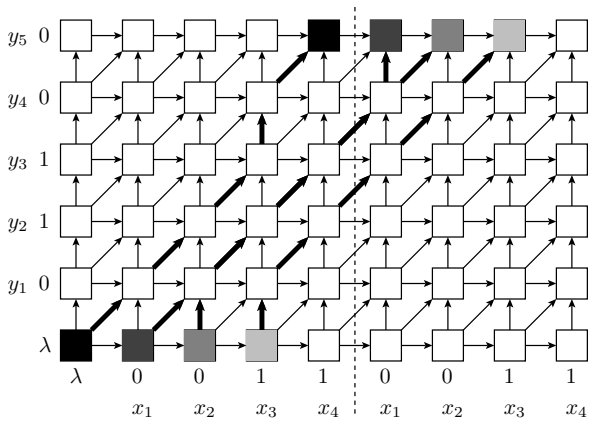

Figure 1: Extended edit graph for strings $x = 0011$ and $y = 01100$. The cyclic edit distance is computed using unit cost for insertions, deletions and substitutions and zero cost for matchings. The optimal path for the cyclic edit distance is the optimal path among those ones starting and ending at nodes with the same colour.

$ED(x, y)$, is the cost of the optimal edit sequence (that is, the one with minimum cost, $\gamma(e)$) that transforms x into y.

The edit distance (ED) can be computed in $O(mn)$ time using a dynamic programming procedure [24]. This procedure finds the optimal path in an edit graph. This graph is connected by horizontal, vertical and diagonal arcs. These arcs represent deletions, insertions and substitutions or matchings, respectively. Each path from $(0, 0)$ to (m, n) corresponds to an edit sequence.

Let $\sigma(x_1x_2 \dots x_m) = x_2 \dots x_m x_1$ denote a cyclic shift. Let σ^k be k cyclic shifts. The equivalence class $[x] = \{\sigma^k(x) : 0 \le k < m\}$ is a cyclic string. We can obtain the value of $CED([x], [y])$ by computing m edit distances in $O(m^2n)$ time:

$$CED([x], [y]) = CED([x], y) = \min_{0 \le k < m} ED(\sigma^k(x), y). \quad (1)$$

Maes' algorithm [8] uses a divide and conquer recursive procedure that computes these m edit distances (that is, the CED) in a more efficient way. This procedure uses an extended edit graph, an edit graph in which the string x is twice (see Figure 1). Let $T(k)$ be an optimal path between nodes $(k, 0)$ and $(k + m, n)$. When we compute $ED(\sigma^k(x), y)$, it is possible to use the non-crossing property of these paths. Each recursive call generates up to two recursive calls (see Figure 2). The total computation time is then $O(mn \log m)$.

Maes' algorithm can be improved if we adopt a branch and bound approach [10] only exploring those branches in the tree that may lead to the optimal path. Formally, a state in the search space is a pair (l, r) where $0 \le l < r \le m$ and $r - l > 1$, and it represents the set of cyclic shifts $\{\sigma^k(x) : l < k < r\}$. The algorithm starts by computing $d = ED(x, y)$ and initializing the set of live search states, $\mathcal{S}$, to $(0, m)$. $\mathcal{S}$ can be implemented as a min-heap [3] ordered by the value of g, a function such that $g(l, r) \le \min_{l<k<r} d(\sigma_k(x), y)$ (a lower bound on d). The branching function splits a given state (l, r) in two states, (l, k) and (k, r), and a singleton $\{k\}$, where $k = l + \lceil \frac{r-l}{2} \rceil$. The live states are implicitly pruned by not inserting new states in $\mathcal{S}$ if their lower bound does not decrease the value of d. It finishes when the lower bound of the best state in $\mathcal{S}$ is greater or equal than d.

Although in [10] other lower bounds, g, are mentioned, the one that obtains the best results in the experiments is the following:

$$g((l, r)) = \max\left(0, \frac{ED(\sigma^l(x), y) + ED(\sigma^r(x), y) - (D + I)(r - l)}{2}\right) \le \min_{l<k<r} ED(\sigma^k(x), y), \quad (2)$$

being $D = \gamma(u \to \lambda)$ and $I = \gamma(\lambda \to v)$.

This lower bound is computed in time $O(1)$. Therefore, the worst case time complexity of this algorithm is $O(mn \log m)$.

3. AESA and LAESA

In certain cases, to perform a search over the entire database is not suitable. In these cases, if we have a metric space, it is a common practice to use indexing methods based on the triangle inequality. These methods organize data in a way that similarity queries can be performed efficiently without processing the entire database. There are many methods of this type: M-tree, R-tree, vp-tree, ... [2], although, among them, when distances have a high computational cost, methods based on AESA stand out. That is precisely our case, since our distance is the CED.

