DESIGNING SYNTACTIC PATTERN CLASSIFIERS USING VECTOR QUANTIZATION AND PARAMETRIC STRING EDITING

B.J. Oommen and R.K.S. Loke

TR-97-09 MAY 1997

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

Designing Syntactic Pattern Classifiers Using Vector Quantization and Parametric String Editing

B. J. Oommen[*][†] (oommen@scs.carleton.ca)

R. K. S. Loke (RichardLo@corel.ca)

Abstract

We consider a fundamental problem in Syntactic Pattern Recognition (PR) in which we are required to recognize a string from its noisy version. We assume that the system has a dictionary which is a collection of all the ideal representations of the objects in question. When a noisy sample has to be processed, the system compares it with every element in the dictionary based on a nearest-neighbor philosophy. This is typically achieved using three standard edit operations - substitution, insertion and deletion. To accomplish this, one usually assigns a distance for the elementary symbol operations, $d(.,.)$, and the inter-pattern distance, $D(.,.)$, is computed as a function of these symbol edit distances. In this paper we consider the assignment of the inter-symbol distances in terms of the novel and interesting assignments - the parametric distances - recently introduced by Bunke *et al.* [4]. We show how the classifier can be trained to get the optimal parametric distance using vector quantization in the meta-space, and report classification results after such a training process. In all our experiments, the training was typically achieved in a very few iterations. The subsequent classification accuracy we obtained using this single-parameter scheme was 96.13 %. The power of the scheme is obvious if we compare it to 96.67%, which is the accuracy of the scheme which uses the *complete* array of inter-symbol distances derived from a knowledge of *all* the confusion probabilities.

1 Introduction

We consider a fundamental problem in Syntactic Pattern Recognition (PR) in which we are required to recognize a string from its noisy version. As opposed to the field of statistical PR where testing and training are achieved using numerical features, in syntactic PR the classifiers are designed by representing the patterns symbolically using primitive or elementary symbols. The PR system essentially models the noisy variations of typical samples of the patterns symbolically, and these models are utilized in both the training and testing phases. Notice that in statistical PR, the noisy samples from a given class are modeled either parametrically or non-parametrically using probability distributions, but the analogous problem in syntactic PR involves characterizing the set of noisy strings within each class.

In this paper we assume that the system has a dictionary which is a collection of all the ideal representations of the objects in question. Although a detailed survey of the types of dictionaries

[*]Senior Member, IEEE. Both authors can be contacted at : School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, CANADA.

[†]Partially supported by NSERC - the Natural Sciences and Engineering Research Council of Canada.

If for all $a, b \in \mathrm{A}$ these distances are assigned unit values as below :

$$\begin{aligned} d_s(a,b) &= 1 \quad \text{if } a \neq b \\ &= 0 \quad \text{if } a = b \\ d_i(a) = d_e(a) &= 1 \quad \text{for all } a \end{aligned} \tag{1}$$

the inter-string edit distance is called the Levenshtein distance.

In general, however, these distances can be real numbers, and the training of the classifier involves determining the optimal values of the inter-symbol distances so as to minimize the classification error (for example, in typewritten text, assigning a smaller distance for adjacent letters of the keyboard, or in molecular biology, for amino acids which are more prone to be replaced by other amino acids).

There are currently three ways of assigning the inter-symbol edit distances. The first method involves assigning the distances in terms of the negative logarithms of the confusion probabilities as follows :

$$\begin{aligned} d_s(a,b) &= -ln\frac{\Pr\ [a \rightarrow b]}{\Pr\ [a \rightarrow a]} \quad \text{if } a \neq b \\ d_e(a) &= -ln\frac{\Pr\ [a \rightarrow b]}{\Pr\ [a \rightarrow \lambda]} \quad \text{for all } a \\ d_i(a) &= K_i d_e(a) \end{aligned} \tag{2}$$

where K_i is an empirically determined constant, and Pr $[a \rightarrow b]$ is the probability that the symbol 'a' was transformed into the 'b'.

Although this method has been used, [8, 9, 16, 17, 19] to set the values of these distances, the method assumes that the confusion probabilities are known *a priori* - prior to the training phase. Similar methods have also been used to compute the so-called **PAM** matrices in comparing biological macromolecules.

Besides the above method we are aware of two other schemes that have been used in learning the inter-symbol distances [18, 21]. The method of Ristad *et al.* [18] is a recent work and is a comprehensive demonstration of the fact that the distances can be obtained as explicit functions of the garbling probabilities for a variety of stochastic models. The authors of [18] have first proposed various stochastic models for the string edit distance, and in each case the model and the corresponding estimation procedure have been developed to yield the optimal set of inter-symbol edit distances from a corpus of examples. The work is fascinating and powerful, and has been tested for the problem of learning the pronunciation of words in conversational speech. Although we have not experimentally tested their algorithms, we believe that their scheme will have powerful applications.

The other method which we would like to particularly highlight is due to Vidal *et al.* [21]. Their method is a classic and brilliant scheme which demonstrates how traditional statistical PR methods can be applied to syntactic problems. The idea which they have used is akin to the acclaimed algorithm which learns the best weight vector describing the discriminant function in the case of linearly separable classes. Vidal *et al.* have shown how to map the syntactic problem into a statistical setting, and then used a hill-climbing scheme to converge to the set of inter-symbol distances.

In this paper we consider how we can utilize the novel and interesting distance assignments - the Parametric String Distances (PSDs) - recently introduced by Bunke *et al.* [4]. In this case, these distances are assigned values as below for all $a, b \in \mathrm{A}$:

$$\begin{aligned} d_s(a,b) &= r \quad \text{if } a \neq b \\ &= 0 \quad \text{if } a = b \\ d_i(a) = d_e(a) &= 1 \quad \text{for all } a. \end{aligned} \tag{3}$$

Indeed, we shall investigate the way of *optimally* assigning the parameter when the input to the problem is data specifying the set of true words and their garbled versions. We shall show how the parameter $'r'$ can be optimally chosen if such a parameter which discriminates patterns perfectly actually exists.

