Optimizing the Cost Matrix for Approximate String Matching using Genetic Algorithms*

Marc PARIZEAU†, Nadia GHAZZALI‡ and Jean-François HÉBERT§

printed July 29, 1997

Abstract

This paper describes a method for optimizing the cost matrix of any approximate string matching algorithm based on the Levenshtein distance. The method, which uses genetic algorithms, defines the problem formally as a discrimination between a set of classes. It is tested and evaluated using both synthetically generated strings of symbols and chain code data extracted from the international Unipen database of online handwritten scripts. Experimental results show that this approach can effectively discover the hidden costs of elementary operations in a set of string classes.

I Introduction

To represent patterns as strings of symbols is a well known approach in pattern recognition [1]. To discriminate between two patterns, one then needs to define a similarity (or dissimilarity) measure between two strings of symbols. One such measure, known as the Levenshtein distance [2] (also known as the edit distance), consists in counting the minimum number of elementary string operations that one has to apply to the first string in order to obtain the second one. These operations are usually defined as insertion, deletion and substitution of symbols [3].

Although several algorithms for computing the Levenshtein distance have been proposed to solve specific problems, they all require the definition of a cost matrix that specifies the costs of individual elementary operations for all combinations of symbols. For instance, the Wagner-Fischer [4] algorithm computes the Levenshtein distance between two ordinary strings and the more recent Myers-Miller [5] algorithm computes this distance between a string and a regular expression. Other recent contributions have been made by Bunke and Bühler [6] in translation/rotation invariant string matching and by Marzal and Vidal [7, 8] in normalized edit distance where the weight of the edit path is normalized by its length.

Thus, in order to use any one of these string matching algorithms, one always has to choose an adequate cost matrix and this selection may have a considerable impact on the recognition performance. Usually, the cost matrix is determined heuristically during system development by trial and error. Sometimes, prior information may guide this procedure but most of the time it consists in a blind search which can be far from optimal. To the authors' knowledge, only a single effort was made at selecting an optimal cost matrix but using severe constraints [9]. Indeed, in that approach, the cost matrix is constrained to unit cost for all insertion/deletion operations and to a single substitution cost r for all pairs of symbols. The Wagner-Fischer algorithm is then modified in order to compute an editing distance which is expressed in terms of r. However, this approach is intractable for more general cost matrices.

Common optimizing techniques (like gradient descent) cannot be used here because the function that we want to optimize, which depends on the editing distance, doesn't have an analytic form (it is an algorithm) and thus cannot be differentiated. The ap-

*This work will appear in *Pattern Recognition*, probably in 1997. It was supported in part by NSERC grant OGP0155389 to M. Parizeau and in part by NSERC grant OGP0155639 to N. Ghazzali.

†Marc Parizeau is with the département de génie électrique et de génie informatique, Université Laval, St-Foy (Québec), Canada, G1K 7P4. E-mail: *parizeau@gel.ulaval.ca*

‡Nadia Ghazzali is with the département de mathématiques et de statistique, Université Laval, St-Foy (Québec), Canada, G1K 7P4. E-mail: *ghazzali@mat.ulaval.ca*

§Jean-François Hébert was with the département of mathématiques et de statistique. He is now with the département de génie électrique et de génie informatique, Université Laval, St-Foy (Québec), Canada, G1K 7P4. E-mail: *jfhebert@gel.ulaval.ca*

proach that we propose here is based on genetic algorithms [10, 11] in order to circumvent this problem. The rest of this paper is subdivided as follows. Section II reviews briefly the concept of edit distances and defines formally the cost matrix of approximate string matching algorithms. Section III then proceeds in defining a fitness function that will serve for evaluating the fitness of a given cost matrix. This section also reviews briefly the concepts behind genetic algorithms and describes the details of our specific implementation. Finally, Section IV reports some experimental results that were obtained on synthetic and real data sets.

II Edit distances and cost matrix

Let Σ be a finite alphabet, Σ^* be the set of finite strings over Σ, and λ denote the null symbol. The function $\gamma_\Sigma : (\Sigma \cup \{\lambda\})^2 \longrightarrow I\!R^+$ then defines a cost matrix for elementary edit operations that can be used to transform a string $X \in \Sigma^*$ into a string $Y \in \Sigma^*$. If $a \rightarrow b$ denotes any such edit operation on X with $(a,b) \neq (\lambda,\lambda)$, then $\gamma_\Sigma(\lambda, b)$ is the cost of inserting symbol $b \in \Sigma$, $\gamma_\Sigma(a,\lambda)$ is the cost of deleting symbol $a \in \Sigma$ and $\gamma_\Sigma(a,b)$ is the cost of substituting a by b.