AESA [22, 23] is characterised by a drastic reduction of the computation of distances. AESA requires, in a preprocessing step, the computation and storage of all the distances in a matrix D. The search procedure in AESA uses a branch and bound technique for finding the nearest prototype, in a training set P, to a query sample x. In the search step, an arbitrary candidate, s, is chosen as the nearest neighbour and the distance to the query sample, $d(x, s)$, is computed. Then, s is eliminated from the set of prototypes and the nearest neighbour is updated. This distance is used for obtaining a new lower bound, G, of the distance of each prototype, p, to the query sample, using for it the triangle inequality (see Figure 3):

$$d(p, x) \ge |d(s, p) - d(x, s)|. \quad (3)$$

If the lower bound is greater than the distance to the nearest neighbour so far, the prototype is pruned. The next prototype with a minimum value of the lower bound

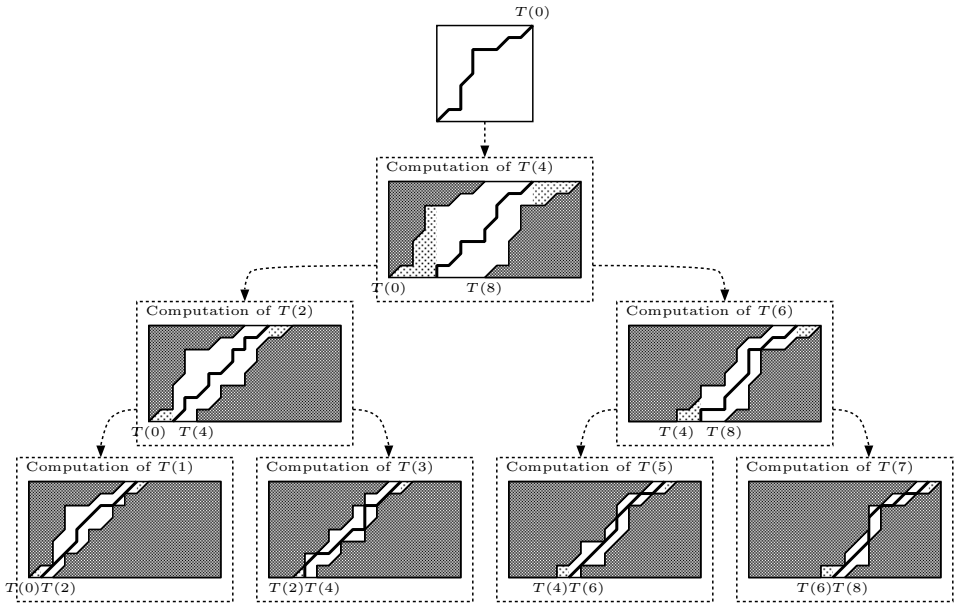

Figure 2: Search space generated for the computation of the cyclic edit distance by Maes' algorithm for $m = 8$. Each frame represents an extended edit graph. Optimal paths never cross already known optimal paths. Shaded regions in each frame are not visited when computing optimal paths.

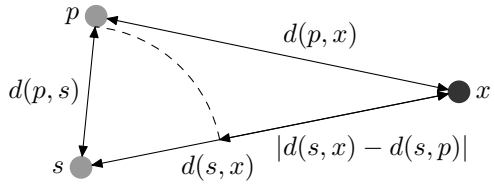

Figure 3: Lower bound using the triangle inequality.

is selected and the process is repeated until there are no more prototypes.

AESA has a major drawback, its spatial complexity is quadratic due to D. To solve this problem, LAESA arose [11]. In the preprocessing step, instead of computing the distances between all the prototypes, this computation is done just for base prototypes, B (that are selected in a previous step) against the remainder prototypes. Then, the matrix, D, is not square and the spatial complexity of LAESA becomes linear with respect to P and the cardinality of B. In the search process, base prototypes are selected as candidates to nearest neighbour and the distance to the query sample is computed, using this distance and precomputed distances (in the preprocessing step) to obtain a lower bound of the distance of each prototype to the query sample. When all the base prototypes are eliminated, the process continues but without updating the lower bounds.

The number of distances that LAESA computes is always greater than the ones of AESA. But, in [11], the authors empirically showed that the number of computed distances, in both algorithms, grows very slowly with respect to the size of P, that is, it does not depend on this size. However, an important factor for the number of computed distances is the number of base prototypes and how we select them.

