An improved fast edit approach for two-string approximated mean computation applied to OCR.

J. Abreu*[,a], J. R. Rico-Juan[b]

[a]*Dpto Informática, Universidad de Matanzas, Carretera a Varadero Km. 3 1/2, Matanzas, Cuba*
[b]*Dpto Lenguajes y Sistemas Informáticos, Universidad de Alicante, San Vicente del Raspeig, Alicante, Spain*

Abstract

This paper presents a new fast algorithm for computing an approximation to the mean of two strings of characters representing a 2D shape and its application to a new Wilson-based editing procedure. The approximate mean is built up by including some symbols from the two original strings. In addition, a greedy approach to this algorithm is studied, which allows us to reduce the time required to compute an approximate mean. The new dataset editing scheme relaxes the criterion for deleting instances proposed by the *Wilson* editing procedure. In practice, not all instances misclassified by their near neighbors are pruned. Instead, an artificial instance is added to the dataset in the hope of successfully classifying the instance in the future. The new artificial instance is the approximated mean of the misclassified sample and its same-class nearest neighbor.

Experiments carried out over three widely known databases of contours show that the proposed algorithm performs very well when computing the mean of two strings, and outperforms methods proposed by other authors. In particular, the low computational time required by the heuristic approach makes it very suitable when dealing with long length strings. Results also show that the proposed preprocessing scheme can reduce the classification error in about 83% of trials. There is empirical evidence that using the greedy approximation to compute the approximated mean does not affect the performance of the editing procedure.

Key words: dataset editing, shape prototypes, edit distance, median string

1. Introduction

Dataset editing has received considerable attention since *Wilson's* seminal studies (Wilson, 1972) because this technique can improve the response of *nearest neighbor classifiers*. Mainly, these algorithms focus on deleting wrongly tagged instances from the training set for a given classifier. Several modifications have been proposed such as those by Tomek (1976a), Devijver and Kittler (1980), Ferri and Vidal (1992) Wilson

*Corresponding author. Fax:965909326
Email addresses: jose.abreu@umcc.cu (J. Abreu), juanra@dlsi.ua.es (J. R. Rico-Juan)

and Martínez (2000), and more recently Guan et al. (2009) and Vázquez et al. (2005), which encountered some of the problems of the basic *Wilson* procedure such as the statistical dependence of estimations over each instance (Penrod and Wagner, 1977). Another group of algorithms also changes some instance tags while editing, such as those proposed by Tomek (1976b) and Koplowitz and Brown (1981).

In many problems, patterns do not have a vectorial representation and another syntactic coding scheme such as strings and trees is commonly used. The methods cited above deal mainly with vectorial representations and distances; therefore, when dealing with strings a suitable distance needs to be selected. In our case, the widely used Levenshtein edit distance (Levenshtein, 1966) is chosen. In addition, some approaches such as that of Sánchez et al. (1997) find representative instances like centroids. Consequently, these concepts need to be extended to the new coding schemes.

Several authors such as Duta et al. (2001), Jian et al. (2000), Martínez and Casacubierta (2003) and Cardenas (2004) have considered the problem of computing the average of 2D shapes represented by strings. Most of these authors built up the desired string by successive refinements of an initial string or by some *ad-hoc* procedure, since (Casacuberta, 2000) demonstrated that the algorithm for finding the exact mean string from various samples is an NP hard problem. An approximate mean is often used instead of the exact one. For example (Jian et al., 2000) proposed a greedy algorithm to compute an approximation by iteratively adding symbols to an empty initial string. This operation uses the most suitable symbols defined using a goodness criterion. In (Jiang et al., 2003), the authors describe an algorithm for computing the mean of N strings based on the previously computed mean of $N-1$ strings and the weighted mean concept. Fisher and Zell (2000) present two methods to compute averages of strings by finding the most frequent edit operations when comparing a candidate with a set of strings. The i-esime symbol from the candidate is changed by applying the commonest operation at this place. Martínez and Casacubierta (2003) start with an initial string as the set mean, which is approximated toward the true mean by successive refinements applying *deletions*, *insertions* and *substitutions* over each position on the string in an attempt to decrease the accumulated distance. This procedure extensively explores the space of all possible strings to locate a good approximation to the exact mean string. These authors report different experiments to evaluate the approximate mean goodness. Although such approaches can lead to very good results, one immediate drawback is that the number of candidate strings grows cubically with the length of the strings.

There are some earlier studies which do not rely explicitly on the mean strings for computing the average of 2D shapes. In Bontempi and Marcelli (1995) contours are made up of basic elements (segments and arcs) and attributes such as curvature and direction. This information is encoded as a string that represents an individual in the population which evolves through a genetic algorithm to find a set of prototypes representing character shapes. In several studies (Duta et al., 1999a,b, 2001) the authors explore the construction of shape models encoded by a set of coordinates. Each contour is re-sampled to construct a polygonal approximation. This is aligned with respect to all the contours to find the polygon which minimizes the average distance between all objects. The shapes matching this polygon are removed from the set to form a cluster. Finally, a cluster prototype is defined as the Procrustes Average of the points extracted from the shapes. Sánchez et al. (2002) encode shapes by cyclic attributed strings which represent a polygonal approximation. To compute the mean shape each string is transformed in a piecewise linear function and then the mean is computed. The mean shape is constructed using a progressive procedure. For example, denoting the mean of two shapes S_i, S_j as $S^{i,j}$, to get the mean of shapes S_1, S_2, S_3, S_4, and S_5, first $S^{1,2}$ and $S^{3,4}$ are computed. From these strings we get $S^{1,2,3,4}$, which is finally combined with S_5 to obtain $S^{1,2,3,4,5}$ Cardenas (2004) describes an update rule to adjust a prototype by removing differences that are frequent with respect to the same class and infrequent with respect to different class closer strings.

More recently, other studies such as those by Platonov and Langer (2007) and Ong et al. (2008) also addressed the construction of prototypes representing a set of shapes exploring other representation formalisms. Many of these studies rely on a similarity criterion, mainly the Levenshtein distance when dealing with strings. However other distances, such as Dynamic Time Warping *(DTW)* were used in Marzal et al. (2006), Yu et al. (2007) and Yoon-Sik (2007). The latter studies report interesting results when *DTW* is used for shape comparison and retrieval. When the strings to be matched are not proper Freeman codes but sequences of points or polygons, the Levenshtein distance may have some drawbacks, as Marzal et al. (2006) pointed out. This distance is very sensitive to sample frequency along the contours since one-to-many correspondence of symbols from one string to another is not allowed.

In Abreu and Rico-Juan (2010) a *Wilson* based approach to edit a dataset of instances encoded by a string representation was described. The inclusion of a prototype representing both a misclassified instance and its same-class nearest neighbor inside a *k*-neighborhood if it exists, is the main difference with respect to others described in the literature. The authors also present an algorithm for computing the prototype representing two strings suitable for the requirements of the editing procedure.

This paper presents a greedy approximation to the algorithm for computing the prototype. This approach makes it possible to reduce the time required to compute the prototype. Also, new experiments were carried out to compare results when the editing procedure uses the greedy solution. Furthermore some comments relating to the size of the edited datasets are included. Section 2 provides a detailed explanation of the two algorithms used to build up the prototype and some useful concepts. Section 3 describes the proposed editing procedure and several considerations relating to the computational complexity of the proposed algorithms. Finally, section 4 illustrates the behavior of the proposed methods by means of various experiments.

2. Prototype construction

To compute the prototype representing two strings, in this case defined as an approximate mean string, the proposed approach focuses on information gathered by calculating the distance between the two strings. This section contains an explanation of how to calculate the selected distance measure, the Levenshtein (Levenshtein, 1966) distance, and a procedure to compute an approximate mean.

