Computing the Cyclic Edit Distance for Pattern Classification by Ranking Edit Paths[⋆]

Víctor M. Jiménez, Andrés Marzal, Vicente Palazón, and Guillermo Peris

DLSI, Universitat Jaume I, 12071 Castellón, Spain
`{vjimenez,amarzal,palazon,peris}@uji.es`

Abstract. The cyclic edit distance between two strings A and B of lengths m and n is the minimum edit distance between A and every cyclic shift of B. This can be applied, for instance, in classification tasks where strings represent the contour of objects. Bunke and Bühler proposed an algorithm that approximates the cyclic edit distance in time $O(mn)$. In this paper we show how to apply a technique for ranking the K shortest paths to an edit graph underlying the Bunke and Bühler algorithm to obtain the exact solution. This technique, combined with pruning rules, leads to an efficient and exact procedure for nearest-neighbour classification based on cyclic edit distances. Experimental results show that the proposed method can be used to classify handwritten digits using the exact cyclic edit distance with only a small increase in computing time with respect to the original Bunke and Bühler algorithm.

Keywords: Cyclic strings, cyclic edit distance, string matching, Bunke and Bühler algorithm, handwritten text recognition, OCR, K shortest paths.

1 Introduction

Measuring dissimilarities between strings is a fundamental problem in pattern recognition [1]. The most widely used measure of dissimilarity between two strings is the *edit distance* (ED), also known as the *weighted Levensthein distance*, which is defined as the weight of the best sequence of edit operations (insertions, substitutions and deletions of symbols) needed to transform one string into the other [2].

There are many applications where the objects are better modelled by *cyclic strings*, which are strings whose last symbol is considered to be followed by the first symbol. For instance, contours of objects can be appropriately represented by cyclic chain-codes [3, 4] (see Fig. 1). The dissimilarity between cyclic strings can be measured by means of the *cyclic edit distance* (CED), which is defined as the weight of the best sequence of edit operations needed to transform any cyclic shift of one string into any cyclic shift of the other. A trivial way to obtain

⋆ This work has been supported by the *Generalitat Valenciana* under grant CTIDIA/2002/209 and by the Spanish *Ministerio de Ciencia y Tecnología* and FEDER under grant TIC2002-02684.

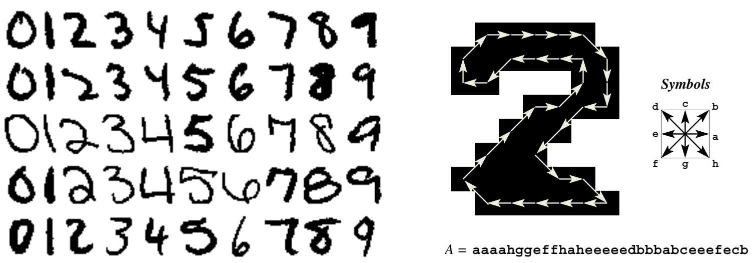


Fig. 1. Example digits from the NIST Special Database 19 and a string representing the contour of a digit with an 8-directional chain-code.

the cyclic edit distance in $\mathcal{O}(mn^2)$ time consists in computing the edit distance between one string and all the possible cyclic shifts of the other. Maes [5] proposed a divide and conquer algorithm that reduces the time cost to $\mathcal{O}(mn \log n)$. Marzal, Barrachina, and Peris [6, 7] reformulated this method as a branch and bound algorithm and proposed bounding functions that produce a significant speeding up of Maes' algorithm while maintaining its worst-case complexity.

In applications where the running-time of the algorithm is a major concern (for instance, in classification systems in which the CED between every string to be classified and a large number of labelled samples is computed), alternative approximate methods that run faster than the exact methods can be used. A well-known approximate method to compute the CED is the Bunke and Bühler algorithm (BBA), which runs in $\mathcal{O}(mn)$ time [3]. Mollineda, Vidal, and Casacuberta have proposed other approximate solutions based on the BBA that require a training stage [8, 9].

In this paper, we present a new *exact* method to compute the CED. Our proposal is based on the BBA, combined with an efficient technique for finding the K shortest paths in graphs [10], which is adapted to this problem. In classification tasks, this method can be combined with pruning rules to abort the computation of distances with values above the best distance found so far. In this way, according to experimental results reported in this paper, the value of K needed to find the exact solution is quite low in practice and the total running time is only slightly greater than the time needed by the BBA to find an approximate solution.