4. LAESA with early abandon (LAESAEA) for cyclic strings

As we commented before, LAESA only updates the lower bounds using the computed distances of the prototypes that belong to B. It is not necessary to compute the distances to prototypes that do not belong to B because we do not need them for updating the lower bounds. This way, we can abandon the computation of these distances as soon as we know that they are not going to be lower than the distance to the nearest neighbour so far. From now on we will call this distance to this nearest neighbour: external bound. Then, when the prototype belongs to B, we compute a regular distance. Otherwise, we compute the distance with early abandon. We can now reduce the computation time of the prototypes that are not pruned by LAESA.

We call this algorithm LAESA with early abandon (LAESAEA). This new algorithm is shown in Figure 4. For transforming it to LAESA, we have to remove the condition (with comments in Figure 4) where we check if $s \in B$ and to compute always the distance function d instead of dEA (distance with early abandon). Finally, for transforming LAESA to AESA, we only have to make $B = P$. Thus, this early abandon is not applicable to AESA since $B = P$ and we have to compute all the distances for updating the lower bounds.

We cannot speed up any sort of distance using LAESAEA. It makes no sense, for instance, to use

Figure 4: LAESAEA (LAESA with early abandon).

```
Input: P: prototypes, x: query sample
Output: nn ∈ P: nearest neighbour
var B ⊂ P: base prototypes, D_B ∈ ℝ^{|B|×|P|}:
distances to base prototypes
begin
    for p ∈ P do G[p] = 0
    nn = unknown; d_nn = ∞; s = any element of B
    while |P| > 0 do
        if s ∈ B then
            d_s = d(x, s)   // computing distance as in
            LAESA
        else
            d_s = dEA(x, s, d_nn)          // computing
            distance with early abandon (dEA)
        P = P − {s}
        if d_s < d_nn then
            nn = s; d_nn = d_s
        next_B = unknown; gmin_B = ∞; next =
        unknown; gmin = ∞
        for p ∈ P do
            if s ∈ B then
            G[p] = max(G[p], |D_B[s, p] − d_s|)
            if G[p] > d_nn then
                P = P − {p}
            else
                if p ∈ B then
                    if G[p] < gmin_B then
                    gmin_B = G[p]; next_B = p
                else
                    if G[p] < gmin then
                    gmin = G[p]; next = p
        if next_B ≠ unknown then  s = next_B
        else  s = next
    return nn
end
```

Figure 5: dEA1 (Branch and bound algorithm with early abandon).

```
Input: x, y : Σ*; external_bound : ℕ
Output: d : ℕ
var 𝒮 : 2^{ℕ×ℕ}; T : array [0..m] of edit paths
begin
    d = ED(x, y)
    if external_bound < d then
        d = external_bound
    Let T(0) be the optimal path underlying
    ED(x, y)
    Let T(m) be T(0) shifted m units to the right
    𝒮 = {(0, m)}
    while 𝒮 ≠ ∅ ∧ d > min_{(l,r)∈𝒮} g((l, r)) do
        (l, r) = arg min_{(l′,r′)∈𝒮} g((l′, r′));
        𝒮 = 𝒮 − {(l, r)}
        k = l + ⌈(r−l)/2⌉
        d = min(d, ED(ρ^k(x), y)) (between T(l) and
        T(r))
        Let T(k) be the optimal edit path underlying
        ED(ρ^k(x), y)
        if k > l + 1 ∧ g((l, k)) < d then
        𝒮 = 𝒮 ∪ {(l, k)}
        if r > k + 1 ∧ g((k, r)) < d then
        𝒮 = 𝒮 ∪ {(k, r)}
    return d
end
```

LAESAEA with a distance that has a constant computational complexity. We need a distance that in the process of computing we can stop it when we know that it is going to be greater than a determined value (the external bound) and this stop must save a considerable amount of time. That is, a distance with early abandon. As we will see in the following, the CED has these properties.

We propose two ways of stopping the computation of the CED using an external bound. The first is a modification of the branch and bound procedure for computing the CED mentioned in Section 2. The adapted algorithm, dEA1, is shown in Figure 5. This modified version initializes d to the external bound if it is lower than $ED(x, y)$ (the cost of $T(0)$) and to $ED(x, y)$ otherwise. Then, the algorithm will not explore those ranges, (l, r), where the external bound is lower than their lower bound, $g((l, r))$.

The second proposal is based on a combination of the previous algorithm and the Bunke and Bühler's algorithm [1] (BB). This algorithm computes an approximation of $CED([x], [y])$ in time $O(mn)$. It obtains the shortest path starting at any node $(k, 0)$, for $0 \leq k < m$, and finishing at any node (k', n), for $m \leq k' < 2m$. This method is suboptimal because the path that it obtains could start at $(k, 0)$ and finish at (k', n), with $k' \neq k+m$. However, the path associated with $CED([x], [y])$ must verify $k' = k + m$. We can then deduce that the distance obtained using the Bunke and Bühler's algorithm is a lower bound of the CED, that is, $BB(x, y) \leq CED([x], [y])$.