2.1. Edit Distance

Let Σ be an alphabet and $S_1 = \{S_{11}, S_{12}, \cdots, S_{1m}\}$, $S_2 = \{S_{21}, S_{22}, \cdots, S_{2n}\}$ two strings over Σ where $m, n \geq 0$, the edit distance between S_1 and S_2, $D(S_1, S_2)$, is defined in terms of elementary edit operations required to transform S_1 into S_2. Usually three edit operations are considered:

- *substitution* of a symbol $a \in S_1$ by a symbol $b \in S_2$, denoted as $w(a, b)$
- *insertion* of a symbol $b \in S_2$ in S_1, denoted as $w(\varepsilon, b)$
- *deletion* of a symbol $a \in S_1$, denoted as $w(a, \varepsilon)$.

where ε denotes an empty string. Let $Q_{S_i}^{S_j} = \{q_1, q_2, ..., q_k\}$ be a sequence of edit operations transforming S_i into S_j. If each operation has a cost $e(q_i)$ the cost of Q_{S2}^{S1} is $E_{Q_{S2}^{S1}} = \sum_{i=1}^{k} e(q_i)$ and the edit distance $D(S_1, S_2)$ is defined as:

$$D(S_1, S_2) = argmin_Q\{\ E_{Q_{S2}^{S1}}\}\ .$$

The strings involved in this paper are Freeman chain-codes and so substitution costs are computed as follows:

$$e(w(a, b)) = min\{|a - b|, 8 - |a - b|\}$$

In the case of the insertions and deletions, denoted as $e(w(\cdot, \varepsilon))$ and $e(w(\varepsilon, \cdot))$ respectively, a cost of 2 was chosen, which is half the maximum cost of the substitution operation; this same fixed number is used in Rico-Juan and Micó (2003). The algorithm described in Wagner and Fischer (1974) makes it possible to compute $D(S_1, S_2)$ in $O(L_{S1} \times L_{S2})$ time, where L_S denotes the length of string S. When it is not confusing $w(a, b)$, $w(a, \varepsilon)$ and $w(\varepsilon, b)$ will refer to the operation as well its cost.

2.2. Fast Approximate Mean String Computation

The mean of a set C_c of strings can be briefly defined as the string R which minimizes (1).

$$CumDist(R, C_c) = \sum D(R, S_i)|S_i \in C_c \qquad (1)$$

As explained above, the proposed approach computes an approximate mean R of two strings S_1 and S_2 by including some symbols from S_1 or S_2 in R. To choose whether or not to include a symbol, each operation in the minimum cost edit sequence Q_{S2}^{S1} is tested to see how it will affect $D(R, S_1)$ and $D(R, S_2)$, since the algorithm looks for a string R satisfying (1) and (2). This additional requirement specifies that an R nearly halfway between S_1 and S_2 will be preferred.

$$R = argmin_R\{|D(R, S_1) - D(R, S_2)|\} \qquad (2)$$

As explained in 2.1 deletions in Q_{S2}^{S1} involve a symbol from S_1, insertions a symbol from S_2, while substitutions relate one symbol from S_1 to another in S_2. The idea behind the algorithm is that each operation q_i in Q_{S1}^{S2} affects the future $D(R, S_1)$ and $D(R, S_2)$. For deletions and insertions the following cases hold:

The operation q_i is accepted:

- If q_i is a deletion then a symbol from S_1 will be included in R in order to make it more similar to S_1.

- q_i is an insertion, in this case a symbol from S_2 will be included in R, this time R tends to be similar to S_2.

If the operation is rejected:

- q_i is an insertion, in this case the symbol from S_2 will be excluded from R, then R remains similar to S_1.
- q_i is a deletion, then a symbol from S_1 will be excluded from R in order to make it resemble S_2.

A close examination of each possible operation helps to explain how accepting or rejecting the operation q_i affects $D(R, S_1)$ and $D(R, S_2)$.

For insertions, let b^k_{S2} be the k-esime symbol from S_2. An operation $q_i = w(\varepsilon, b^k_{S2})$ from $Q^{S2}_{S_1}$ indicates the insertion of this symbol into S_1 to obtain R. Suppose q_i is accepted, then it can be expected that Q^{S2}_R will not involve insertion of this symbol in R since it has already been done, so $D(R, S_2)$ does not change but $D(R, S_1)$ grows by $e(w(b^k_{S2}, \varepsilon))$, since the symbol needs to be deleted in this case. If the operation is rejected, i.e. b^k_{S2} is not included in R, then $D(R, S_2)$ increases by $e(w(\varepsilon, b^k_{S2}))$ since b^k_{S2} must be inserted in R to get S_2. The distance from R to S_1 will be not affected.

In turn, for accepted deletions $w(b^k_{S1}, \varepsilon)$, the symbol $b^k_{S_1}$ will be placed in R, thus $D(R, S_1)$ can be expected to be unchanged while $D(R, S_2)$ increases by $e(w(b^k_{S1}, \varepsilon))$. If the operation is rejected, $b^k_{S_1}$ will be excluded from R so $D(R, S_1)$ will grow by $e(w(\varepsilon, b^k_{S1}))$ while $D(R, S_2)$ remains unchanged.

Substitutions $w(b_{S_1}, b_{S_2})$ will always be accepted, but whenever possible a symbol m will be placed in R instead of b_{S_1} or b_{S_2}. The choice of m attempts to make R similar to both S_1 and S_2, thus it must satisfy:

$$e(w(b_{S_1}, b_{S_2})) = e(w(m, b_{S_1})) + e(w(m, b_{S_2})). \tag{3}$$

$$m = argmin_m\{|e(w(m, b_{S_1})) - e(w(m, b_{S_2}))|\}. \tag{4}$$

Substitutions make both $D(R, S_1)$ and $D(R, S_2)$ grow by $e(w(b_{S1}, m))$ and $e(w(m, b_{S2}))$ respectively.

For example, let $S_1 = \{2, 3, 4\}$, $S_2 = \{6, 0\}$ with a *substitution* cost as defined in section 2.1 but fixing a cost of 1 for *insertions* and *deletions*. Thus $Q^{S2}_{S1} = \{w(2, \varepsilon), w(3, \varepsilon), w(4, 6), w(\varepsilon, 0)\}$. Possible select/reject options yield the tree in Fig. 1 where each leaf node shows a candidate R including the symbols involved in the *accepted* operations from the corresponding branch. Inside brackets, the cumulative contribution of each operation up from the node to $D(R, S_1)$ and $D(R, S_2)$ respectively is shown. For example, string $R = \{2, 3, 5, 0\}$ in the leftmost branch results from accepting $\{w(2, \varepsilon), w(2, \varepsilon), w(4, 6), w(\varepsilon, 0)\}$. In this case $D(R, S_1) = 2$ and $D(R, S_2) = 3$. The procedures *FMSC* and *FindOp* outlined below allow us to search through the tree for the edit operations which yield an R satisfying the established requirements.