In their paper Bunke *et al.* [4] alluded to a strategy by which the optimal parameter could be learnt, but we shall show that the time complexity required for their process is prohibitive. On the other hand, we shall present a scheme which utilizes the excellent properties of Bunke's parametric string distance, but converges to the optimal parameter by a variant of vector quantization. This demonstrates that these properties present us with a platform to tackle the problem of inferring a systematic set of edit costs (which are not entirely *symbol dependent*) based on a sample set of true and noisy patterns. Furthermore, the scheme does not require the estimation of the confusion probabilities, but utilizes a strategy which dynamically *adjusts* the edit costs *via* the parameter r. It is also noteworthy that like any vector quantization strategy, these costs will be capable of tracking and adapting to a dynamically changing environment, where the confusion probabilities are non-stationary. Indeed, in our setting, vector quantization will be performed in the meta-space (as opposed to the *physical* space in which the features are), to assign the optimal value of the parameter.

Although, we shall discuss in greater detail how this vector quantization is achieved, before we proceed, we shall first present Bunke's PSD for strings and specify the properties which render such a learning strategy feasible. A *brief* overview of vector quantization is also presented to clarify the salient features of our algorithm.

2 Parametric String Edit Distance

2.1 Overview

As mentioned earlier, Bunke and Csirik proposed a new Parametric String Distance (PSD) measure [4] which is a special case of the Generalized Levenshtein Distance[1] which can be computed using the Wagner and Fischer [22] algorithm. The edit cost for the insertion and deletion operations are assigned the value unity, but the substitution (replacement) cost, $d_s(a, b)$ is given as a parameter, r, whenever the symbols $a \neq b$, and is assigned the value zero otherwise. Thus, the edit distance between two strings is almost entirely computed in terms of the parameter r, which due to the triangular inequality is bounded by 2. This bound is obvious, since when the value of r exceeds two, the edit process will not utilize the substitution edit operation at all.

Using the edit cost function defined in terms of the parameter r, Bunke and Csirik [4] specified the properties of the *inter-string* edit distance. If the distance for a pair of strings is uniquely minimum over the interval $r = (0, 2)$, we will obtain a unique distance function (in terms of r). If, however, the distance function associated with a pair of strings is not uniquely defined, then the inter-string distance is specified in terms of a variety of distance functions, and each function is uniquely minimal for a given interval of the value of r. To clarify the situation, consider the following example.

Let $A = \{a, b, c\}$, and let $X = baacb$ and $Y = acba$. The resulting PSD as computed by the algorithm described in [4] for the strings is shown in Table 1. Table 1 is essentially a modified trellis which shows the associated PSD with their corresponding intervals of r-values instead of *merely* the usual numeric edit distance.

We shall refer to *items* in the modified trellis in a matrix-like manner where $D(i, j)$ refers to the distance tabulated in row i and column j. From the table we see that until $D(2, 2)$, all

[1]We assume that the reader is reasonably aware of the properties of the Levenshtein Distance and the algorithm due to Wagner and Fischer [22].

		a	c	b	a
	[0, (0,2)]	[1, (0,2)]	[2, (0,2)]	[3, (0,2)]	[4, (0,2)]
b	[1, (0,2)]	[r, (0,2)]	[1+r, (0,2)]	[2, (0,2)]	[3, (0,2)]
a	[2, (0,2)]	[1, (0,2)]	[2r, (0,1)] [2, (1,2)]	[1+2r, (0,1)] [3, (1,2)]	[2, (0,2)]
a	[3, (0,2)]	[2, (0,2)]	[1+r, (0,2)]	[3r, (0,1)] [2+r, (1,2)]	[1+2r, (0,1)] [3, (1,2)]
c	[4, (0,2)]	[3, (0,2)]	[2, (0,2)]	[1+2r, (0,1)] [3, (1,2)]	[4r, (0,1)] [4, (1,2)]
b	[5, (0,2)]	[4, (0,2)]	[3, (0,2)]	[2, (0,2)]	[1+3r, (0,2/3)] [3, (2/3,2)]

Table 1: The modified trellis for the PSD. Every element in the modified trellis is a pair of the form $[D, i]$ where D is the PSD, and i is the interval in which D is defined.

distances in the trellis are uniquely defined over the entire r-interval. In other words, for all values of r between zero and two, the same distance function can be used to produce a minimum edit distance with respect to the value of r. Thus, for example, at $D(1,1)$, to compute $D('a','b')$, the edit distance could be due to a substitution which implies a value of r, or an insertion-deletion pair which implies a value of 2. Now, since $r \in (0,2)$, the minimum value at $D(1,1)$ will obviously be r. Things become more interesting at $D(2,2)$ where the distance function due to a substitution is $2r$, while the distances due to an insertion and a deletion are $2+r$ and 2 respectively. Consequently, it can be seen that the distance function due to a substitution, namely $2r$, is minimum when $r \in (0,1)$, while in the interval where $r \in (1,2)$, the distance function due to an insertion, namely 2, is minimum.

Subsequent to this juncture, the computations could proceed in one of two ways. The first deals only in the interval where $r \in (0,1)$ while the second computes the distances for the functions when $r \in (1,2)$. This, of course, is done sequentially where computations are performed first for $r =$ (0,1). The computations will be performed recursively until the ends of the strings (in our case, it is $D(5,4)$) are encountered. When this is done, the algorithm will return to *position* $D(2,2)$ in the trellis to re-compute the distances for $r \in (1,2)$. As a result of this, subsequent trellis elements can potentially have identical distance functions with adjacent intervals (eg. [1+2r, (0,1)] and [1+2r, (1,2)] in the same trellis entry), and in these situations, the distance functions will be merged. The value of $D(5,4)$ will eventually denote the *actual* PSD between the strings *'baacb'* and *'acba'*. From Table 1, we can conclude that

$$D(X,Y) = \begin{cases} 1+3 \text{ if } r \in (0,2/3) \\ 3 \text{ if } r \in (2/3,2) \end{cases}$$

Bunke and Csirik [4] proved the following properties about the PSD :

Property 1 *: At any position $D(i,j)$, $1 \le i \le N$, $i \le j \le M$, in the cost matrix, the distance* *between X_i and Y_j obeys the following functional form:*

$$D(i,j) = D(X_i, Y_j) = \begin{cases} a_1 + b_1 r \text{ if } 0 = r_0 \leq r \leq r_1 \\ a_2 + b_2 r \text{ if } r_1 \leq r \leq r_2 \\ \vdots \qquad \vdots \\ a_k + b_k r \text{ if } r_{k-1} \leq r \leq r_k = 2, \end{cases}$$

where $0 = r_0 < r_1 < r_2 < \cdots < r_k = 2$ are rationals and $a_1, \cdots, a_k$, $b_1, \cdots, b_k$ are non-negative integers.