Based on this reasoning we can also develop a Bunke and Bühler's algorithm with early abandon. We can abort BB computation whenever we can guarantee that the final result will not improve the external bound. That is, we can stop the computation at first row where all the values are greater or equal than this external bound. The modified version of this algorithm is shown in Figure 6.

The combination of both algorithms, dEA2, is depicted in Figure 7. Basically, we compute BBEA and if the value that returns is greater than the external bound, we return infinity. This means that the BBEA has pruned the computation of the distance of this particular element

in the database. If that is not the case, we compute dEA1. If the value that it returns is equal to the external bound, dEA1 has pruned the computation of the real distance (infinity is also returned). Otherwise, this is the new nearest neighbour and we return its distance.

Both algorithms, dEA1 and dEA2 can substitute dEA in LAESAEA (see Figure 4). Although dEA2 seems to be computationally more expensive than dEA1, we will see in the experiments section that in practice it obtains better results.

Figure 6: BBEA (Bunke and Bühler's algorithm with early abandon).

Input: $x, y : \Sigma^*$; $external\_bound : \mathbb{N}$
Output: $d : \mathbb{N}$
var $X : x$ concatenated to $x, EEG \in \mathbb{N}^{2m \times n}$: extended edit graph
begin
  for i **in** $0..m$ **do**
    $EEG[i][0] = 0$
  for i **in** $m+1..2m$ **do**
    $EEG[i][0] = \infty$
  for j **in** $0..n$ **do**
    $EEG[0][j] = EEG[0][j-1] + \gamma(X[0], y[j])$
  for j **in** $1..n$ **do**
    for i **in** $1..2m$ **do**
      $EEG[i][j] =$
      $\min(EEG[i-1][j-1]+\gamma(X[i-1], y[j-1]),$
      $EEG[i-1][j]+\gamma(X[i], \lambda),$
      $EEG[i][j-1])+\gamma(\lambda, y[j]))$
    if every element at row j is $\geq$ *external_bound* **then**
      $d = \infty$
      return d
  $d = \min(EEG[i][n-1]$ **for** i in $|X|-1 \,..\, 2|X|-1)$
  return d
end

Other two important things that we have to consider in order to speed up classification and retrieval are the following (both must be applied at the beginning of dEA1 and dEA2 algorithms). First, if $m > n$, we must swap x and y, in order to have a lower computational complexity $O(nm \log n)$. Second, we do not need to compute the distance if $m > n$ and $(m-n)\min_{u\in\Sigma}\gamma(u \to \lambda) \geq$ *external_bound*, or $n > m$ and $(n-m)\min_{v\in\Sigma}\gamma(\lambda \to v) \geq$ *external_bound*. This happens because at least $|m-n|$ insertions or deletions must be used to transform one string into the other.

Finally, we can also apply our approach for retrieval of samples or for k-nearest neighbour classification. We simply have to use a sorted list of the k nearest neighbours and to use the kth nearest neighbour as the external bound.

Figure 7: dEA2.

Input: $x, y : \Sigma^*$; $external\_bound : \mathbb{N}$
Output: $d : \mathbb{N}$
begin
  if $BBEA(x, y, external\_bound) > external\_bound$ **then**
    return ∞
  $d = dEA1(x, y, external\_bound)$
  if $d == external\_bound$ **then**
    return ∞
  else
    return d
end

5. Evaluation and results

We performed comparative experiments in classification and retrieval tasks on the following databases:

- Digits from the NIST Special Database 19 [16] (see Figure 8). A total of 5000 digits were randomly selected from the sets hsf_{0,1,2,3,4}. Each digit has 500 samples.
- Silhouette database [21]. It contains 1070 shapes (see Figure 9). The shapes belong to 41 classes representing different objects.
- Cybase [14, 25], a database of proteins. It consists of 434 cyclic proteins.

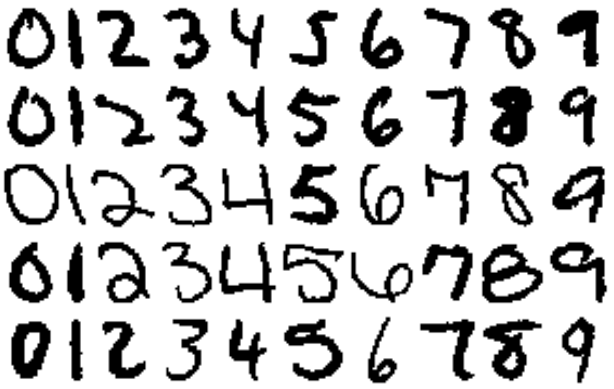

Figure 8: Some digits in NIST Special Database 19.