Function FindOp(op_i,a_{S1},a_{S2}) : $\langle d, r \rangle$

```
/* $Q_{S1}^{S2}$: minimum cost edit sequence to transform $S_1$ into $S_2$ */
/* $r$: string resulting from applying or not to $S_1$ edit operations in $Q_{S1}^{S2}$ */
/* $op_i$: $i$-esime edit operation $op \in Q_{S1}^{S2}$ */
/* $a_{S1} = 0$ and $a_{S2} = 0$ : cumulative distance $D(r, S1)$ and $D(r, S2)$ respectively */
/* $d$: difference between $a_{S1}$ and $a_{S2}$ */
/* $better = \langle \infty, \emptyset \rangle$:better partial result, $better[0]$ distance, $better[1]$ respective $r$ */
if $op_i == 0$ then $better \leftarrow \langle a_{S1} - a_{S2}, \infty \rangle$ else
    switch $Q_{S1}^{S2}[op_i]$ do
        case $w(b_{S1}, \varepsilon, )$:                                    /* Deletion */
            $\langle d_{no}, r_{no} \rangle \leftarrow FindOp(op_i - 1, a_{S1} + w(\varepsilon, b_{S1}), a_{S2})$ /* Rejected */
            $\langle d_{yes}, r_{yes} \rangle \leftarrow FindOp(op_i - 1, a_{S1}, a_{S2} + w(b_{S1}, \varepsilon))$ /* Accepted */
            if $|d_{yes}| < |d_{no}|$ then $better \leftarrow \langle d_{yes}, r_{yes} \cup \{b_{S1}\} \rangle$ else
                $better \leftarrow \langle d_{no}, r_{no} \rangle$
            end if
        case $w(\varepsilon, b_{S2})$:                                      /* Insertion */
            $\langle d_{no}, r_{no} \rangle \leftarrow FindOp(op_i - 1, a_{S1}, a_{S2} + w(\varepsilon, b_{S2}))$ /* Rejected */
            $\langle d_{yes}, r_{yes} \rangle \leftarrow FindOp(op_i - 1, a_{S1} + w(b_{S2}, \varepsilon), a_{S2})$ /* Accepted */
            if $|d_{yes}| < |d_{no}|$ then $better \leftarrow \langle d_{yes}, r_{yes} \cup \{b_{S2}\} \rangle$ else
                $better \leftarrow \langle d_{no}, r_{no} \rangle$;
            end if
        case $w(b_{S1}, b_{S2})$:                                          /* Substitution */
            foreach symbol m satisfying (3) and (4) do
                $\langle d, r \rangle \leftarrow FindOp(op_i - 1, a_{S1} + w(m, b_{S1}), a_{S2} + w(m, b_{S2}))$;
                if $|d| < |better[0]|$ then $better \leftarrow \langle d, r \cup \{m\} \rangle$;
            end foreach

    endsw
end if
return $better$
```

Function FMSC(S_1, S_2) :R

```
/* $S_1$ and $S_2$: strings to compute its approximate mean $R$ */
compute $D(S_1, D_2)$ to get $Q_{S1}^{S2}$;
$\langle d, R \rangle \leftarrow$ FindOp($L_{Q_{S1}^{S2}}$, 0, 0);
return $R$;
```

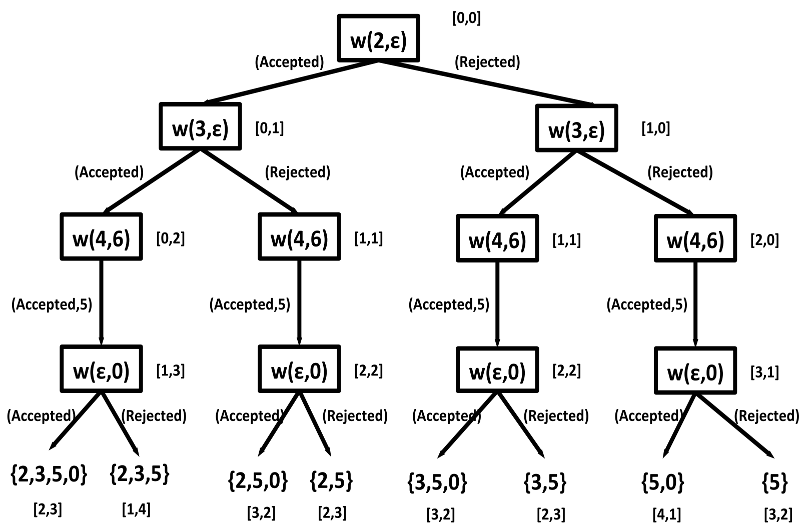


Figure 1: Each branch represents a set of *accept* or *reject* edit operations to get the string R in the leaf node from S_1. Values for $D(R, S_1)$ and $D(R, S_2)$ are given in brackets.

2.3. Heuristic Approach to Fast Computing to an Approximate Mean String.

As explained above, this paper proposes a heuristic approach to the previously commented algorithm. This implementation focuses on reducing the number of explored tree branches by discarding some alternatives. Each time at most two branches B_1 and B_2 are kept alive. The tree is explored breadth-wise first. Again, either a *substitution* or a *deletion* splits a branch into two new ones corresponding to the accept/reject options. This leads to four branches, two of which need to be pruned. The branch leading to better partial distances $D(R, S_1)$ and $D(R, S_2)$ is one of those selected to stay alive. The other surviving branch is the one corresponding to the contrary decision on the opposite branch of the tree. For example, suppose the better alternative is a reject decision over B_1, so the other surviving branch is the one corresponding to an accept decision over B_2. With regard to processing substitutions, no changes were introduced. Fig. 2 helps to clarify this situation graphically.

3. Editing Algorithm

Let T be a set of instances. *Wilson* (Wilson, 1972) based editing procedures such as those described by Tomek (1976a) and Ferri and Vidal (1992) remove all instances t_i misclassified by their k-nearest neighbors (K-NN). This kind of editing cleans interclass overlapping regions while the boundaries between classes are smoothed. A K-NN classifier that uses the edited set as a training set could improve its classification results compared with those using the original dataset.

```
Function FMSC-Greedy($S_1, S_2$):r
  /* $S_1$ y $S_2$:  strings to compute its approximate mean $R$                         */
  /* $Q_{S1}^{S2}$:  minimum cost edit sequence to transform $S_1$ into $S_2$            */
  /* $op_i$:  $i$-esime edit operation $op \in Q_{S1}^{S2}$                              */
  /* $a_{S1}^{B1}, a_{S2}^{B1}$:  respective cumulative distances $D(R,S_1)$ and $D(R,S_2)$, branch 1   */
  /* $a_{S1}^{B2}, a_{S2}^{B2}$:  respective cumulative distances $D(R,S_1)$ and $D(R,S_2)$, branch 2   */
  foreach $op_i \in Q_{S1}^{S2}$ do
    switch $Q_{S1}^{S2}[op_i]$ do
      case $w(b_{S1}, \varepsilon)$:                                               /* Deletion */
        $\langle d_{S1}^{B1}, d_{S2}^{B1}\rangle \leftarrow \langle a_{S1}^{B1} + w(\varepsilon, b_{S1}), a_{S2}^{B1}\rangle$;      /* Reject, branch 1 */
        $\langle c_{S1}^{B1}, c_{S2}^{B1}\rangle \leftarrow \langle a_{S1}^{B1}, a_{S2}^{B1} + w(b_{S1}, \varepsilon)\rangle$;      /* Accept, branch 1 */
        $\langle d_{S1}^{B2}, d_{S2}^{B2}\rangle \leftarrow \langle a_{S1}^{B2} + w(\varepsilon, b_{S1}), a_{S2}^{B2}\rangle$;      /* Reject, branch 1 */
        $\langle c_{S1}^{B2}, c_{S2}^{B2}\rangle \leftarrow \langle a_{S1}^{B2}, a_{S2}^{B2} + w(b_{S1}, \varepsilon)\rangle$;      /* Accept, branch 1 */
        UpdateDistances $(\langle d_{S1}^{B1}, d_{S2}^{B1}\rangle, \langle c_{S1}^{B2}, c_{S2}^{B2}\rangle, \langle d_{S1}^{B1}, d_{S2}^{B1}\rangle, \langle c_{S1}^{B1}, c_{S2}^{B1}\rangle)$;

      case $w(\varepsilon, b_{S2})$:                                              /* Insertion */
        $\langle d_{S1}^{B1}, d_{S2}^{B1}\rangle \leftarrow \langle a_{S1}^{B1}, a_{S2}^{B1} + w(\varepsilon, b_{S2})\rangle$;      /* Reject, branch 1 */
        $\langle c_{S1}^{B1}, c_{S1}^{B1}\rangle \leftarrow \langle a_{S1}^{B1} + w(b_{S2}, \varepsilon), a_{S2}^{B1}\rangle$;      /* Accept, branch 1 */
        $\langle d_{S1}^{B2}, d_{S2}^{B2}\rangle \leftarrow \langle a_{S1}^{B2}, a_{S2}^{B2} + w(\varepsilon, b_{S2})\rangle$;      /* Reject, branch 2 */
        $\langle c_{S1}^{B2}, c_{S2}^{B2}\rangle \leftarrow \langle a_{S1}^{B2} + w(b_{S2}, \varepsilon), a_{S2}^{B2}\rangle$;      /* Accept, branch 2 */
        UpdateDistances $(\langle d_{S1}^{B1}, d_{S2}^{B1}\rangle, \langle c_{S1}^{B2}, c_{S2}^{B2}\rangle, \langle d_{S1}^{B1}, d_{S2}^{B1}\rangle, \langle c_{S1}^{B1}, c_{S2}^{B1}\rangle)$;

      case $w(b_{S1}, b_{S2})$:                                                    /* Substitution */
        select the symbol $m$ that better satisfies the equations (3) and (4);
        $\langle a_{S1}^{B1}, a_{S2}^{B1}\rangle \leftarrow \langle a_{S1}^{B1} + w(m, b_{S1}, a_{S2}^{B1} + w(m, b_{S2})))$;
        $\langle a_{S1}^{B2}, a_{S2}^{B2}\rangle \leftarrow \langle a_{S1}^{B2} + w(m, b_{S1}, a_{S2}^{B2} + w(m, b_{S2}))\rangle$;

    endsw
  end foreach
  select the better from $\langle a_{S1}^{B1}, a_{S2}^{B1}\rangle$ and $\langle a_{S1}^{B2}, a_{S2}^{B2}\rangle$;
  return $R$, including symbols involved in the accepted operations from the better branch.
```