2 Notation and Problem Formulation

Let Σ be a set of symbols and let $\Sigma^\star$ be the set of all finite strings over Σ. Let a, b denote symbols in Σ and let λ denote the empty string. Throughout this paper, we consider that $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$ are strings in $\Sigma^\star$ of length m and n, respectively.

Edit Distance. An *edit sequence* is a sequence $E = e_1 e_2 \dots e_p$ in which e_i, for $1 \le i \le p$, is one of four possible edit operations: (i) deletion of a symbol a, denoted $a \to \lambda$; (ii) insertion of a symbol b, denoted $\lambda \to b$; (iii) substitution of a symbol a by a symbol b, denoted $a \to b$; and (iv) matching (substitution of a symbol a by itself), denoted $a \to a$. Let $\gamma(E) = \sum_{i=1}^{p} \gamma(e_i)$ be the weight of the edit sequence E, with $\gamma(e_i)$ being the weight of the edit operation e_i. The *edit distance* $ED(A, B)$ is defined as the minimum value of $\gamma(E)$ for any edit sequence E that transforms A into B.

Cyclic Edit Distance. Let $\sigma(A) = a_2 a_3 \dots a_m a_1$ denote a *cyclic shift* of A, and let $\sigma^j(A) = a_{j+1} a_{j+2} \dots a_m a_1 \dots a_j$ denote the composition of j cyclic shifts. In many applications (for instance, in classification tasks where strings represent contours of objects) it makes sense to consider that the strings A and $\sigma^j(A)$, for any $j \in \mathbb{N}$, are equivalent. The equivalence class $[A] = \{\sigma^j(A) : j \in \mathbb{N}\}$ is called a *cyclic string*. The *cyclic edit distance* $CED([A], [B])$ is a measure of dissimilarity between the classes that the strings A and B represent, and is defined as $CED([A], [B]) = \min_{i,j \in \mathbb{N}} ED(\sigma^i(A), \sigma^j(B))$, which is the same as $CED([A], [B]) = \min_{1 \le j \le n} ED(A, \sigma^j(B))$ [5].

In this paper, we are interested in computing $CED([A], [B])$ for any given pair of strings A and B. In the next section we review how a refinement of the Bunke and Bühler algorithm, which approximates the value of $CED([A], [B])$ in time $O(mn)$, can be seen as an algorithm for finding a shortest s-t path in a graph. Then in Sect. 4 we will see how an algorithm for finding the K shortest s-t paths can be adapted to compute the exact value of $CED([A], [B])$.

3 Approximating the Cyclic Edit Distance with the Bunke and Bühler Algorithm

The computation of the edit distance $ED(A, B)$ using the Wagner and Fischer algorithm [2] can be formulated in terms of finding the shortest path between a pair of nodes in a graph G_A^B (the so-called *edit graph*). The nodes of G_A^B are all the pairs (i, j) for $0 \le i \le m$ and $0 \le j \le n$. There are (at most) three incoming edges for each node (i, j) (see Fig. 2a): (i) coming from $(i-1, j)$, if $i > 0$, with weight $\gamma(a_i \to \lambda)$; (ii) from $(i, j-1)$, if $j > 0$, with weight $\gamma(\lambda \to b_j)$; and (iii) from $(i-1, j-1)$, if $i > 0$ and $j > 0$, with weight $\gamma(a_i \to b_j)$. The edit distance $ED(A, B)$ is the weight of the shortest path between nodes $s = (0, 0)$ and $t = (m, n)$. The edit graph is acyclic and has $O(mn)$ edges; therefore, the shortest s-t path can be found in $O(mn)$ time by following any topological order of nodes [11].

In order to compute the cyclic edit distance $CED([A], [B])$, we can consider the edit graph G_A^{BB} associated to $ED(A, BB)$, the edit distance between A and B concatenated with itself. In this graph, the shortest path from the node $s = (0, j)$ to the node $t = (m, n + j)$, for every $j = 1, 2, \dots, n$, represents the best edit sequence that transforms A into $\sigma^j(B)$, whose weight is $ED(A, \sigma^j(B))$. The minimum of these n weights is $CED([A], [B])$. This value can be computed in time $O(mn^2)$ by just running a shortest s-t path algorithm for each of these n s-t pairs.