The images were scaled, binarized and their contours were represented by chain codes of eight directions. The average length of these cyclic strings is: 100 for the digits, 190 for the shapes of the Silhouette database and 27 for the proteins.

Time experiments were measured on a 2.66 GHz Intel i7 running under Linux 3.2. The algorithms were implemented in C++ and its code was compiled using the

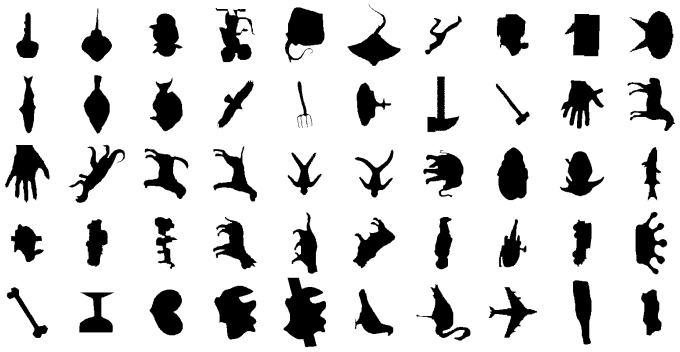

Figure 9: Some images in Silhouette database.

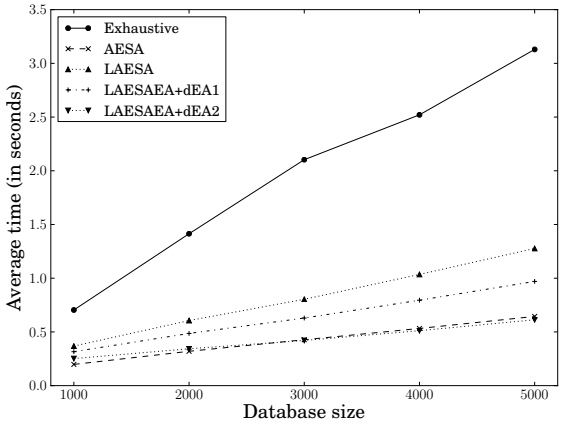

Figure 10: Average time needed to classify a query sample as a function of the number of prototypes in the training set for the digits database.

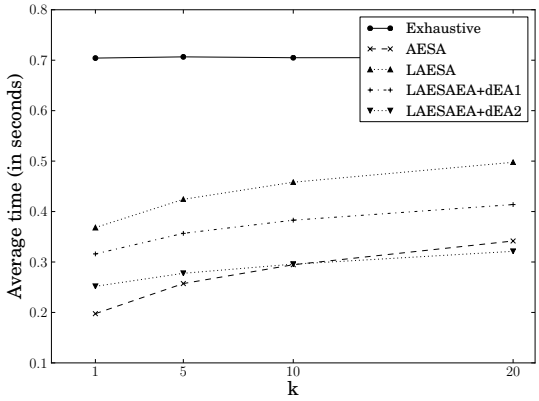

Figure 11: Average time needed for obtaining the nearest neigbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the digits database of 1000 samples.

GNU g++ compiler with -O2 optimizations. We used a leaving-one-out approach [4]. The cyclic edit distances were computed using unit cost for insertions, deletions and substitutions of symbols, and zero cost for matchings.

We compare our approaches, LAESAEA using dEA1 (LAESAEA+dEA1) and dEA2 (LAESAEA+dEA2), with respect to an exhaustive search (Exhaustive), LAESA and AESA.

Time results of the digits database are shown in Figures 10, 11, 12, 13, 14 and 15. With this database we created other 4 databases for performing the comparisons with different number of samples: 1000, 2000, 3000 and 4000 (the original database has 5000 samples). In Figure 10 the classification results ($k = 1$) for different sizes are shown. As we can see, both, LAESAEA+dEA1 and mainly LAESAEA+dEA2 outperform the results of LAESA. The results of LAESAEA+dEA2 and AESA are very similar. LAESAEA+dEA2 has better results for a size greater than 4000 samples.

The classification rate in the whole dataset (5000 samples) is 96.88% using the ED and 98.34% using the CED.

In Figures 11, 12, 13, 14 and 15 we have results for different values of k for shape retrieval or k nearest neighbour classification (a figure for 1000, 2000, 3000, 4000 and 5000 prototypes). We can see that LAESAEA+dEA2 has a very good behaviour with respect to AESA. LAESAEA+dEA2 obtains the best result when k or the size of the database begin to grow.