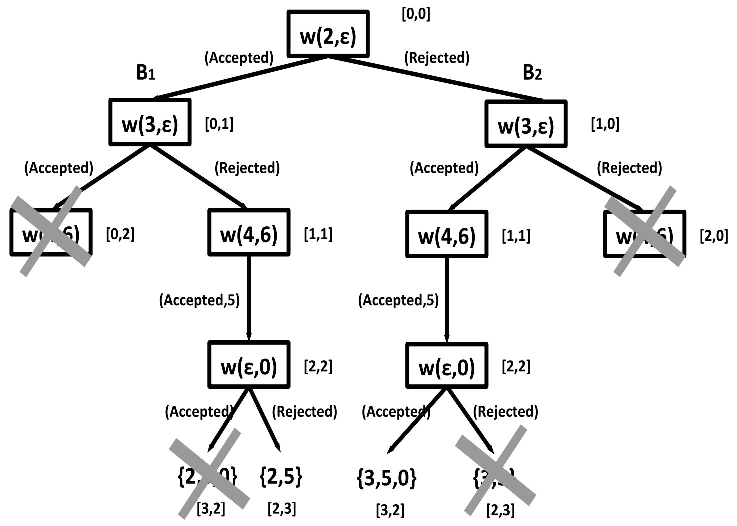


Figure 2: Heuristic approach example where crossed nodes represent pruned branches at each level.

As Wilson and Martínez (2000) point out, in some cases editing needs to be done carefully because the algorithm may remove a lot of instances and spoil the generalization capabilities. When $k \geq 1$, a wrong classification of t_i does not mean that no one k-nearest neighbor belongs to the same class as t_i. Thus, it is reasonable to assume that t_i is not necessarily an outlier, but can be a boundary instance useful for subsequent classifications. In other cases, due to some factors such as the high dimensionality of the data (Ferri et al., 1999), every instance is near to examples belonging to other classes. This means that any reduction in the original set increases the average classification error.

The proposed approach aims to deal successfully with this problem by adding to T an artificial instance R computed from t_i and its same-class nearest neighbor, t_j, if it belongs to k-nearest neighbors. R, tagged as t_i, satisfies $D(R, t_i) \leq D(t_i, t_j)$ and (2). Its inclusion boosts the chance t_i will be correctly reclassified since by definition R lies in the t_i k-neighborhood. Moreover, this means that some poorly covered regions may be better represented. From these assumptions it may de deduced that this editing scheme leads to fewer classification errors than when the original dataset is used. The artificial instance R will be computed by the **F**ast **M**ean **S**tring **C**omputation procedure, *FMSC* for short, described in section (2.2), as $R = FMSC(t_i, t_j)$. One of the purposes of this paper is to compare how results using *Greedy-FMSC* vary from those using *FMSC*. The algorithm sketched below allows the edited set to be computed.

3.1. Computational Cost Analysis

Let L_S be the length of the string S, L the length of the longest chain-code in a set of $N \geq 2$ shapes and $L_{Q_{Si}^{Sj}} = L_{Si} + L_{Sj} \leq 2L$ for the worst case, i.e. when there are

```
Procedure UpdateDistances($\langle d^{B1}_{S1}, d^{B1}_{S2}\rangle, \langle c^{B2}_{S1}, c^{B2}_{S2}\rangle, \langle d^{B1}_{S1}, d^{B1}_{S2}\rangle, \langle c^{B1}_{S1}, c^{B1}_{S2}\rangle$)
$\langle t_{S1}, t_{S2}\rangle \leftarrow$ better from $\{\langle d^{B1}_{S1}, d^{B1}_{S2}\rangle, \langle c^{B2}_{S1}, c^{B2}_{S2}\rangle, \langle d^{B1}_{S1}, d^{B1}_{S2}\rangle, \langle c^{B1}_{S1}, c^{B1}_{S2}\rangle\}$;
switch $\langle t_{S1}, t_{S2}\rangle$ do
    case $\langle d^{B1}_{S1}, d^{B1}_{S2}\rangle$:
        $\langle a^{B1}_{S1}, a^{B1}_{S2}\rangle \leftarrow \langle d^{B1}_{S1}, d^{B1}_{S2}\rangle$;
        $\langle a^{B2}_{S1}, a^{B2}_{S2}\rangle \leftarrow \langle c^{B2}_{S1}, c^{B2}_{S2}\rangle$;

    case $\langle c^{B1}_{S1}, c^{B1}_{S2}\rangle$:
        $\langle a^{B1}_{S1}, a^{B1}_{S2}\rangle \leftarrow \langle c^{B1}_{S1}, c^{B1}_{S2}\rangle$;
        $\langle a^{B2}_{S1}, a^{B2}_{S2}\rangle \leftarrow \langle d^{B2}_{S1}, d^{B2}_{S2}\rangle$;

    case $\langle d^{B2}_{S1}, d^{B2}_{S2}\rangle$:
        $\langle a^{B1}_{S1}, a^{B1}_{S2}\rangle \leftarrow \langle c^{B1}_{S1}, c^{B1}_{S2}\rangle$;
        $\langle a^{B2}_{S1}, a^{B2}_{S2}\rangle \leftarrow \langle d^{B2}_{S1}, d^{B2}_{S2}\rangle$;

    case $\langle c^{B2}_{S1}, c^{B2}_{S2}\rangle$:
        $\langle a^{B1}_{S1}, a^{B1}_{S2}\rangle \leftarrow \langle d^{B1}_{S1}, d^{B1}_{S2}\rangle$;
        $\langle a^{B2}_{S1}, a^{B2}_{S2}\rangle \leftarrow \langle c^{B2}_{S1}, c^{B2}_{S2}\rangle$;

endsw
```

```
Function JWilson(T,K) :$T$
/* $T$:  instance set to edit.                                   */
/* $K$:  number of near neighbors.                               */
foreach instance $t_i \in T$ do
    classify $t_i$ by its $k$-nearest neighbors in $T - t_i$;
    if $t_i$ is wrongly classified then
        find $t_j$, the $k$-nearest same-class neighbor of $t_i$;
        if $t_j$ exists then
            build $R = FMSC(t_i, t_j)$ or $R = FMSC - Greedy(t_i, t_j)$;
            make $T = T \cup R$;
        else
            mark $t_i$ to deletion;
        end if
    end if
end foreach
delete from $T$ all marked instances;
return $T$;
```

no substitutions, the length of the minimum cost edit sequence to transform S_i into S_j. Computing an approximate mean R from two strings S_1 and S_2 involves calculating $D(S_1, S_2)$, which can be accomplished in $O(L_{S1} \times L_{S2})$ as pointed out in subsection 2.1 or more simply, in $O(L^2)$.