Property 2 *: For $i = 1, \cdots, k-1$ the following relations hold:*

1. $a_i + b_i r_i = a_{i+1} + b_{i+1} r_i$
2. $b_i > b_{i+1}$.

Property 3 *:*
$b_i \leq min(N, M)$, where N and M are the lengths of the two strings.

Property 4 *: For the PSD of any two strings of length N and M respectively given in the form of Property 1, the number of subintervals (of values of r) is less than or equal to $min(N, M)$.*

Property 5 *: The time complexity of the algorithm for PSD computation is $O(N^2M)$ where N and M are the lengths of the two strings to be compared, and $N = min(N, M)$.*

In [4], Bunke and Csirik had alluded to using the PSD in applications of PR with the intention of determining a suitable value of r such that the classification criteria for the training problem are satisfied. In order to achieve this, they considered the problem of combining different PSDs or distance functions (in term of r) of pairs of strings. The combination of the PSD functions is crucial in locating the *perfect* value of r for a given PR setting since we have to maximize or minimize a criterion function which varies with r. In other words, in order to find the value of r that, for example, maximized the training classification accuracy, we need to consider all the resulting distance functions that emerge from a given set of patterns.

Now, given a criterion function T, one can definitely maximize (or minimize) T by a brute force method by computing T for all possible values of r and then selecting the value of r that maximizes (or minimizes) T. This is, however, not feasible since $r \in (0, 2)$ is real. Fortunately, Bunke and Csirik proved in [4] that we do not need to consider every possible value of r, but rather only need to examine the *critical points* of the set of distance functions available from the set of data, where the critical points are defined below.

Consider a set of distance functions, $S = \{D_1(r), D_2(r), \cdots, D_K(r)\}$, where $D_i(r)$ is as defined in Property 1 and is of the form:

$$D_i(r) = \begin{cases} a_1^{(i)} + b_1^{(i)} r \text{ if } 0 = r_0^{(i)} \leq r \leq r_1^{(i)} \\ a_2^{(i)} + b_2^{(i)} r \text{ if } r_1^{(i)} \leq r \leq r_2^{(i)} \\ \vdots \qquad \vdots \\ a_{L_i}^{(i)} + b_{L_i}^{(i)} r \text{ if } r_{L_{i-1}}^{(i)} \leq r \leq r_{L_i}^{(i)} = 2 \end{cases}$$

Then, the set of critical points of S is completely defined by the following rules:

1. Any interval border $r_j^{(i)}$ is a critical point; $i = 1, \cdots, K$; $j = 0, \cdots, L_i$

2. Any point of intersection between two distinct linear pieces of distinct $D_i(r)$ and $D_j(r)$ is a critical point. Thus, if

$$a_k^{(i)} + b_k^{(i)} r = a_l^{(i)} + b_l^{(i)} r$$

for $i, j = 1, \cdots, K$; $i \neq j$; $k \in \{1, \cdots, L_i\}$; $(a_k^{(i)}, b_k^{(i)} \neq a_l^{(i)}, b_l^{(i)})$,

then r is a critical point.

In addition, Bunke and Csirik also defined a *basic interval* as follows. Given a set of critical points $\{r_0, r_1, \cdots, r_L\}$ of the set of distance functions S where $0 = r_0 < r_1 < \cdots < r_L = 2$, a basic interval is the interval (r_j, r_{j+1}) for $j = 0, \cdots, L-1$. Intervals defined for each individual distance function D_i (intervals in which different string edit distance functions attain the minimum over the intervals as described earlier) are also defined as basic intervals. Bunke and Csirik proved that there are at most LK^2 basic intervals where K is the number of distance functions in S and L is the number of basic intervals for each distance function in S. With the definition of critical points and basic intervals, Bunke and Csirik proved the following property of the PSD.

Property 6 : *Given a set of distance functions, S, if a maximum (or minimum) of a criterion function occurs within a basic interval, this extreme point also occurs at least at one of the borders of the interval (i.e., it also occurs at one of the critical points bounding the basic interval).*

Property 6 has an important implication on the problem of systematically inferring the edit costs based on a sample set of training data and an associated criterion function, for example, the training classification accuracy. Indeed, utilizing the PSD one can *intelligently* assign values of edit costs without the exhaustive enumeration of all possible edit costs.

2.2 Drawback of Bunke's Result

Although the PSD allows for the assignment of edit costs without explicit and exhaustive enumeration of all possible edit costs, the computation that is required would still be prohibitive when the set of sample patterns is large. This is because in order to obtain the set critical points, all possible pairs of strings in the sample set and their corresponding sets of basic intervals would have to be considered. This leads us to the following results.

Theorem 1 : *To assign a suitable value of r, the substitution cost, for a given set of sample data and a given criterion function, each possible pair of strings in the set has to be considered.*

Proof : Bunke *et al.* have shown in [4] that to compute the optimal r we will have to determine all the *critical points* in a given set of sample patterns. Using these we should be in a position to determine the maximum (or minimum) of the given criterion function with respect to r. In order to obtain all the critical points, the corresponding distance functions for every possible pair of strings in the sample set will have to be evaluated. Only with this complete set of distance functions will we be able to determine all the possible critical points which, in turn, are crucial for the computation of the intersecting points of the distance functions. Subsequently, these can be used to learn the critical points that extremize the criterion function. ■

Directly from Theorem 1, we can obtain the following result.