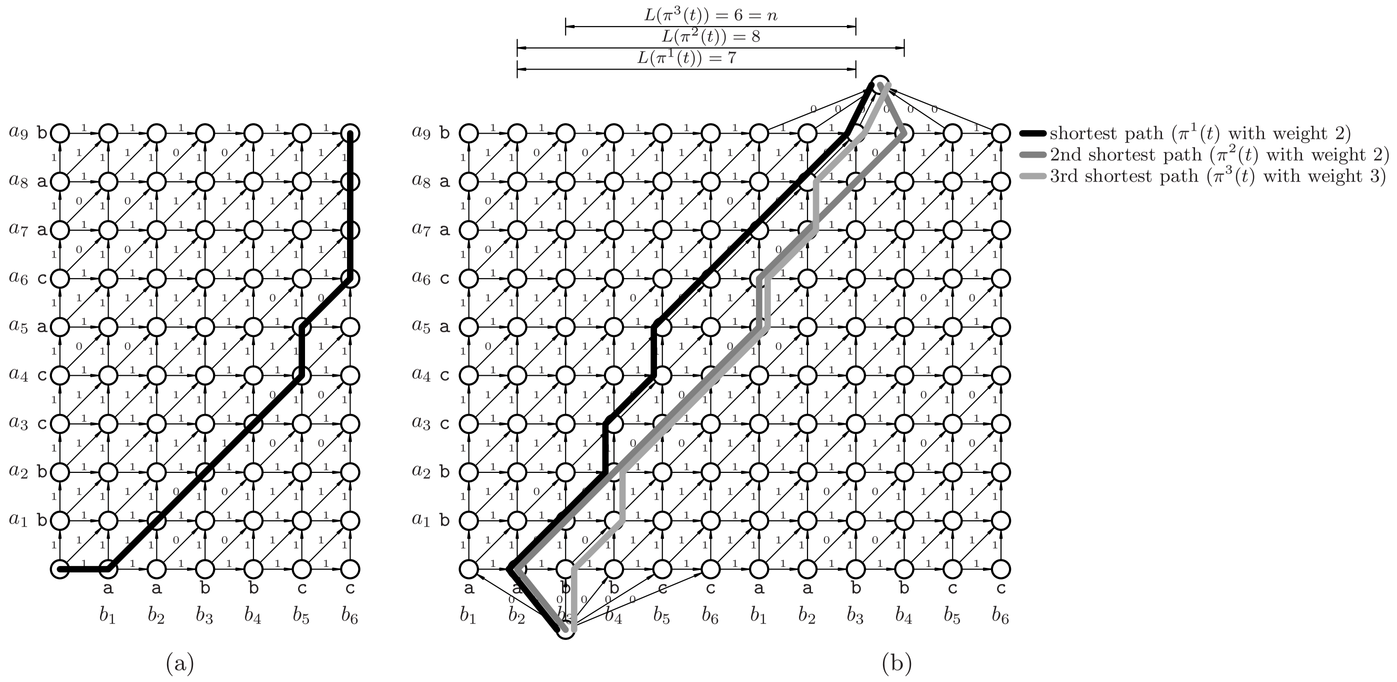

$L(\pi^3(t)) = 6 = n$
$L(\pi^2(t)) = 8$
$L(\pi^1(t)) = 7$
shortest path ($\pi^1(t)$ with weight 2)
2nd shortest path ($\pi^2(t)$ with weight 2)
3rd shortest path ($\pi^3(t)$ with weight 3)
a_9 b
a_8 a
a_7 a
a_6 c
a_5 a
a_4 c
a_3 c
a_2 b
a_1 b
a a b b c c
b_1 b_2 b_3 b_4 b_5 b_6
a a b b c c a a b b c c
b_1 b_2 b_3 b_4 b_5 b_6 b_1 b_2 b_3 b_4 b_5 b_6
(a)
(b)

Fig. 2. Let $A =$bbccacaab and $B =$aabbcc represent two cyclic strings. (a) Edit graph G_A^B. (b) Edit graph $\bar{G}_A^{BB}$. In this example, matchings have a weight of 0 and any other edit operation has a weight of 1. The Wagner and Fischer algorithm finds $ED(A, B) = 7$, the weight of the shortest s-t path in G_A^B. The Bunke and Bühler algorithm obtains 2 as an approximate value of $CED([A], [B])$: the weight of the shortest s-t path in $\bar{G}_A^{BB}$. The exact value of $CED([A], [B])$ is 3: the weight of $\pi^3(t)$, the shortest s-t path π in $\bar{G}_A^{BB}$ such that $L(\pi) = 6 = n$, computed using the method described in Sect. 4.