Using a sequence $E = e_1 \cdots e_i \cdots e_k$ of elementary edit operations ($e_i = a_i \rightarrow b_i$), one can obviously transform any X into any Y. For example, one trivial solution is to delete every character in X and insert every character in Y. An edit distance $\delta(X,Y)$ between X and Y is defined as the minimum cost sequence over all possible sequences of operations:

$$\delta(X,Y) = \min_E \{\Gamma(E)\} \tag{1}$$

where $\Gamma(E)$ is the sum of costs for the elementary edit operations of sequence E:

$$\Gamma(E) = \sum_{i=1}^{k} \gamma_\Sigma(a_i, b_i) \tag{2}$$

and $a_i \rightarrow b_i$ corresponds to operation e_i of E. Equation 1 formulates the basic string-to-string editing distance [4]. Other similar problems have similar formulations [5, 7] in the sense that the edit distance depends on a cost matrix γ_Σ that defines the costs of elementary operations.

For $\delta(X,Y)$ to define a distance in a metric space, γ_Σ must respect three of the four fundamental properties of a distance: $\forall (a,b) \in (\Sigma \cup \{\lambda\})^2$

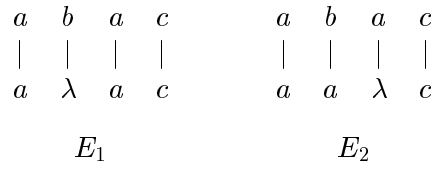

Figure 1: *Example of two different sequences (E_1 and E_2) of elementary operations for strings* abac *and* aac.

1. $\gamma_\Sigma(a,b) \geq 0$ guarantees that a sum of positive or null values will always result in a positive or null distance;
2. $\gamma_\Sigma(a,b) = 0 \Longleftrightarrow a = b$ states that a null distance implies identical symbols;
3. $\gamma_\Sigma(a,b) = \gamma_\Sigma(b,a)$ makes the distance symmetric.

The usual triangular inequality is not required here since it is implied in Equation 1. Indeed, if we proceed by contradiction, $\gamma_\Sigma(a,b) > \gamma_\Sigma(a,\lambda) + \gamma_\Sigma(\lambda,b)$ implies that deleting a and inserting b is always cheaper than substituting a for b; thus the latter is never part of the minimum cost sequence of operations. The reader should note that this situation is equivalent to assigning infinite cost to $\gamma_\Sigma(a,b)$.

Figure 1 illustrates two different sequences (E_1 and E_2) of elementary operations for strings *abac* and *aac*. Clearly, if we assume unit cost for all elementary operations, then the sequence E_1 is best since it requires a single operation (i.e. delete b in string *abac*) compared with two operations (i.e substitute b for a and delete the second a) for the sequence E_2. Thus $\Gamma(E_1) = 1 < \Gamma(E_2) = 2$. But is this assumption optimal? Unfortunately, there is no universal answer to this question since it depends on the application. In pattern recognition, the usual objective is to classify unknown patterns into coherent classes. So what do we know about the classes? If for instance symbol b is an essential part of the class to which string *abac* belongs, then the sequence E_1 which deletes symbol b with the same cost as any other operation is probably not optimal and thus the sequence E_2 might be better if the deletion cost for b is raised to 2.5 (i.e. $\Gamma(E_1) = 2.5 > \Gamma(E_2) = 2$). This example shows that structural information about the classes is needed for selecting the values of the cost matrix. But in general, this information is not directly available.

Counting relative frequencies of the symbols within the classes could be an approach, but it has an important limitation. Indeed, certain symbols may be present in each string of a particular class but their positions within the strings may vary considerably.

Thus they may not be the most stable and characteristic features of their class, and imposing high deletion costs for high frequency symbols can lead to high within-class distances. Furthermore, this scheme is of no help for selecting substitution costs. The alternative approach proposed in this paper consists in defining an objective function that seeks to minimize the within-class distances and simultaneously maximize the between-class distances. This objective function is then incorporated into a genetic algorithm.