Theorem 2 : *The time required by Bunke and Csirik's algorithm is $O(J^2Q^2N^2MLp)$ where J is the number of strings in the sample set, Q is the size of the dictionary, N and M are the sizes of the strings in the set, L the number of basic intervals as defined earlier (prior to Property 6) and p the number of critical points derivable from the distance functions of the sample set.*

Proof : Assume that there are J strings in the sample set, all of length M, and that there are Q words, all of length[2] N, in the dictionary, H.

This means that the number of possible pairs of strings to be compared is $O(JQ)$. Furthermore, by Property 5, for each pair, a time of $O(N^2M)$ is required to perform the computations to obtain the PSD. Thus, in order to obtain all the necessary distance functions (by Theorem 1), we require $O(JQN^2M)$. Additionally, to correctly evaluate the optimal value of r we will also have to locate all the intersecting points among the set of distance functions found previously. This requires another $O(JQL)$ time, where L is the maximum number of basic intervals of a distance function. Finally, given that we have obtained p critical points, we need $O(p)$time in order to determine the correct value of r from among these points. Combining these terms yields the required time complexity. ■

Due to the prohibitive time required, the method described in [4] cannot be realistically applied to infer the edit costs in a practical PR setting. This has motivated us to investigate an alternative approach to inferring the edit costs. As alluded to earlier, we shall be utilizing vector quantization together with the properties of the PSD discussed above. Before describing our proposed method, we shall give a *brief* overview on vector quantization.

3 Vector Quantization

Vector quantization (VQ) is based on concepts that have been applied traditionally in the estimation of probability density functions. A probability density function, $P(u)$ is approximated using a finite set of *codebook* vectors, $\{m_i \mid i = 1, \cdots, k\}$ by the VQ process, where each m_i approximates the value at u. Given a set of codebook vectors, the approximation of u involves finding the reference vector m_c closest to u.

With the introduction of codebook vectors, VQ essentially compresses the amount of information received. This is important and extremely practical in applying VQ to applications which process large amounts of information, as in signal processing. It is interesting to note that VQ represents data in the actual feature space through the use of the codebook vectors. These codebook vectors are **migrated** in the feature domain so that they collectively represent the distribution under consideration.

Traditionally, the codebook vectors are initially randomly selected from the feature space. The algorithm is repeatedly presented with a sample data point and using an appropriate updating (or migrating) algorithm, it locates the closest codebook vector and migrates the vector towards the data point. Explicitly, the updating algorithm can be described as follows:

$$m_i(t+1) = \begin{cases} (1-\alpha)m_i(t) + \alpha p \text{ if } m_i \text{ is the closest point to the data point } p \\ m_i(t) \text{ otherwise,} \end{cases}$$

where t is the time or iteration number.

The choice of α is rather significant in obtaining a respectable estimation of the true distribution in terms of codebook vectors. As recommended in the literature [10], α should be decremented linearly from unity for the initial learning phase and in the subsequent phase, a small value of α is recommended. The subsequent phase is utilized to fine-tune the codebook vectors and thus require no significant changes. It is recommended that in this fine-tuning phase, α should decrease linearly from a value of 0.2.

[2]If the words are of different lengths, N_i and M_j would be the lengths of X_i and the j^{th} word compared. The complexity argument is however, in principle, the same.

The discussion of vector quantization presented here is by no means complete - it is not appropriate for us to describe VQ in any more depth here. Interested readers are referred to an excellent survey by Kohonen in [10]. However, with this general overview of vector quantization, we are now in a position to describe our proposed algorithm for optimally inferring edit costs in a PR setting.

4 Optimal Parametric String Editing Using Vector Quantization in the Meta-space

The algorithm that we propose here is based on vector quantization in the *meta-space* - the space of all possible values of r, the parameter used in the PSD. Traditionally, as described in the previous section, vector quantization works on a sample set of data from the set of patterns presented. This sample set, also known as codebook vectors, are then migrated in its feature domain to represent the distribution that is implicit from the given set of patterns. However, this is not directly applicable in our case since our aim is to assign the value of r based on the discriminant criterion function. Instead, in our quest for obtaining the optimal value of r, we shall *not* work in the space where the features (the strings/symbols) reside, but rather in the meta-space where the parameter r resides, since *evolving* the value of r will enable us to maximize the training classification accuracy. A detailed description of our algorithm presented below.

The algorithm is presented with a particular set (dictionary) of prototypical strings, H and an associated set of typical noisy samples. We assume that the strings are linearly separable by using the PSD and a suitable parameter r^*. Thus for each noisy string Y the algorithm is provided with the true word $X^* \in H$ which it represents. These noisy samples are compared with each individual element of H using a string comparison algorithm [22], and classified to be the noisy version of the closest word in H. If this results in a misclassification, the PSD functions associated with the noisy string and the *mis-classified string*, X^+, and the correct string, X^*, respectively are generated using the algorithm proposed in [4] and their associated critical points, c are computed.

For each critical point, c, let the PSD functions based on the strings X^* and X^+ be D^* and D^+ respectively. Let the actual values of D^* and D^+ when evaluated at the corresponding critical points c_1 and c_2 be $D^*_{c_1}$ and $D^+_{c_2}$ respectively. The question now is one of determining how we can get the value of r for the next iteration in the VQ cycle. Indeed, in the true philosophy of VQ the next value of r is obtained by migrating the current values of r towards one of these critical points. The criterion function for determining which critical point is to be chosen in the migration for the next value for r is obtained by evaluating the difference of the distances, $D^*_{c_1}$ and $D^+_{c_2}$ relative to the critical point c, where F is the criterion function given below:

$$F = \frac{D^+_{c_2} - D^*_{c_1}}{c}$$

Indeed, we have chosen the above function, because in the true spirit of VQ, the migration of r is done in the direction of the critical point that maximizes the distance between Y and X^+, and simultaneously minimizes the distance between Y and X^*. Thus, the new r is obtained by moving in the direction of the linear discriminant, and consequently increases the rate of classification in the nearest neighbor PR setting. The algorithm is given formally below and its convergence properties follow.

```
Algorithm Compute_Optim_Param
Input:
  1. The dictionary of prototype strings, $H$.
  2. The set of noisy strings assumed to be linearly separable. The current noisy string
     processed is $Y$.
  3. An initial value of the substitution edit cost, $r$.
  4. Number of training iterations, Iteration_num.
  5. Edit cost migration factor, $\alpha$.
Output: The optimal substitution edit cost, $r$.
Method:
Loop
  (Iteration_num times) Or
  (until a training cycle with 100% recognition rate)
  /* Start of a training cycle */
  Classify $Y$ by comparing it with every $X \in H$
  If $Y$ is incorrectly classified to $X^+$ instead of $X^*$ Then
    Determine the distance functions, $D^*$, of $Y$ and $X^*$
    Determine the distance functions, $D^+$, of $Y$ and $X^+$
    Determine the associated critical points as in [4]
    Each Distance function is of the form $a + br$
    Each critical point is a potential optimal value of $r$
    For each critical point ($c$) Do
      • Calculate the value of the particular functions of the
      form $a + bc$ in $D^*$ and $D^+$
      • Calculate $\rho$, the criterion ratio by evaluating the
      difference of distances at $c$ for $D^*$ and $D^*$ realtive
      to $c$ where $\rho = \frac{D^+_{c_2} - D^*_{c_1}}{c}$
    End For
    • Choose $c$ which maximizes $\rho$ to be the new optimal
    edit cost and set this value to $r_{mig}$
    • $r = (1 - \alpha)r + \alpha r_{mig}$
  End If
End Loop
Return $r$
End Algorithm Compute_Optim_Param
```

Theorem 3 : *If the samples are linearly separable using a PSD criterion, the algorithm will converge to* **an** *optimal value of* r *- even though such an optimal value may not be unique.*

Proof : From the results of [4], we can see that to recognize a particular string, Y, the optimal value of r lies in an *interval* described by the critical points of the various distance functions. This is because, as proven in [4], each critical point is bounded by two mutually exclusive basic intervals. Hence, for a specific Y, given a critical point r_{i_Y}, the optimal values of r may also exist in either of the neighboring intervals (r_{i-1_Y}, r_{i_Y}) and/or $(r_{i_Y}), r_{i+1_Y})$. But since the PSD functions intersect each other at the critical points, the optimal value could also have been at r_{i-1} or r_{i+1}. We refer to these left and right boundaries of *any one* such interval for Y as $r_{L_j}(Y)$

and $r_{R_j}(Y)$ respectively. This proves that if the string classification accuracy of a *single* string is considered as a function of r, the function is maximized in a sequence of intervals each of which is defined by the end-points $r_{L_j}(Y)$ and $r_{R_j}(Y)$, and within each such interval the classification is perfect because the strings are assumed to be linearly separable.

We are now searching for a value of r, say r^* which will classify every noisy sample to its true un-garbled element of the dictionary. Since the strings are linearly separable, it is clear that any such r^* lies in the intersection of the regions defined for each individual Y. Thus,

$$r^* \in [\bigcup_j \bigcap_Y [r_{L_j}(Y), r_{R_j}(Y)]],$$

where each intersecting region $\bigcap_Y [r_{L_j}(Y), r_{R_j}(Y)]$ represents a domain where all the strings will be correctly classified.

Consider now an iteration in the algorithm in which X^* is the true word in H, from which Y was derived, and that the current value of r has mis-classified Y to $X^+ \neq X^*$. This implies that as far as this *particular* Y is concerned, the classification is not at its maximum at this current value of r. Consequently, a migration in the direction of one of the maxima will lead us to a local maximum. But, in this case, all the local maxima are also global maxima, since all the maxima attain the value unity. Thus, we are guaranteed that by migrating towards the appropriate boundary point $r_{L_j}(Y)$ or $r_{R_j}(Y)$ we will definitely be moving towards one of the global maximum. Indeed, this is achieved by maximizing ρ, because we move in the direction which minimizes the PSD between Y and X^* and which simultaneously maximizes the PSD between Y and X^*.

Just like the perceptron algorithms [7, 10], since the VQ process eventually converges for these two strings in H to r_{mig}, we are guaranteed that every time X^* and X^+ compete, the current value of r will be forced to migrate to this optimal boundary value which will correctly classify the corresponding noisy string Y to X^*.

The result follows since this is true for every single pair of strings in H which could be competing elements for any Y, and since every intersection of corresponding sets must contain a solution r^*. ■

The advantage of our proposed algorithm is obvious. First of all it is amazingly simple in principle and easy to implement. It also converges extremely rapidly. Although Bunke and Csirik's algorithm [4] enables us to concretely obtain the value of the desired parameter, it is difficult to apply it to a practical problem because the time required is prohibitive. Also, unlike the traditional method which relates the inter-symbol distances to the confusion probabilities [8, 9, 16, 17, 19] and the scheme proposed by Ristad [18], we do not require the algorithm perform any estimation procedure in the assignment of the weights. And finally, unlike the classical method of Vidal *et al.* [21] we do not have to map our problem into a statistical domain where the learning is achieved. Instead, our algorithm is very easy to implement and practical since we always only perform comparisons between individual pairs of strings, and do not have to consider the overall recognition accuracy. Indeed, we believe that an analogous algorithm can be developed which considers the overall recognition accuracy (instead of the individual string recognition accuracy) at every step of the migration process. The reasoning for this is given below.

Let Y^k be the noisy version of an unknown string X^*. For the given PSD classification scheme with a parameter $r \in [\sigma, \gamma)$, the distance function $D_r(X^*, Y^k)$ represents a distance measure between the noisy string Y^k and the true string X^*.

We can now define

$$A_r(Y^k) = \begin{cases} 1 & \text{iff} \qquad D_r(X^*, Y^k) = \min_{X \in H} D_r(X, Y^k) \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

where H is a dictionary of strings.

Thus, $A_r(Y^k)$ is an indicator function which indicates whether or not the given string will be classified correctly under the PSD scheme with a given dictionary H and a parameter r.