Both *FMSC* and *FMSC-Greedy* involve computing $D(S_1, D_2)$. Searching through the tree with *FindOp* can be viewed as evaluating all possibilities of assigning $\{accepted/rejected\}$ to every q_i in Q_{S1}^{S2}, so there are 2^{2L} chances, that is, the maximum number of branches on the tree. Thus *FindOp* requires a first step, computing the distance, with cost $O(L^2)$ followed by another block with cost $O(2^{2L})$. By the *sum rule* (Aho et al, 1983) this leads to $O(2^{2L})$ time. Logically in practice this value is lower since *substitutions* do not split a branch in two new ones.

Regarding *FMSC-Greedy*, the **foreach** loop needs to examine all $L_{Q_{S1}^{S2}}$ operations in Q_{S1}^{S2} but just once. This can be accomplished in $O(2L)$. Also the cost of computing the distance from S_1 to S_2 needs to be considered. Again by the *sum rule* the cost for *FMSC-Greedy* is $O(L^2)$.

The editing procedure needs to classify every instance in T, which requires $O(|T|^2)$ time. A second step involves computing *FMSC* for every wrongly classified instance having a same-class k-nearest neighbor. In the worst case, all $|T|$ instances will need to be processed, so this step entails $O(|T| \times FMSC)$ time or $O(|T| \times FMSC\text{-}Greedy)$ for the greedy approach.

4. Experimental Results

Experiments were carried out to evaluate the performance of *FMSC*, *FMSC-Greedy* and *JWilson* in their respective tasks.

A first test experiment was done to compare *FMSC* and *FMSC-Greedy* when computing the average of two shapes. These methods were also compared to the one proposed by (Martínez and Casacubierta, 2003) initialized with the set mean and with a greedy approximate mean, labelled as *MH(Set mean)* and *MH(Greedy)* respectively. Since the algorithm in (Martínez and Casacubierta, 2003) only focuses on finding an approximate mean satisfying (1), a modification was introduced to check also the restriction in (2).

To ensure the results were independent of the database, the class and the string length data were randomly collected from three widely known contour datasets. The Special Database 3 of the National Institute of Standards and Technology, the US Post Service digit recognition corpus (UPS) and the MPEG7 CE Shape-1 Part B with 26, 10 and 11 different classes respectively. The length of the strings coding the 2D shapes can be seen to vary greatly. The average contour length was 724 with standard deviation of 844. The longest and the shortest chain-codes have length of 3431 and 41 respectively. All pair-wise averages were then computed even if the shapes belonged to different classes.

To evaluate how well the algorithms behave with respect to (1) and (2), these values were computed. In terms of (1) no differences were found for *FMSC-Greedy*, *FMSC* and Martínez and Casacubierta (2003) when initialized with the set mean since for each pair-wise average the computed approximate mean had the same value for (1).

When a greedy string was set as the initial instance for the approach in (Martínez and Casacubierta, 2003), the worst results were obtained.

Table 1 shows that *FMSC* and *FMSC-Greedy* behave very well in terms of (2), i.e. the average shape computed is in the ”middle” of the two original shapes. *MH(SetMean)* also performs well, but the high standard deviation indicates that in some cases the computed approximate mean is biased towards one of the strings. Again MH(Greedy) achieves the worst results.

It is remarkable that *FMSC-Greedy* is as effective as *FMSC* and (Martínez and Casacubierta, 2003), but requires significantly less computational power. For example every time a candidate string is modified to test if it improves the approximate mean, two distances need to be computed. In this case *MH(SetMean)* needs to compute about 43218.2 distances on average, *MH(Greedy)* a total of 73143.3, whereas *FMSC-Greedy* requires only one. Although theoretically *FMSC-Greedy* is less time consuming than *FMSC*, an empirical comparison was done. On average, the *Greedy* approach was about 16 times faster.

Table 1: Mean value of $|D(R, S_1) - D(R, S_2)|$ for all pair-wise averages.

Algorithm	Avg.±Stdv
FMSC	0.5 ± 0.5
FMSC-Greedy	0.9 ± 0.7
MH(Greedy)	3.3 ± 44.4
MH(SetMean)	0.7 ± 5.3

4.1. Evaluation of the editing procedure.

A second experiment was done to evaluate the performance of the proposed editing procedure and determine how using *FMSC-Greedy* instead of *FMSC* affects performance of *JWilson*. A comparison with the classic *Wilson* procedure was also made. A sample of 80 instances per class was drawn from the NIST-3 Uppercase Letters and the USPS Digits datasets. MPEG-7 was discarded in this experiment since there are only 20 instances available per class. Each set was split for a 4-fold cross-validation technique.

Initially all training sets were edited by the *Wilson* procedure and each test set classified by the K-NN rule using the respective edited set. This editing-classification experiment was repeated twice but using the proposed editing scheme: first, computing the prototypes using the *FMSC* approach and finally using *FMSC-Greedy*. As a baseline, the original training set classification was used. In each fold, editing was repeated for odd values of K from 3 to 17, while in the classification stage, the range was from 1 to 17. The weighted edit distance, described in section 2.1 was used as the distance measure.

Tables 2 and 3 show the 4-fold average classification error and the *standard deviation* for all K values tested at the editing and classification stages of the experiments. In none of these experiments did the *Wilson* procedure reduce the baseline error rate (classification with original training sets). This result improved when editing was done using our proposed algorithm and the *FMSC* procedure. This approach reduced the baseline error in 87.5% of the trials for the character dataset and in 79.2% in the case of the digit dataset. These improvements are highlighted in bold type in the tables of results. Similar results were obtained when the greedy implementation was used. Previous results show that, on average, the proposed algorithm for editing outperforms the *Wilson* approach as regards error rate reduction. Heat maps in Fig. 3 visually represent the data in tables (2) and (3) and also include the respective *standard deviation*. Better results are obtained when editing is done taking high values for K and classification is done taking low values for K. As was expected, there were no remarkable differences in performance of the editing procedure depending on the algorithm selected to compute the artificial prototype. An example of this can be seen in the graphics in Fig. 4, where a significant improvement may be seen in the case of classification. This illustrates the 4-fold average error rate and the standard deviation when classification is done with $K = 1$ and $K = 17$, taking data sets edited by different values of K. For other values of K in the classification, results are very similar.

Another interesting subject to analyze is the number of instances in the edited set. The *Wilson* procedure removes any wrongly classified instances and so the size of the edited set will never increase. The proposed editing scheme may not erase all instances tagged to be deleted by Wilson; moreover, new prototypes may be added to the data set. In all the editing experiments the size of the edited set appears to grow like a linear factor bounded by K. This is, in all cases $S_{ES} \leq S_{OS} + \frac{K*S_{OS}}{100}$, where S_{OS} and S_{ES} are the size of the original and the edited set respectively.