An approximate value (a lower bound) of $CED([A],[B])$ can be found more efficiently, in time $O(mn)$, by finding, in the edit graph G_A^{BB}, the shortest path starting at any node in $S = \{(0,j) : 1 \le j \le n\}$ and finishing at any node in $T = \{(m, n+j) : 1 \le j \le n\}$. This is equivalent to finding the shortest path from s to t in the graph obtained from G_A^{BB} by removing the nodes $\{(i,0) : 0 \le i \le m\}$ and the edges departing from them, and adding edges of weight 0 from an extra node s to every node in S and from every node in T to an extra node t. Let $\bar{G}_A^{BB}$ denote the resulting graph (see Fig. 2b). Again, this is a shortest s-t path problem in an acyclic graph with $O(mn)$ edges and can be solved in $O(mn)$ time. More precisely, it takes twice the time required to compute $ED(A,B)$. We call this method the Bunke and Bühler Algorithm (BBA) since a similar proposal to estimate a lower bound of $CED([A],[B])$ was originally made by Bunke and Bühler in [3]. The suboptimality of this method is due to the fact that the optimal path that it finds could start going from s to $(0,j)$ and finish going from (m,j') to t, with $j' \neq n+j$, while the path corresponding to $CED([A],[B])$ should verify $j' = n+j$. In the next section, we will see how an algorithm to enumerate the K shortest s-t paths in a weighted graph, the so-called *Recursive Enumeration Algorithm* (REA) [10], can be adapted to find the path with the minimum weight verifying $j' = n+j$. The weight of such a path is the exact value of $CED([A],[B])$.

4 Computing the Cyclic Edit Distance by Ranking Paths in the Bunke and Bühler Edit Graph

Let V be the set of nodes and let E be the set of edges in $\bar{G}_A^{BB}$. Given a path π and a node v, let $\pi \cdot v$ denote the path formed by π followed by v. For any path π in $\bar{G}_A^{BB}$ that starts going from s to $(0,j)$ and ends at (i,j'), as well as for any path π in $\bar{G}_A^{BB}$ that starts going from s to $(0,j)$ and ends by going from (m,j') to t, let us define $L(\pi) = j' - j$ (see Fig. 2b). In order to compute the exact value of $CED([A],[B])$, we are interested in finding the path π from s to t with the minimum weight among those verifying $L(\pi) = n$. This can be done by enumerating, by ascending weight value, the paths from s to t until the first path verifying $L(\pi) = n$ is found, as follows [10]:

```
A.1 Compute $\pi^1(v)$, the shortest path from $s$ to $v$, for all $v \in V$ and set $k \leftarrow 1$.
A.2 While $L(\pi^k(t)) \neq n$ do:
    A.2.1 Set $k \leftarrow k+1$ and compute $\pi^k(t)$ by calling $NextPath(t,k)$.
```

For $k > 1$, and once $\pi^1(v)$, $\pi^2(v)$,..., $\pi^{k-1}(v)$ are available, $NextPath(v,k)$ computes $\pi^k(v)$ as follows:

```
B.1 If $k = 2$, then initialise a set of candidates to the next shortest path from $s$
    to $v$, $C[v] \leftarrow \{\pi^1(u) \cdot v : (u,v) \in E$ and $\pi^1(v) \neq \pi^1(u) \cdot v\}$.
```

```
B.2 If $v = s$, then $\pi^k(v)$ does not exist; else
    B.2.a Let $u$ and $k'$ be the node and index such that $\pi^{k-1}(v) = \pi^{k'}(u) \cdot v$. If
          $\pi^{k'+1}(u)$ has not already been computed, then compute it by calling
          $NextPath(u, k'+1)$.
    B.2.b If $\pi^{k'+1}(u)$ exists, then insert $\pi^{k'+1}(u) \cdot v$ in $C[v]$.
    B.2.c If $C[v] = \emptyset$, then $\pi^k(v)$ does not exist.
    B.2.d If $C[v] \neq \emptyset$, then extract the path $\pi$ with minimum weight from $C[v]$
          and let $\pi^k(v) \leftarrow \pi$.
```

Proof of correctness of this method to compute the K shortest s-t paths in a weighted graph can be found in [10]. In this particular application to compute the CED, the algorithm runs in $O(mn + K(m+n))$ time: each of the K shortest paths is computed by recursively visiting, at most, the nodes of the previous shortest s-t path [10], and each s-t path in $\bar{G}_A^{BB}$ has $O(m+n)$ nodes.