III Cost matrix optimization

A genetic algorithm (GA) is essentially an evolutionary process that explores the solution space of a problem by breeding populations of chromosomes using stochastic reproduction and genetic operations like mutation and crossover [10, 11]. In this context, a chromosome is a simple sequence of genes that represent the parameters of the problem. If this sequence is interpreted as a vector in space, then the chromosome corresponds to a point in the solution space of the problem.

The basic genetic paradigm may be summarized in four steps as follows: 1) generate a random initial population of size N and evaluate the fitness of each chromosome; 2) generate a new population of the same size using a stochastic reproduction process biased towards the fittest chromosomes (this new population becomes the current one); 3) apply crossover and mutation operators and evaluate the fitness of each chromosome in the resulting population; 4) repeat step 2 for a specified number of generations or until some other stopping criterion is verified.

To apply this paradigm to our specific problem, four basic issues must be resolved: i) how to code a cost matrix as a chromosome; ii) how to implement reproduction and operate crossovers and mutations; iii) how to set the GA parameters (population size, probability of operators, etc.); and iv) how to evaluate the fitness of a cost matrix. The first three of these issues are easily addressed using very standard procedures (see for example [11], chap. 2); the details are given below. Point iv), which is most critical for this paper, is examined in the next subsection.

First, a cost matrix is coded by concatenating the bit representations of individual costs that are quantified on m bits over an interval $[0, x]$. Thus, for an alphabet of size $|\Sigma| = n$, we obtain strings of $\binom{n+1}{2} \times m$ bits, where $\binom{n+1}{2} = \frac{n(n+1)}{2}$ is the number of combinations for distinct parameters in the cost matrix. With this coding, generating the initial population simply consists in generating random sequences of bits. Second, reproduction is implemented by a selection process based on a roulette wheel constructed with slots sized proportional to relative fitness: the fitter the chromosome, the higher the probability of being selected (some chromosomes may be selected more than once). Then, selected chromosomes may be crossed with others with probability p_c and mutated with probability p_m. The crossover operator works on two chromosomes. It exchanges bits that lie to the right of some randomly chosen crossing point. The mutation operator, which is applied independently on each bit of each chromosome, consists in flipping the bit. Third, the following GA parameters have been chosen: population size $N = 50$, number of bits per gene $m = 10$ with quantization interval from 0 to $x = 5$, crossover probability $p_c = 0.3$, and mutation probability $p_m = 0.015$.

III.1 Fitness of a cost matrix

Let cost matrix γ_Σ be used for classification of a set $\mathcal{C}$ of p object classes:

$$\mathcal{C} = \{C_1, \ldots, C_i, \ldots, C_p\} \tag{3}$$

where each class C_i is represented by a set of q_i prototype strings:

$$C_i = \{\chi_{i1}, \ldots, \chi_{ik}, \ldots, \chi_{iq_i}\} \tag{4}$$

with χ_{ik} being the k^{th} prototype of class C_i.

In this context, optimizing γ_Σ for discriminating between the classes requires the definition of an objective function $\mathcal{F}$ that both minimizes the within-class distances and maximizes the between-class distances. The first such function that comes to mind, call it $\mathcal{F}_1$, is the ratio of the average between distance $\bar{\delta}_{ij}$ over the average within distance $\bar{\delta}_{ii}$:

$$\mathcal{F}_1 = \frac{2 \sum_{i=1}^{p} \sum_{j=i+1}^{p} \bar{\delta}_{ij}}{(p-1) \sum_{i=1}^{p} \bar{\delta}_{ii}} \tag{5}$$

where:

$$\bar{\delta}_{ii} = \frac{2}{q_i^2 - q_i} \sum_{k=1}^{q_i} \sum_{l=k+1}^{q_i} \delta(\chi_{ik}, \chi_{il}) \ , \tag{6}$$