Table 2: Average error rate (4-fold) as a percentage for classification with different edited sets in letter database.

			K on Classification								
			1	3	5	7	9	11	13	15	17
K on Edition	Not edited		13.7±1.4	14.7±1.9	15.4±1.5	16±1	17.1±1.1	17.7±1.7	18±2	18.6±1.8	19.6±1.8
	K=3	Wilson	17±2	17.6±1.8	17.6±1.0	19.4±1.4	19.5±1.8	20±2	21.0±1.8	22.0±1.5	22.1±1.6
		Dynamic	14.5±1.9	15.5±1.7	**15.3±1.4**	16.6±1.1	**16.4±1.9**	18±2	18±2	19±2	**19.0±1.9**
		Greedy	14.4±1.8	15±2	**15.3±1.9**	16.2±1.5	**16±2**	18±3	19±2	19±2	**19±2**
	K=5	Wilson	15.8±1.7	17.6±1.7	17.9±1.3	19.6±1.4	20.1±1.7	21±2	21±2	21.6±1.5	22.7±1.7
		Dynamic	14.0±1.6	**14.5±1.8**	**14.5±1.7**	**15.6±1.1**	**16.0±1.8**	**16.3±1.7**	**17.5±1.6**	**17.7±1.7**	**18.4±1.9**
		Greedy	13.8±1.9	15±3	**14.7±1.9**	**15.0±1.2**	**15.5±1.8**	**16.1±1.5**	**17.1±1.4**	**18.1±1.8**	**18±2**
	K=7	Wilson	16.3±1.9	17.5±1.2	17.9±1.3	19.9±1.6	20±2	21±2	22±2	22.3±1.4	23.0±1.4
		Dynamic	13.9±1.6	**14.4±1.7**	**14.2±1.6**	**15.3±1.0**	**15.6±1.3**	**16.3±1.0**	**16.6±1.4**	**17.3±1.4**	**17.8±1.7**
		Greedy	13.9±1.7	**14±2**	**14.1±1.3**	**15.1±0.5**	**15.0±1.1**	**15.9±1.4**	**16.4±1.2**	**17.1±1.8**	**18±2**
	K=9	Wilson	17±2	17.8±1.7	18.5±1.8	19.8±1.9	21±2	21±3	22±2	22.6±1.6	23.8±1.6
		Dynamic	13.8±1.4	**14.2±1.3**	**14.0±1.5**	**15.0±0.8**	**15.4±0.9**	**16.2±0.9**	**16.8±0.9**	**17.1±0.8**	**17.6±1.2**
		Greedy	13.7±1.4	**14.0±1.7**	**14.0±0.9**	**14.7±0.2**	**14.3±1.0**	**15.8±1.1**	**16.4±0.9**	**16.4±1.5**	**17.2±1.8**
	K=11	Wilson	17.1±1.9	18.8±1.7	18.9±1.6	20±2	21±2	22±2	23±2	23.3±1.4	24.0±1.4
		Dynamic	**13.6±1.4**	**14.0±1.6**	**14.1±1.6**	**14.9±0.7**	**15.2±1.0**	**15.6±1.0**	**16.3±1.1**	**16.5±0.8**	**17.3±1.2**
		Greedy	**13.5±1.4**	**14.0±1.5**	**13.8±1.2**	**14.5±0.5**	**14.3±1.2**	**15.4±1.1**	**15.8±1.0**	**15.9±1.5**	**16.9±1.6**
	K=13	Wilson	17.1±1.7	18.9±1.3	19.5±1.4	21±2	22±2	22±2	22.8±1.6	23.5±1.5	24.0±1.5
		Dynamic	**13.6±1.3**	**14.2±1.5**	**14.1±1.5**	**14.6±1.0**	**15.1±1.0**	**15.6±1.1**	**16.3±1.1**	**16.6±1.2**	**17.2±1.2**
		Greedy	**13.5±1.3**	**14.3±1.7**	**13.9±1.3**	**14.3±0.7**	**14.6±1.2**	**15.3±1.5**	**15.5±1.0**	**16.1±1.3**	**17±2**
	K=15	Wilson	17.5±1.8	19.6±1.6	19.8±1.4	20.8±1.8	22±2	22±2	23.3±1.6	23.7±1.6	24.6±1.4
		Dynamic	**13.4±1.3**	**13.9±1.3**	**13.8±1.4**	**14.7±1.0**	**15.3±1.2**	**15.6±1.2**	**16.2±0.9**	**16.5±1.4**	**17.0±1.4**
		Greedy	**13.3±1.3**	**13.9±1.4**	**13.7±1.3**	**14.1±0.5**	**14.1±0.9**	**15.2±1.3**	**15.3±0.9**	**16.1±1.7**	**16.6±1.7**
	K=17	Wilson	17.8±1.5	19.9±1.2	20.2±1.2	21.3±1.5	22.2±1.6	23.1±1.7	23.7±1.9	24.2±1.3	24.8±1.3
		Dynamic	**13.5±1.2**	**14.0±1.1**	**13.9±1.4**	**14.8±1.0**	**15.3±0.9**	**15.7±0.9**	**16.0±1.0**	**16.3±0.9**	**16.6±1.5**
		Greedy	**13.3±1.1**	**13.9±1.0**	**13.8±1.1**	**14.5±0.4**	**14.2±0.7**	**15.0±0.8**	**15.3±1.0**	**15.9±1.4**	**16.3±1.5**

Table 3: Average error rate (4-fold) as a percentage for classification with different edited sets in digits database.