In the same vein, for a given a dictionary H, we can also define a function $A_r^*(r)$ which represents the classification *power* of the PSD scheme as a function of the parameter r, as:

$$A_H^*(r) = \sum_{k=1}^{J} A_r(Y^k), \forall r \in [\sigma, \gamma] \tag{5}$$

Now assume that for each noisy string Y^k there exists a range of parameter values $\sigma \leq \sigma^k \leq r < \gamma^k < \gamma$ for which the PSD scheme correctly classifies Y^k. Thus,

$$\begin{aligned} A_{r^k}(Y^k) &= 1 \quad \sigma^k \leq r < \gamma^k \\ &= 0 \quad \textit{otherwise.} \end{aligned} \tag{6}$$

We are now ready to state and prove an important property of the classification power function $A_H^*(r)$.

Theorem 4 : *If there exists a parameter r^* such that the PSD scheme with dictionary H correctly classifies all noisy strings $Y^k, k = 1, \cdots, J$ for this parameter value, then the function $A_H^*(r)$ has no local maximum for any parameter r in the range $[\sigma, \gamma)$ defined above.*

Proof : We prove the theorem by contradiction. Assume that, there exists at least a single local maximum, say, for the parameter $r^+ \in [\sigma, \gamma)$. We have the two possibilities,

$$\sigma \leq r^* < r^+ < \gamma$$
$$\sigma \leq r^+ < r^* < \gamma.$$

In the first case, consider any parameter value r' between r^* and r^+, i.e., $r^* < r' < r^+$. Then, by definition of a local maximum, we have,

$$A_H^*(r^*) > A_H^*(r') < A_H^*(r^+).$$

By substituting from the definition of $A_H^*(r^*)$ from (5), we get,

$$\sum_{k=1}^{N} A_{r^*}(Y^k) > \sum_{k=1}^{N} A_{r'}(Y^k) < \sum_{k=1}^{N} A_{r^+}(Y^k)$$

This means that there exists at least some string Y^k which is correctly classified by the PSD scheme for parameter r^+ but not for parameter r'. Further, by hypothesis, $A_H^*(r^*) = N$, since every string is correctly classified by the PSD scheme for parameter r^*. That is,

$$\begin{aligned} A_{r^*}(Y^k) &= 1 \\ A_{r'}(Y^k) &= 0 \\ A_{r^+}(Y^k) &= 1. \end{aligned}$$

Since $r^* < r' < r^+$, this is impossible given the assumption in (6) and hence, we arrive at a contradiction.

The proof of the theorem is identical for the other case where $\sigma \leq r^+ < r^* < \gamma$. ■

Since the classification function has a unique maximum in each such interval, we believe that the fundamental principle of migrating the current value of r to a point within this interval will still lead to an efficient training procedure. However, this is still unproven. Indeed, the development of a training procedure/classifier which uses the overall classification accuracy as a criterion remains open.

5 Experimental Results

We have tested our algorithm to demonstrate its use in the training and testing of a PR system in which the dictionary, H, consisted of 310 words which was a subset of the most common words in English augmented with words used in the computer literature [6]. The length of the strings was greater than or equal to 7 and the average length of a string was approximately 8 characters. From these, a set of 930 noisy strings were generated using the method described in [15]. This method is given below in the interest of completeness so that the results can be verified.

5.1 Noisy String Generation

Let us suppose that we have to obtain a noisy version of a string $U \in A^*$, where A is the alphabet under consideration. The generation process assumes the definition of three distributions, G, R and S defined as follows. G is a distribution over the set of positive integers and defines the number of insertions performed in the mutating process and it satisfies :

$$\sum_{z \geq 0} G(z) = 1.$$

Examples of the distribution G are the Poisson and the Geometric Distributions.

The second distribution required is the distribution R, where the quantity $R(a)$ is the probability that $a \in A$ will be the inserted symbol conditioned on the fact that an insertion operation is to be performed. Note that R has to satisfy the following constraint :

$$\sum_{a \in A} R(a) = 1.$$

Finally, apart from G and R, the generation requires a probability distribution S over $A \times (A \cup \lambda)$, where λ is the null symbol. S is called the Substitution and Deletion Distribution. The quantity $S(b|a)$ is the conditional probability that the given symbol $a \in A$ in the input string is mutated by a stochastic substitution or deletion – in which case it will be transformed into a symbol $b \in (A \cup \lambda)$. Hence, $S(c|a)$ is the conditional probability of $a \in A$ being substituted for by $c \in A$, and analogously, $S(\lambda|a)$ is the conditional probability of $a \in A$ being deleted. Observe that S has to satisfy the following constraint for all $a \in A$:

$$\sum_{b \in (A \cup \lambda)} S(b|a) = 1.$$

In our case, the matrix of confusion probabilities was based on the proximity of keys on the typewriter keyboard.

Using the above distributions we now describe the garbling algorithm (the noisy string generation process). Let $U| = N$. Using the distribution G, we first randomly decide on the

Original String	Noisy String
according	acvording
account	acchouufnt
addition	manxdction
address	ajddvdregss
administration	xadministeatiin
affairs	apgfjirk
afternoon	aftderxgooin
against	whagaincst
altogether	dltoghetnr
announced	snnouncwd
apparently	apparyeontly
approach	apprwfoixach
approximately	appeoxivpnaterfly
arrived	rrived
artillery	artixlxery
attacks	attaawcrks
attention	srtention
attitude	atgitudr
authority	jauthoritj
available	aailaxbtle

Table 2: A subset of the original strings in the dictionary H and their corresponding noisy versions.

number of symbols to be inserted, say, k. The algorithm then determines the position of the insertions among the individual symbols of U. In this case, each of the $\frac{(N+k)!}{(N!k!)}$ possible positions are assumed equally likely. Using this, the individual symbols of the alphabet are inserted using the distribution R at the inserted positions. Finally, the actual symbols of U which are not at the inserted positions are now substituted or deleted using the distribution S.

The above process has been shown to be stochastically consistent and functionally complete in [15] and is to our knowledge, the only reported method by which noisy strings with arbitrary noise characteristics can be generated.

5.2 Training and Testing Results

A subset of the strings in the dictionary and their respective noisy versions are given in Table 2. To test the hypothesis that our scheme could be used in training linearly separable classes, samples which were not linearly separable for the true (but unknown) value of r^* were discarded.