Time results of the Silhouette database for different values of k are in Figure 16. These results are very similar to the ones of the digits database for 1000 and 2000 samples. The Silhouette database has 1069 samples and longer strings. As we can see, for $k = 1$ LAESAEA+dEA2 obtains a close result to the one of AESA. For $k = 10$ and $k = 20$, LAESAEA+dEA2 improves the result of AESA.

Here we obtain a classification rate of 96.63% using the CED and 81.66% using the ED. For the shape retrieval we use the mean average precision (MAP) [9]. The MAP using the CED, in this database, is 64.00 and using the ED is 22.81. We have to say that in this task the results of the CED are much better because we solve a different problem [7] than the one of the digits database. In the digits database, we reduce the error of choosing as the beginning of the contour the upper leftmost point. However, in the Silhouette database we suppose any rotation of the shapes. Then, we have a random selection of the starting point.

Figure 17 shows the experiments with Cybase, the database of cyclic proteins. Although these time results are competitive, they are not as good as the ones with the other databases. This is due to the small size of this database and it also has shorter strings.

Finally, in Figures 18, 19 and 20 we show the experiments for selecting the number of base prototypes for each database. For selecting B we use the EC1 approximation [11] as it is done in posterior works of the same authors [6, 12]. In these figures a cross indicates the number of base prototypes selected. For speeding up the search, it is performed from 10 to 10 starting at $B = 5$.

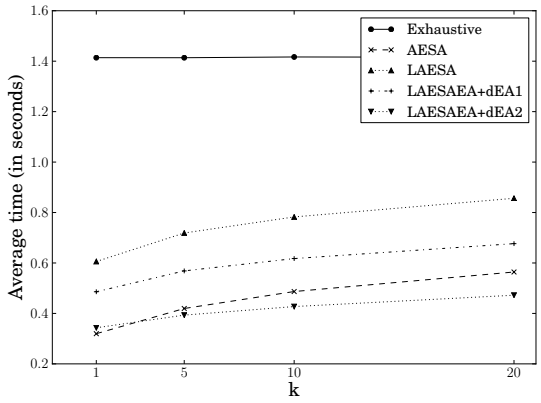

Figure 12: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the digits database of 2000 samples.

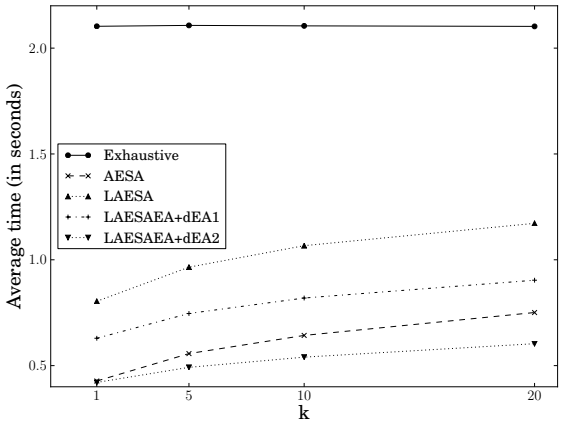

Figure 13: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the digits database of 3000 samples.

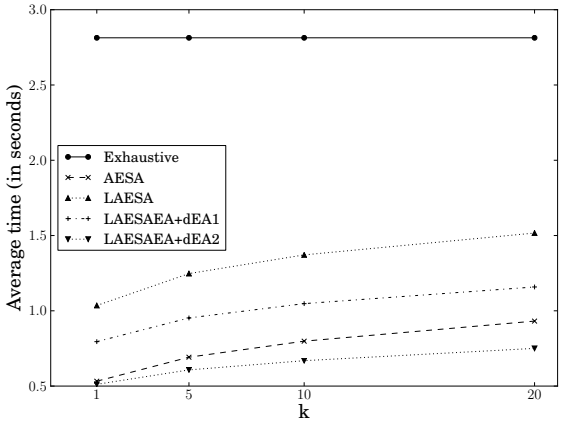

Figure 14: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the digits database of 4000 samples.

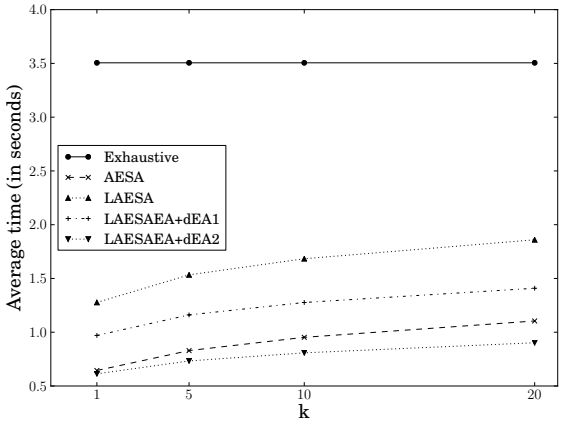

Figure 15: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the digits database of 5000 samples.