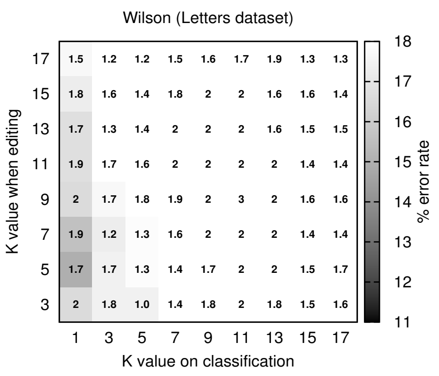
			K on Classification								
			1	3	5	7	9	11	13	15	17
K on Edition	Not edited		1.8±1.3	2.0±0.7	3.0±0.4	3.5±0.4	3.6±0.3	4.1±0.5	4.4±0.9	4.8±1.3	4.9±0.9
	K=3	Wilson	2.8±0.9	3.1±0.8	3.6±0.3	4.3±0.9	4.3±0.9	4.5±0.8	4.8±0.6	5.1±1.1	5.1±0.9
		Dynamic	2.0±1.1	2.3±0.5	**2.9±0.3**	**3.4±0.5**	**3.5±0.7**	**3.9±0.5**	**4.1±0.8**	**4.5±1.3**	**4.6±0.5**
		Greedy	1.9±1.0	2.3±0.5	**2.9±0.3**	**3.3±0.3**	**3.5±0.7**	**3.9±0.5**	**4.3±0.6**	**4.6±1.4**	**4.8±0.6**
	K=5	Wilson	2.6±0.9	2.9±0.6	3.6±0.3	4.3±0.6	4.1±0.6	4.5±0.8	4.8±0.6	5.1±1.1	5.1±0.9
		Dynamic	1.9±1.1	2.4±0.6	**2.8±0.3**	**3.0±0.4**	**3.1±0.3**	**3.8±0.5**	**3.9±0.5**	**4.3±1.4**	**4.4±0.6**
		Greedy	1.8±1.0	2.3±0.5	**2.8±0.3**	**3.0±0.4**	**3.1±0.3**	**3.5±0.4**	**3.8±0.3**	**4.3±1.4**	**4.4±1.0**
	K=7	Wilson	2.5±1.1	3.0±0.7	3.8±0.5	4.3±0.9	4.3±0.9	4.6±0.9	5.0±0.9	5.4±1.3	5.4±0.6
		Dynamic	1.8±1.0	2.1±0.6	**2.5±0.6**	**2.9±0.8**	**3.1±0.5**	**3.6±0.6**	**3.9±0.5**	**4.1±1.3**	**4.3±0.6**
		Greedy	**1.6±1.0**	2.0±0.4	**2.5±0.6**	**2.9±0.8**	**3.1±0.5**	**3.4±0.6**	**3.8±0.3**	**4.0±1.2**	**4.3±1.0**
	K=9	Wilson	2.9±1.1	3.3±0.9	4.3±0.3	4.5±0.7	4.6±0.6	4.6±0.8	5.0±0.9	5.5±1.1	5.5±0.6
		Dynamic	1.8±1.0	2.0±0.4	**2.5±0.6**	**2.6±0.5**	**3.0±0.4**	**3.4±0.3**	**3.5±0.7**	**4.0±1.1**	**4.1±0.9**
		Greedy	**1.6±1.0**	**1.9±0.3**	**2.5±0.6**	**2.6±0.5**	**2.9±0.5**	**3.3±0.5**	**3.5±0.7**	**3.8±0.9**	**4.1±1.3**
	K=11	Wilson	2.8±1.0	3.4±0.9	4.3±0.3	4.6±0.6	4.9±0.8	4.6±0.8	5.3±1.0	5.5±1.1	5.5±0.9
		Dynamic	1.8±1.0	2.1±0.6	**2.5±0.6**	**2.6±0.5**	**2.9±0.6**	**3.0±0.7**	**3.3±0.5**	**3.9±0.9**	**4.0±0.9**
		Greedy	**1.6±1.0**	2.0±0.4	**2.5±0.6**	**2.6±0.5**	**2.8±0.6**	**2.9±0.9**	**3.3±0.5**	**3.6±0.8**	**4.0±1.3**
	K=13	Wilson	2.8±0.9	3.4±0.6	4.1±0.5	4.8±0.3	4.9±0.8	4.8±0.9	5.6±1.4	6.1±1.5	6.1±1.4
		Dynamic	1.8±1.0	2.1±0.6	**2.4±0.5**	**2.6±0.5**	**2.9±0.6**	**3.0±0.7**	**3.3±0.5**	**3.9±0.9**	**4.0±0.9**
		Greedy	**1.6±1.0**	2.0±0.4	**2.4±0.5**	**2.6±0.5**	**2.8±0.6**	**2.9±0.9**	**3.3±0.5**	**3.8±0.9**	**4.0±1.3**
	K=15	Wilson	2.8±0.9	3.8±0.9	4.3±0.6	5.0±0.7	5.1±0.9	4.9±0.8	5.9±1.3	6.3±1.3	6.0±1.6
		Dynamic	1.8±1.0	2.1±0.6	**2.4±0.5**	**2.6±0.5**	**2.9±0.6**	**3.1±0.5**	**3.3±0.5**	**3.6±0.9**	**3.9±1.1**
		Greedy	**1.6±1.0**	2.0±0.4	**2.4±0.5**	**2.6±0.5**	**2.9±0.9**	**2.9±0.9**	**3.3±0.5**	**3.5±0.7**	**3.9±1.1**
	K=17	Wilson	2.8±0.9	3.9±1.0	4.4±0.8	5.1±0.6	5.4±0.8	5.3±0.5	5.9±1.0	6.3±1.3	6.1±1.4
		Dynamic	**1.6±1.0**	2.0±0.7	**2.4±0.9**	**2.6±0.5**	**3.0±0.8**	**3.3±0.6**	**3.3±0.5**	**3.8±1.0**	**3.9±1.1**
		Greedy	**1.6±1.0**	2.0±0.4	**2.4±0.5**	**2.6±0.5**	**2.9±0.9**	**3.0±1.1**	**3.3±0.5**	**3.6±0.9**	**3.9±1.1**

5. Conclusions and Future Work

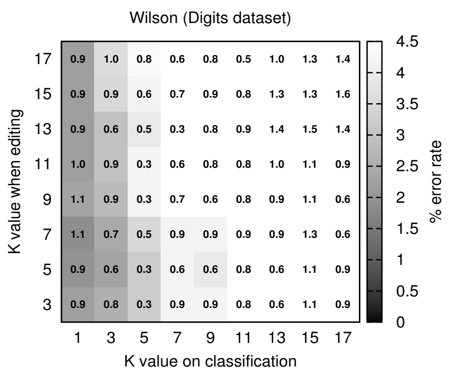
A new method was presented for editing a dataset of contours encoded by Freeman chain-codes. As the experimental results show, adding artificial prototypes to populate regions close to instances with a few same-class near neighbors contributes to reducing the classification error. In addition a new method, *FMSC*, was presented for computing the average between two shapes represented as Freeman chain codes. The average shape is computed as an approximated mean string of the chain codes representing the shapes. Another approach, *FMSC-Greedy*, a greedy approximation of the former, was also described. Experiments show that when computing the average of two shapes both methods behave very well and achieve similar results to other approaches described in the literature while, in the case of *FMSC-Greedy*, significantly reducing the computational time required. Since at present both methods can handle only two instances at a time, further investigations may be done to expand on the idea behind the algorithm between two examples to extend it to $N > 2$ instances.

Acknowledgments

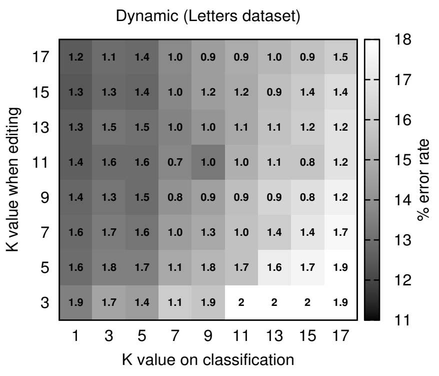
This work is partially supported by the Spanish CICYT under project DPI2006-15542-C04-01, the Spanish MICINN through project TIN2009-14205-CO4-01 and by the Spanish research program Consolider Ingenio 2010: MIPRCV (CSD2007-00018).



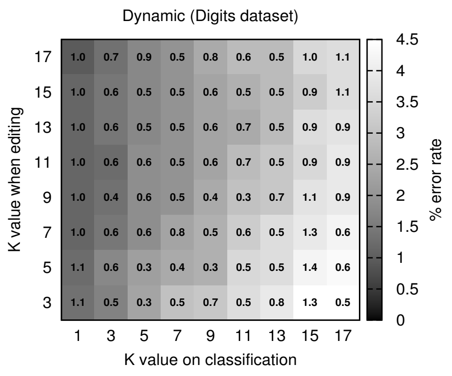
(a)



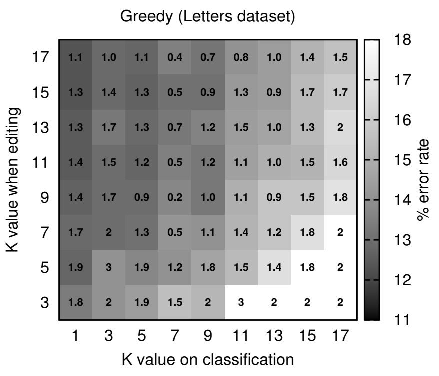
(b)



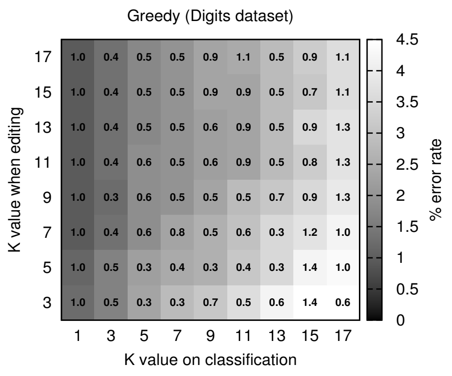
(c)