Our proposed algorithm was tested on this set of data and after only *a single* iteration (as given in the formal description in the previous section), the algorithm was able to find an optimal value of the substitution edit cost, r which correctly classified all the training samples. In our case it converged to a value of $r = 1.271$.

The system was now tested using this value of r for a different set of 930 noisy strings generated from the same dictionary. Of these 930 test strings, 894 were correctly classified - yielding an accuracy of 96.13%. The same noisy strings were then classified using the *complete set of inter-symbol distances.* These distances were obtained assuming the knowledge of the the true confusion probabilities and were explicitly related to them in terms of their negative logarithms as explained in (2). In this case, the accuracy obtained was 96.67% (899 of the 930 noisy strings were correctly classified). The power of the scheme is obvious, especially because we have *not* used symbol-dependent distances, but a single parameter. Furthermore, we have achieved this level of learning accuracy without invoking any estimation strategy which learns the inter-symbol and operation garbling probabilities.

6 Conclusions

In this paper we have considered the training and testing problem in Syntactic Pattern Recognition (PR) in which we are required to recognize a string from its noisy version after training the classifier using noisy versions of the true un-garbled strings. We assume that the system has a dictionary which is a collection of all the ideal representations of the objects in question. In the testing phase, whenever a noisy sample has to be processed, the system compares it with every element in the dictionary based on a nearest-neighbor philosophy using three standard edit operations - substitution, insertion and deletion. To accomplish this, one usually assigns a distance for the elementary symbol operations, $d(.,.)$ and the inter-pattern distance, $D(.,.)$ is computed as a function of these symbol edit distances. In this paper we have considered the training of the classifier using the recently introduced distance assignments - the parametric distances due to Bunke *et al.* [4]. Although the latter distances have *very interesting* properties, we have shown that it is infeasible to utilize it in a PR setting because the time required to obtain the best value of r can be prohibitive. Instead, we have shown how the classifier can be trained to get the optimal parametric distance using *vector quantization* in the meta-space (the space containing r, and not the feature/primitive space typically utilized in PR problems). We have also reported classification results after such a training process. The training algorithm is very easy to both understand and implement and converges very quickly - in our experiments it converged in a *single* iteration - i.e., after considering every noisy string exactly once. Furthermore, the scheme we propose does not require any estimation of the garbling probabilities and is extremely easy to both implement and use.

Acknowledgments : The authors are very grateful to our friend Raghu (Mr. Govindachari Raghunath) for providing us with the Proof of Theorem 4. This paper was partially supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] A. V. Aho. Algorithms for finding patterns in strings. In *J van Leeuwen (ed.): Handbook of Theoretical Computer Science*, pages 255–230. Elsevier Science Publsihers B. V., 1990.

[2] V. Bafna, S. Muthukrishnan, and R. Ravi. Computer similarity between rna strings. In *Proceedings of the Sixth Symposium on Combinatorial Pattern Matching CPM-95*, pages 1–16, 1995.

[3] H. Bunke. Recent advances in string matching. In *Advances in Structural and Syntatic Pattern Recognition*, pages 3–21. World Scientific, 1992.

[4] H. Bunke and J. Csirik. Parametric string edit distance and its application to pattern recognition. *IEEE Transaction on Systems, Man and Cybernetics*, 25(1):202–206, 1993.

[5] A. L. Cobbs. Fast approximate matching using suffix trees. In *Proceedings of the Sixth Symposium on Combinatorial Pattern Matching, CPM-95*, pages 41–54, 1995.

[6] G. Dewey. *Relative Frequency of English Speech Sounds.* Harvard Univ. Press, Cambridge, MA, 1923.

[7] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[8] R. L. Kashyap and B. J. Oommen. An effective algorithm for string correction using generalized edit distances -i. description of the algorithm and its optimality. *Inform. Sci.*, 23(2):123–142, 1981.

[9] R. L. Kashyap and B. J. Oommen. The noisy substring matching problem. *IEEE Trans. Software Engg.*, 9:365–370, 1983.

[10] T. Kohonen. The self-organizing map. *Proc. IEEE*, 78:1464–1480, 1990.

[11] A. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Phys. Dokl.*, 10:707–710, 1966.

[12] R. Lowrance and R. A. Wagner. An extension of the string to string correction problem. *J. Assoc. Comput. Mach.*, 22:177–183, 1975.

[13] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *J. Comput. System Sci.*, 20:18–31, 1980.

[14] B. J. Oommen. String alignment with substitution, insertion, deletion, squashing and expansion operations. *Information Sciences*, 77:89–107, 1995.

[15] B. J. Oommen and R. L. Kashyap. A formal theory for optimal and information theoretic syntactic pattern recognition. In *Proceedings of SSPR-96, the 1996 International Symposium on Syntactic and Structural Pattern Recognition*, pages 11–20, Leipzig, Germany, August 1996. Preliminary Version. Submitted for Publication.

[16] B. J. Oommen and R. K. S. Loke. Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1154–1159, Vancouver, October 1995. Preliminary version. To Appear in *Pattern Recognition.*

[17] B.J. Oommen. Recognition of noisy subsequences using constrained edit distances. *IEEE Trans. on Pattern Anal. and Mach. Intel.*, 9:676–685, 1987.

[18] E. S. Ristad and P. N. Yianilos. *Learning String Edit Distances.* Research report cs-tr-532-96, Princeton University, 1996.

[19] D. Sankoff and J. B. Kruskal. *Time Warps, String Edits and Macromolecules: The Theory and practice of Sequence Comparison.* Addison-Wesley, 1983.

[20] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[21] E. Vidal and F. Casacuberta. A hybrid framework combining structural and decision-theoretic pattern recognitions and applications. *International Journal of Pattern Recognition and Artificial Intelligence*, 3:181–206, 1989.

[22] R. A. Wagner and M. J. Fischer. The string to string correction problem. *J. Assoc. Comput. Mach.*, 21:168–173, 1974.