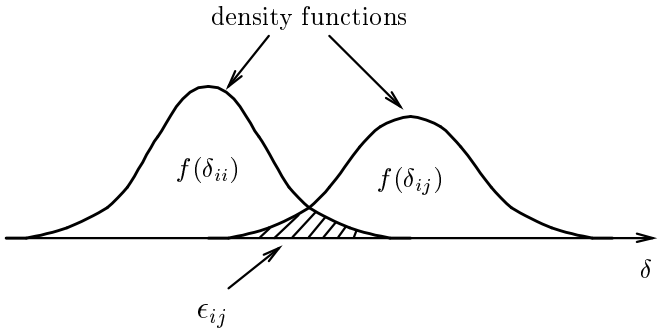

Figure 2: *Density functions of the within and between distances for classes i and j.*

$$\bar{\delta}_{ij} = \frac{1}{q_i q_j} \sum_{k=1}^{q_i} \sum_{l=1}^{q_j} \delta(\chi_{ik}, \chi_{jl}) \ . \tag{7}$$

However, this function has severe limitations because it makes no distinction between those classes that are already well separated and those that are overlapping. Consider for example Figure 2, which illustrates the density functions $f(\delta_{ii})$ and $f(\delta_{ij})$, corresponding to the within distance for class i and the between distance for classes i and j, respectively. Then, it is the classification error ϵ_{ij} between classes i and j which needs to be minimized since if $\epsilon_{ij} \to 0$, classes i and j need not be separated further. Assuming normal distributions for $f(\delta_{ii})$ and $f(\delta_{ij})$, ϵ_{ij} can be estimated using the Mahalanobis distance in the univariate case [12]:

$$\hat{\epsilon}_{ij} = \Phi\left[\frac{\bar{\delta}_{ii} - \bar{\delta}_{ij}}{2S_{ij}}\right] \tag{8}$$

where Φ is the distribution function of a standardized normal distribution, $N(0,1)$, $\bar{\delta}_{ii}$ and $\bar{\delta}_{ij}$ are the mean values for the within and between distances, and S_{ij} is the pooled standard deviation of the within and between distances:

$$S_{ij} = \sqrt{\frac{[q_i(q_i-1)-1]s_{ii}^2 + [q_i q_j - 1]s_{ij}^2}{[q_i(q_i-1) + q_i q_j - 2]}} \tag{9}$$

with s_{ii}^2 and s_{ij}^2 standing for the variances of the within and between distances, respectively. Using this estimated error, a second fitness function $\mathcal{F}_2$ can thus be proposed:

$$\mathcal{F}_2 = \frac{\sum_{i=1}^{p} \sum_{\substack{j=1 \\ j \neq i}}^{p} (1 - \hat{\epsilon}_{ij})}{p(p-1)} \ . \tag{10}$$

The reader should note that $\mathcal{F}_2$ takes its values in the interval $[0,1]$. Now the problem with $\mathcal{F}_2$ (as with $\mathcal{F}_1$) is that evaluating this function requires many string comparisons, especially when the number of classes and the average number of prototypes per class are great. Indeed, assuming that the number of prototypes per class is a constant equal to q, then a single evaluation of $\mathcal{F}_2$ requires exactly $\binom{pq}{2} = \frac{p^2q^2-pq}{2}$ string comparisons (i.e. for $p = 2$ and $q = 100$, $\binom{200}{2} = 19\,900$). A solution to this problem is to concentrate the evaluation process on a subset of the worst comparisons, that is, those within distances that are large and those between distances that are small. In other words, one may wish to try to separate only those prototypes that are the most overlapping.

Let $\mathcal{F}_3$ be yet another fitness function defined in a similar fashion as $\mathcal{F}_2$ but in which at most r string comparisons are made in order to estimate any of the distance density functions. Then, only $\binom{p+1}{2} \times r = \frac{p^2r+pr}{2}$ string comparisons will be needed for each evaluation of $\mathcal{F}_3$ (for $p = 2$ and $r = 30$, $\frac{p^2r+pr}{2} = 90 << 19\,900$) and this number is independent of the number of prototypes per class. Of course, all $\binom{pq}{2}$ comparisons will sometimes have to be made in order to determine, for each ordered couple (i,j) of classes, those r within prototype pairs for which the distances are maximum and those r between prototype pairs for which the distances are minimum. The main idea, however, is that this selection process can be made infrequently, in fact only when at the end of a generation, a new individual (cost matrix) achieves a maximum fitness. In this way, the GA always uses the most overlapping prototypes throughout the optimization process.