(d)

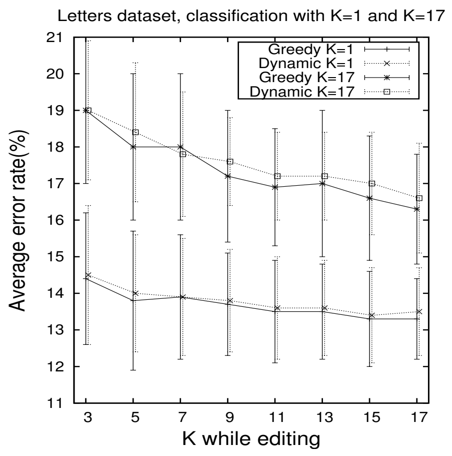


(e)

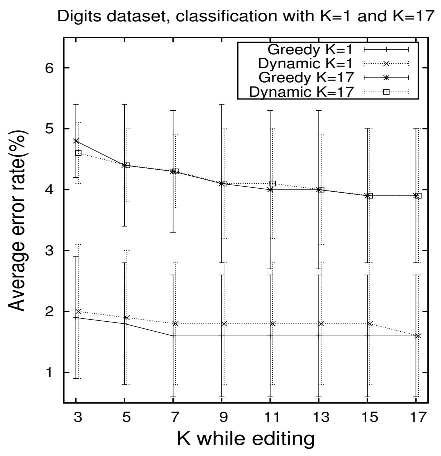


(f)

Figure 3: Heat map for average error rates for different algorithms and datasets. The number in the cells indicates the standard deviation.



(a)



(b)

Figure 4: Error rate when classifying edited letter (4a) and digit (4b) datasets for K=1 and K=17.

References

Abreu, J., Rico-Juan J.,2010. A new editing scheme based on a fast two-string median computation applied to OCR. In: Structural, Syntatic and Statisticas Pattern Recognition, Vol 6218, pp. 748-756.

Aho, A., J. Hopcroft, and J. Ullman, Data Structures and Algorithms. 1983: Addison-Wesley.

Bontempi, B., Marcelli, A., 1995. Towars a genetic based prototyper for character shapes. In: 3rd Int. Conf. on Document Analysis and Recognition. Vol. 2, pp. 694–697.

Cardenas, R., 2004. A learning model for multiple-prototype classification of strings. In: 17th Int. Conf. on Pattern Recognition. Vol. 4, pp. 420–423.

Casacuberta, F., 2000. Topology of strings: median string is NP-complete. Theoretical Computer Science. Vol. 230, pp. 39–48.

Devijver, I., Kittler, J., 1980. On the edited nearest neighbour rule. In:5th Int. Conf. on Pattern Recognition, pp. 72-80.

Duta, N., Jain, A., Dubuisson-Jolly, M., 1999a. Learning 2D shape models. In: IEEE Computer Society Conf. on Computer Vision and Pattern Recognition. Vol. 2, pp. 8-14.

Duta, N., Sonka, M., Jain, A., 1999b. Learning shape models from examples using automatic shape clustering and procrustes analysis. LNCS,Information Processing in Medical Imaging Vol. 1613, pp. 370-375.

Duta, N., Jain, A., Dubuisson-Jolly, M.,2001. Automatic construction of 2D shape models. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 23, pp. 433-446.

Ferri, F., Vidal, E., 1992. Comparison of several editing and condensing techniques for colour image segmentation and object location. Pattern Recognition and Image Analysis.

Ferri, F., Albert, J. V., Vidal, E., 1999. Considerations about sample-size sensitivity of a family of edited nearest-neighbor rules. IEEE Transactions on Systems, Man,and Cybernetics, Part B, Vol. 29 (5), pp. 667-672.

Fisher, I., Zell, A., 2000. String averages and self-organizing maps for strings. In: Proceedings of the Neural Computation 2000. pp. 208–215.

Guan, D., Yuan, W., Lee, Y., Lee, S., 2009. Nearest neighbor editing aided by unlabeled data. Information Sciences, Vol. 179, pp. 2273-2282.

Jiang, X.,Schiffmann, L., Bunke, H, 2000. Computation of median shapes. In: 4th Asian Conference on Computer Vision.

Jiang, X., Abegglen, K., Bunke, H., Csirik, J., 2003. Dynamic computation of generalised median strings. Journal Pattern Analysis and Applications. Vol. 6, pp. 185–193.

Koplowitz, J., Brown, T., 1981. On the relation of performance to editing in nearest neighbour rules. Pattern Recognition, Vol. 13, pp. 251-255.

Levenshtein, V., 1966. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics, Vol. 10, pp. 707-710.

Martínez, C., Juan, A., Casacubierta, F., 2003. Median strings for k-nearest neighbour classification*1. Pattern Recognition Letters, Vol. 24, pp. 173-181.

Marzal, . A., Palazón, V., Peris, G., 2006. Contour-Based Shape Retrieval Using Dynamic Time Warping. Lecture Notes in Computer Science, Vol. 4177, pp. 190-199.

Ong, J. L., Seghouane, A., Osborn, K., 2008. Mean shape models for polyp detection in CT Colonography. IEEE Computer Society, DICTA. pp. 287-293.

Penrod, C., Wagner, T., 1977. Another look at the edited neares neighbour rule. IEEE Trans. on Systems, Man and Cybernetics, Vol. 7, pp. 92-94.

Platonov, J., Langer, M., 2007. Automatic contour model creation out of polygonal (CAD) models for markless augmented reality. In: Proceedings of ISMAR 2007. pp. 75-78.

Rico-Juan, J. R., Micó, L., 2003. Comparison of aesa and laesa search algorithms using string and tree-edit distances. Pattern Recognition Letters. Vol. 24, pp. 1417–1426.

Sánchez, J., Pla, F., Ferri, F., 1997. Using the nearest centroid neighbourhood concept for editing purposes. In: 7th Symposium National de Reconocimiento de Formas y Análisis de Imágen, Vol. 1, pp. 175-180.

Sánchez, G., Lladós, J., Tombre, K., 2002. A mean string algorithm to compute the average among a set of 2d shapes. Pattern Recognition Letters. Vol. 23, pp. 203–213.

Tomek,I., 1976. An experiment with the edit nearest neighbour. IEEE Trans. on Systems, Man and Cybernetics, Vol. 6, pp. 448-452.

Tomek,I., 1976. A generalization of the k-NN rule. IEEE Trans. on Systems, Man and Cybernetics, Vol. 6, pp. 121-126.

Vázquez, F., Sánchez, J., Pla, F., 2005. A stochastic approach to Wilson's editing algorithm. Lecture Notes on Computer Science, Vol. 3523, pp. 35-42.

Wagner, R., Fischer, M., 1974. The String-to-String Correction Problem. Journal of the ACM, Vol. 21, pp. 168-173.

Wilson, D., 1972. Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans. on Systems, Man. and Cybernetics, Vol. 2, pp. 408-421.

Wilson, D., Martínez, T., 2000. Reduction techniques for instance based learning algorithms. Machine Learning, vol 38, pp. 257-286.

Yoon-Sik, T., 2007. A Leaf Image Retrieval Scheme Based on Partial Dynamic Time Warping and Two-Level Filtering. In: Proceedings of 7th IEEE Int. Conf. on Computer and Information Technology. pp. 633-638.

Yu, S., Tan, D., Huang, K., Tan, T. 2007. Reducing the effect of noise on human contour in gait recognition. Biometric Autentication. pp.38–346.