The algorithm can be speeded up in this application by taking into account that we are not interested in the K shortest paths, but only in the first s-t path π that satisfies the restriction $L(\pi) = n$. Therefore, the partial paths that do not lead to a new s-t path with a different value of L can be discarded. This can be done by simply replacing Step B.2.d by:

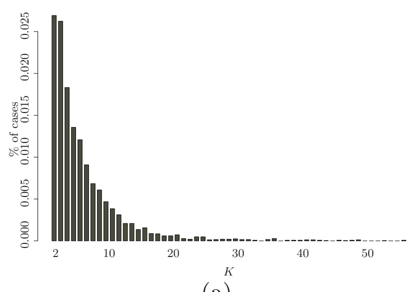
```
B.2.d If $C[v] \neq \emptyset$, then extract the path $\pi$ with minimum weight from $C[v]$. If
      $L(\pi) \neq L(\pi^j(v))$, for all $j = 1, 2, \ldots, k-1$, then let $\pi^k(v) \leftarrow \pi$; else
      B.2.d.1 Let $u$ and $k'$ be the node and index such that $\pi = \pi^{k'}(u) \cdot v$.
              If $\pi^{k'+1}(u)$ has not already been computed, then compute it by
              calling $NextPath(u, k'+1)$.
      B.2.d.2 Goto B.2.b
```

With this modification, $\pi^k(v)$ is the path from s to v with minimum weight such that $L(\pi^k(v))$ is different from $L(\pi^j(v))$ for all $j \in \{1, 2, \ldots, k-1\}$.

5 Pruning the Search Space in Classification Tasks

The method described in Sect. 4 can be further speeded up in nearest-neighbour classification, where we have N labelled samples, B^1, B^2, ..., B^N, and we want to compute $\min_{1 \leq i \leq N} CED([A], [B^i])$ in order to classify A. Let us assume that we have already computed $d_{j-1} = \min_{1 \leq i < j} CED([A], [B^i])$ and that we are going to compute $d_j = \min\{d_{j-1}, CED([A], [B^j])\}$. The computation of $CED([A], [B^j])$ can be aborted as soon as we know that its value cannot be lower than d_{j-1}, according to these rules:

1. The computation can be avoided if $m > n$ and $(m-n) \min_{a \in \Sigma} \gamma(a \to \lambda) \geq d_{j-1}$, or $n > m$ and $(n-m) \min_{b \in \Sigma} \gamma(\lambda \to b) \geq d_{j-1}$. This rule is based on the fact that at least $|m-n|$ insertions or deletions must be performed to transform one string into the other.
2. Taking into account that the edit weights are non-negative, the execution of Step A.1 can be aborted, for any $i \in \{1, 2, \ldots, m\}$, if the weight of $\pi^1((i,j))$ is greater than or equal to d_{j-1} for all $j \in \{1, 2, \ldots, 2n\}$.



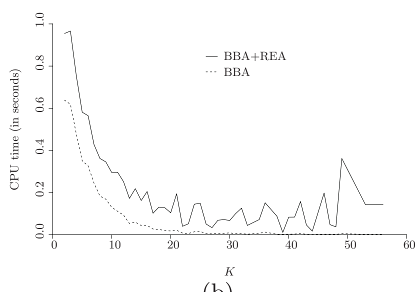


Fig. 3. (a) Percentage of cases for which K shortest s-t paths with different value of L have to be computed, for $K > 1$. (b) Total CPU time invested in those cases.

3. The ranking of s-t paths by Step A.2 can be stopped as soon as we reach a value of k such that the weight of $\pi^k(t)$ is greater than or equal to d_{j-1}.

None of these rules modify the worst-case computational complexity of the algorithm and, in practice, they entail a significant reduction in running time. They can also be extended to deal with the N nearest-neighbours classification rule.

6 Experimental Results

In order to assess the behaviour of the algorithm in practice, we performed experiments on a handwritten digits recognition task. A test set containing 500 digit images (5 instances of each digit by 10 writers) randomly selected from the hsf_4 set in the NIST Special Database 19 [12], was used (see Fig. 1). Each test digit was compared to 5 000 labelled instances from the sets hsf_{0,1,2,3} (5 instances of each digit by 25 writers from each set) in order to perform a nearest-neighbour classification. All the images were clipped, scaled into a 32×32 pixels matrix and binarised, and their outer contours were represented by 4-directional chain-codes. The average length of the resulting cyclic strings is 125. The edit distances were then computed assuming unit weight for insertions, deletions and substitutions of symbols, and zero weight for matchings.

The classification error rate is 8.6% using the (non-cyclic) edit distance, 3.8% using the approximate cyclic edit distance obtained with the BBA, and 3.2% using the exact cyclic edit distance. This confirms previous results showing that the classification using the exact cyclic edit distance performs better than the approximate method [9].