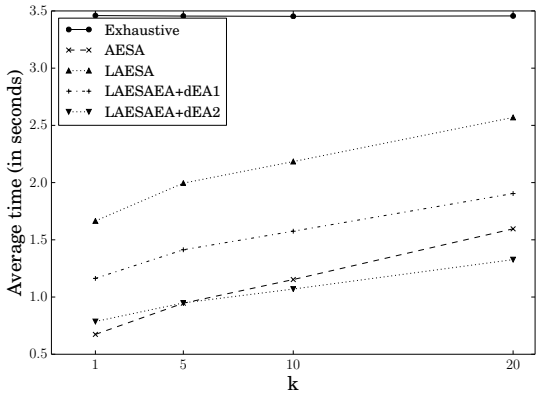

Figure 16: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the Silhouette database.

6. Discussion

AESA has a major drawback, its spatial complexity is quadratic. When the database begins to grow LAESA is instead used because of its linear spatial complexity. But, its results are not as good as the ones of AESA. In this work, we propose a modified version of LAESA to work with distances with early abandon such as the CED. We call this version LAESA with early abandon (LAESAEA). For stopping the computation of the CED we propose two different methods: dEA1 and dEA2. Experimental results with three databases in classification and retrieval show that the proposed method considerably reduces the required time of LAESA for cyclic strings. In most cases LAESAEA+dEA2 offers similar results to the ones of AESA or even better results. But, if we also consider the spatial complexity, LAESAEA+dEA2 is the best option.

In posterior work we want to explore the application of LAESAEA in other contexts.

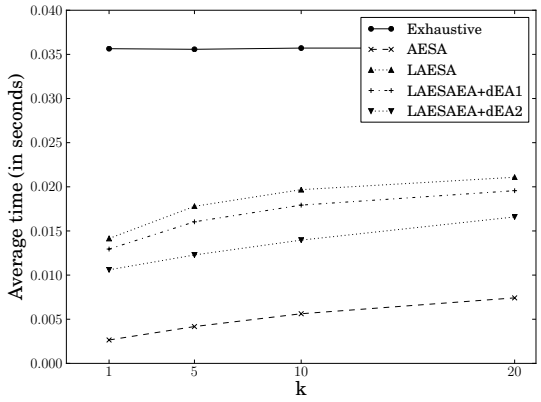

Figure 17: Average time needed for obtaining the nearest neighbours of a query sample as a function of k (the number of nearest neighbours retrieved) for the Cybase database.

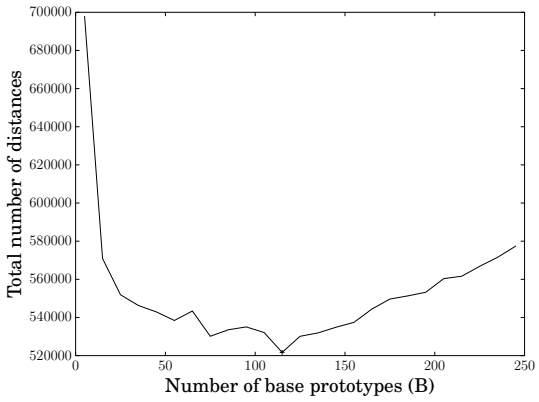

Figure 19: Number of distances as a function of the number of base prototypes (B) for the Silhouette database. A cross indicates the number of base prototypes selected.

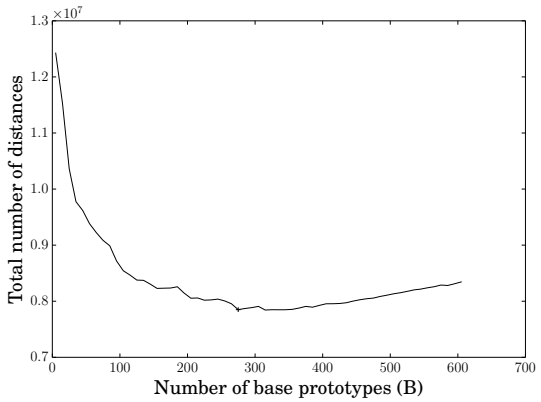

Figure 18: Number of distances as a function of the number of base prototypes (B) for the digits database. A cross indicates the number of base prototypes selected.