School of Computer Science, Carleton University

Recent Technical Reports

TR-96-01 **Hierarchical Load Sharing Policies for Distributed Systems**
Sivarama Dandamudi and Michael Lo, January 1996

TR-96-02 **Randomized Parallel List Ranking for Distributed Memory Multiprocessors**
Frank Dehne and Siang W. Song, January 1996

TR-96-03 **Randomized Sorting on Optically Interconnected Parallel Computer**
G. Bhattacharya, J. Chrostowski, F. Dehne, P. Palacharla, January 1996

TR-96-04 **Authenticated Multi-Party Key Agreement**
Mike Just and Serge Vaudenay, February 1996

TR-96-05 **Boolean Routing on Cayley Networks**
Evangelos Kranakis and Danny Krizanc, February 1996

TR-96-06 **Secure Non-Interactive Electronic Cash**
Patrick Morin, February 1996

TR-96-07 **An Upper Bound for a Basic Hypergeometric Series**
Lefteris M. Kirousis, Evangelos Kranakis and Danny Krizanc, March 1996

TR-96-08 **A Formal Theory for Optimal and Information Theoretic Syntactic Pattern Recognition**
B.J. Oommen and R.L. Kashyap, March 1996

TR-96-09 **A Better Upper Bound for the Unsatisfiability Threshold**
Lefteris M. Kirousis, Evangelos Kranakis and Danny Krizanc, March 1996

TR-96-10 **Minimal Sense of Direction in Regular Networks**
Paola Flocchini, March 1996

TR-96-11 **Correlation Inequality and its application to a Word Problem**
Dimitris Achlioptas, Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Michael S.O. Molloy, March 1996

TR-96-12 **On Systems with Sense of Direction**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-13 **Computing on Anonymous Networks with Sense of Direction**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-14 **Symmetries and Sense of Direction in Labeled Graphs**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-15 **Distance Routing on Series Parallel Networks**
Paola Flocchini, Flaminia L. Luccio, April 1996

TR-96-16 **Maximal Length Common Non-intersecting Paths**
Evangelos Kranakis, Danny Krizanc, Jorge Urrutia, May 1996

TR-96-17 **Discrete Vector Quantization for Arbitrary Distance Function Estimation**
John Oommen, I. Kuban Altinel, Necati Aras, May 1996

TR-96-18 **Symmetry and Computability in Anonymous Networks: A Brief Survey**
Evangelos Kranakis, July 1996

TR-96-19 **Many-to-One Packet Routing via Matchings**
Danny Krizanc, Louxin Zhang, July 1996

TR-96-20 **Power Consumption in Packet Radio Networks**
Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, August 1996

TR-96-21 **Performance of Hierarchical Processor Scheduling in Shared-Memory Multiprocessor Systems**
Sivarama P. Dandamudi, Samir Ayachi, August 1996

TR-96-22 **Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems**
Michael Lo and Sivarama P. Dandamudi, August 1996

TR-96-23 **Performance Impact of I/O on Sender-Initiated and Receiver-Initiated Load Sharing Policies in Distributed Systems**
Sivarama P. Dandamudi and Hamid Hadavi, August 1996

TR-96-24 **Characterization of Domino Tilings of Squares with Prescribed Number of Nonoverlapping 2x2 Squares**
Evangelos Kranakis, September 1996

TR-96-25 **On the Impact of Sense of Direction on Communication Complexity**
Paola Flocchini, Bernard Mans, Nicola Santoro, October 1996

TR-96-26	**The Effect of Scheduling Discipline on Dynamic Load Sharing in Heterogeneous Distributed Systems** Sivarama P. Dandamudi, November 1996
TR-96-27	**Approximating the Unsatisfiability Threshold of Random Formulas** Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Yannis C. Stamatiou, November 1996
TR-96-28	**Paper foldings as chaotic dynamical systems** F. Geurts, November 1996
TR-96-29	**Compositional complexity in cellular automata: a case study** P. Flocchini, F. Geurts, November 1996
TR-96-30	**Compositional complexity in dynamical systems** F. Geurts, November 1996
TR-96-31	**Compositional experimental analysis of cellular automata: attraction properties and logic disjunction** P. Flocchini, F. Geurts, N. Santoro, November 1996
TR-96-32	**Approximating Weighted Shortest Paths on Polyhedral Surfaces** Mark Lanthier, Anil Maheshwari and Jörg-Rüdiger Sack, December 1996
TR-97-01	**Performance Comparison of Processor Scheduling Strategies in a Distributed-Memory Multicomputer System** Yuet-Ning Chan, Sivarama P. Dandamudi and Shikharesh Majumdar, January 1997
TR-97-02	**Performance Evaluation of a Two-Level Hierarchical Parallel Database System** Yifeng Xu and Sivarama P. Dandamudi, January 1997
TR-97-03	**Using Networks of Workstations for Database Query Operations** Sivarama P. Dandamudi, January 1997
TR-97-04	**A Performance Study of Locking Granularity in Shared-Nothing Parallel Database Systems** S. Dandamudi, S.L. Au, and C.Y. Chow, January 1997
TR-97-05	**Continuous Learning Automata Solutions to the Capacity Assignment Problem** B. John Oommen and T. Dale Roberts, April 1997
TR-97-06	**Discrete Learning Automata Solutions to the Capacity Assignment Problem** B. John Oommen and T. Dale Roberts, May 1997
TR-97-07	**The Swap-with-Parent Scheme: a Self-Organizing Sequential Search Algorithm which uses Non-Lexicographic Heaps** John Oommen and Juan Dong, May 1997
TR-97-08	**Time reversibility: a mathematical tool for creating arbitrary generalized swap-with-parent self-organizing lists** John Oommen and Juan Dong, May 1997
TR-97-09	**Designing Syntactic Pattern Classifiers Using Vector Quantization and Parametric String Editing** B. J. Oommen, R.K.S. Loke, May 1997
TR-97-10	**Performance of a Parallel Application on a Network of Workstations** A. Piotrowski and S.P. Dandamudi, May 1997

All the above-listed reports are available electronically. For further information,
(1) Access the school's worldwide web server at http://www.scs.carleton.ca.
(2) Click on to the 'Technical Reports' entry found on the main home page.
(3) Make sure that you have set the 'download to local disk' option on your mosaic or other client program. Click onto the appropriate technical reports and the postscript file will be downloaded onto your local disk.