In principle, 2 500 000 cyclic edit distances had to be computed in order to classify the 500 test digits. The method proposed in this paper only required the computation of the K shortest s-t paths with different value of L, for K greater than 1, in 0.15% of the cases. Figure 3a shows a histogram with the percentage of cases for each value of K. It can be seen that computing the exact CED never required the computation of more than 60 shortest paths (the average value of K, when $K > 1$ shortest paths had to be computed, was 6.53).

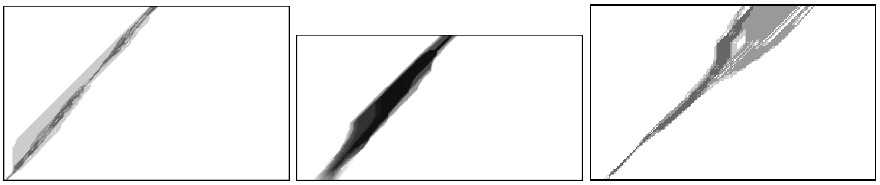


Fig. 4. Edit graph $\bar{G}_A^{BB}$, colouring the region where the REA searches for alternative paths, for 3 cases in which the shortest path is not the exact solution. A darker colour represents a higher number of computed paths.

The total time required to classify the 500 test digits was 763.89 seconds on a 2.4GHz Pentium 4 running under Linux 2.4 (the algorithms were implemented in C). The execution of the BBA accounts for 754.09 seconds. Only 9.80 seconds (1.28% of the total running time) were devoted to computing alternative paths with the REA. Figure 3b shows, for the cases in which $K > 1$ shortest s-t paths have been computed, the total running time and the running time of the BBA. It can be observed that, for the largest values of K, the execution time of the REA is significantly greater than the time due to the BBA but, thanks to the pruning rules given in Sect. 5, such values are required in a very small percentage of cases, and they hardly affect the total running time of the classification procedure.

In practice, the efficiency of the REA not only depends on the number of computed paths, but also on the number of internal nodes in which alternative paths must be computed. Figure 4 shows these nodes for three different cases and illustrates that only a small region of the graph needs to be visited when looking for alternative paths.

7 Conclusions

The algorithm proposed by Bunke and Bühler [3] computes very efficiently an approximate value of the cyclic edit distance between two cyclic strings. In this paper, we have shown how a K shortest paths algorithm [10] can be adapted to this problem and applied to an edit graph underlying the Bunke and Bühler algorithm in order to find the first shortest path satisfying a particular restriction. The weight of this path is the exact cyclic edit distance. In classification tasks, this method can be combined with pruning rules to abort the computation of distances with values above the best distance found so far. Experimental results with a handwritten digit classification system show that the proposed method can serve to reduce the error rate and only entails a very small increase in computing time with respect to the approximate method.

References

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

2. R.A. Wagner, M.J. Fischer: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–173
3. H. Bunke, H. Bühler: Applications of approximate string matching to 2D shape recognition. Pattern Recognition **26** (1993) 1797–1812
4. D. Zhang, G. Lu: Review of shape representation and description techniques. Pattern Recognition **37** (2004) 1–19
5. M. Maes: On a cyclic string-to-string correction problem. Information Processing Letters **35** (1990) 73–78
6. A. Marzal, S. Barrachina: Speeding up the computation of the edit distance for cyclic strings. Int. Conf. on Pattern Recognition (2000) 271–280
7. G. Peris, A. Marzal: Fast cyclic edit distance computation with weighted edit costs in classification. Int. Conf. on Pattern Recognition (2002) 184–187
8. R. A. Mollineda, E. Vidal, F. Casacuberta: Efficient techniques for a very accurate measurement of dissimilarities between cyclic patterns. Lecture Notes in Computer Science **1876** (2000) 337–346
9. A. Marzal, R. Mollineda, G. Peris, E. Vidal: Cyclic string matching: efficient exact and approximate algorithms. In D. Chen, X. Cheng, eds.: Pattern Recognition and String Matching. Kluwer Academic (2002) 477–497
10. V. M. Jiménez, A. Marzal: Computing the K shortest paths: a new algorithm and an experimental comparison. Lecture Notes in Computer Science **1668** (1999) 15–29
11. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press, Cambridge, MA (1990)
12. P. J. Grother: NIST Special Database 19: Handprinted forms and characters database. Technical report, National Institute of Standards and Technology (1995)