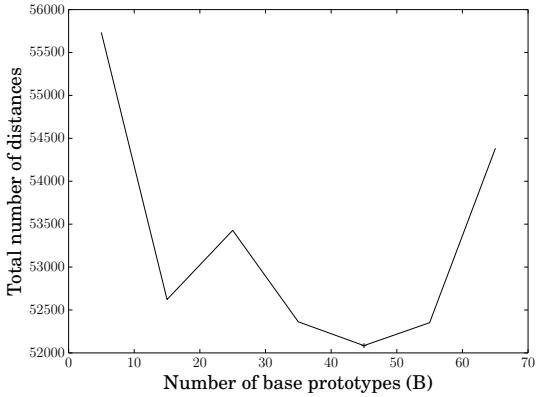

Figure 20: Number of distances as a function of the number of base prototypes (B) for the Cybase database. A cross indicates the number of base prototypes selected.

Acknowledgements

Work partially supported by the Spanish Government (TIN2010-18958), and the Generalitat Valenciana (PROMETEOII/2014/062).

References

[1] Bunke, H., Bühler, H., 1993. Applications of approximate string matching to 2D shape recognition. Pattern Recognition 26, 1797–1812.

[2] Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J., 2001. Searching in metric spaces. ACM Computing Surveys (CSUR) 33, 273–321.

[3] Cormen, T., Leiserson, C., Rivest, R., 1990. Introduction to Algorithms. The MIT Press Cambridge, MA.

[4] Duda, R.O., Hart, P.E., 1973. Pattern Classification and Scene Analysis. Wiley.

[5] Fernandes, F., Pereira, L., Freitas, A.T., 2009. Csa: An efficient algorithm to improve circular dna multiple alignment. BMC bioinformatics 10, 230.

[6] Juan, J.R.R., Mico, L., 2003. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. Pattern Recognition Letters 24, 1417–1426.

[7] Keogh, E., Wei, L., Xi, X., Vlachos, M., Lee, S., Protopapas, P., 2009. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. The VLDB Journal 18, 611–630.

[8] Maes, M., 1990. On a cyclic string-to-string correction problem. Information Processing Letters 35, 73–78.

[9] Manning, C.D., Raghavan, P., Schtze, H., 2008. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA.

[10] Marzal, A., Barrachina, S., 2000. Speeding up the computation of the edit distance for cyclic strings, in: International Conference on Pattern Recognition, pp. 483–519.

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

[12] Moreno-Seco, F., Mico, L., Oncina, J., 2003. A modification of the LAESA algorithm for approximated k-NN classification. Pattern Recognition Letters 24, 47–53.

[13] Mosig, A., Hofacker, I.L., Stadler, P.F., 2006. Comparative analysis of cyclic sequences: Viroids and other small circular rnas., in: German Conference on Bioinformatics, pp. 93–102.

[14] Mulvenna, J.P., Wang, C., Craik, D.J., 2006. Cybase: a database of cyclic protein sequence and structure. Nucleic acids research 34, 192–194.

[15] Nicolas, Rivals, 2007. Longest common subsequence problem for unoriented and cyclic strings. Theoretical Computer Science 370, 1–18.

[16] P. J. Grother, 1995. NIST Special Database 19: Handprinted Forms and Characters Database. Technical Report. National Institute of Standards and Technology.

[17] Palazón-González, V., Marzal, A., 2012. On the dynamic time warping of cyclic sequences for shape retrieval. Image and Vision Computing 30, 978–990.

[18] Palazón-González, V., Marzal, A., Vilar, J.M., 2014. On hidden markov models and cyclic strings for shape recognition. Pattern Recognition 47, 2490–2504.

[19] Peris, G., Marzal, A., 2002. Fast cyclic edit distance computation with weighted edit costs in classification, in: International Conference on Pattern Recognition, pp. 184–187.

[20] Sankoff, D., Kruskal, J. (Eds.), 1983. Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison. Addison-Wesley, Reading, MA.

[21] Sharvit, D., Chan, J., Tek, H., Kimia, B.B., 1998. Symmetry-based indexing of image databases, in: Workshop on Content-Based Access of Image and Video Libraries, pp. 56–62.

[22] Vidal, E., 1986. An algorithm for finding nearest neighbours in (approximately) constant average time. Pattern Recognition Letters 4, 145–157.

[23] Vidal, E., 1994. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). Pattern Recognition Letters 15, 1–7.

[24] Wagner, R., Fisher, M., 1974. The string-to-string correction problem. Journal of the ACM 21, 168–173.

[25] Wang, C.K., Kaas, Q., Chiche, L., Craik, D.J., 2008. Cybase: a database of cyclic protein sequences and structures, with applications in protein discovery and engineering. Nucleic acids research 36, 206–210.

[26] Zhang, D., Lu, G., 2004. Review of shape representation and description techniques. Pattern Recognition 37, 1–19.