IV Experimental results

In order to evaluate the performance of the proposed fitness functions and GA for optimizing a cost matrix, experiments with both synthetic and real data will be described in this section.

IV.1 Synthetic data

The following procedure was used to generate synthetic strings of symbols:

1. Given an alphabet Σ, generate randomly a universal source string of length η;
2. Make a copy of the source string;
3. Scan this copy and replace each character with the result of a roulette wheel spin where slots are

sized proportional to some given between-class substitution-deletion probabilities for this character; this string becomes the source string for the current class;

(a) Make a copy of the source string for the current class;

(b) Scan this copy and replace each character with the result of a roulette wheel spin where slots are sized proportional to some given within-class substitution-deletion probabilities for this character; this string becomes a prototype for the current class;

(c) repeat steps 3a and 3b to generate q prototypes per class;

4. Repeat step 2 and 3 to generate p classes.

Using this procedure, five different data sets (DS1 to DS5) were generated. Their characteristics are summarized in Table 1 with p, q and η representing respectively the number of classes, the number of prototypes per class and the length of the source string, Σ being the alphabet, and P_{between} and P_{within} corresponding to the substitution-deletion probability matrices used for generating the class source strings and class prototypes.

DS1 is a data set that uses an alphabet of 4 symbols for which a uniform cost matrix should be optimal (i.e. equal cost for all operations except $\gamma_\Sigma(a,a) = 0, \forall a \in \Sigma$). Indeed, its prototypes were generated from essentially independent source strings for the two classes (i.e. between-class substitution probabilities are equiprobable: $\forall a,b \in \Sigma, P_{\text{between}}(a \to b) = 1/|\Sigma| = 0.25$) and uniform within-class substitutions (i.e $\forall a,b \in \Sigma, a \neq b, P_{\text{within}}(a \to b) = 0.08$). DS2 is another data set similar to DS1, but for which only three within-class operations are allowed with equal probability: deletion of symbols 1 and 2 (i.e. $P_{\text{within}}(1 \to \lambda) = P_{\text{within}}(2 \to \lambda) = 0.35$) and substitution of symbols 3 and 4 (i.e. $P_{\text{within}}(3 \to 4) = 0.35$). All other operations have zero probability. This data set should thus lead to a cost matrix were possible within-class operations have low cost and other operations have high cost. DS3 is a more difficult data set that uses an alphabet of 6 symbols and for which prototype strings are both longer and more numerous. For this data set, unlike for DS2, all substitution and deletion operations are possible although the following within-class operations are more probable than others: substitution of symbols 2 and 3, substitution of symbols 4 and 5 and substitution of symbols 1 and 6. DS3 should thus lead to a cost matrix where these three operations have the lowest cost. Finally, DS4 and DS5 are similar to DS3 except that they respectively contain three and six classes instead of only two (they also have shorter and less numerous strings).

GAs were run on all five data sets and for the three proposed fitness functions. Table 2 summarizes the results (in percentage of $\mathcal{F}_2$) obtained after a maximum of 300 generations (the runs were stopped when the average population fitness was non-increasing for more then 50 generations). Each γ_{opt} column is the fitness of the optimal matrix found over all simulated generations. Column γ_{unit} represents the fitness of the unit cost matrix (i.e. unit cost for every operation except $\gamma_\Sigma(a,a) = 0, \forall a \in \Sigma$) and serves as a reference value. Function $\mathcal{F}_3$ was optimized for three values of r (i.e. $r = 10$, $r = 20$ and $r = 30$). The reader should note that the fitness values in this table have all been computed using $\mathcal{F}_2$, although the GA runs were made using the function identified by the column heading. $\mathcal{F}_2$ was preferred for comparing cost matrix performances because this function is bounded between 0 and 1, and can easily be interpreted as an average separation rate of the classes.

The results in Table 2 show that $\mathcal{F}_1$ can lead to an optimal cost matrix for which, when compared with the result obtained using $\mathcal{F}_2$, the average separation rate between the classes ranges from very poor for DS2 to pretty good for DS3, DS4 and DS5. For the very poor case (DS2), the optimal cost matrix found is the following (the $-$ marks denote prohibited operations; values have been expressed in %; the lower half of the matrix is symmetric):

γ_{opt}	λ	1	2	3	4
λ	$-$	0.06	0.06	45.8	52.3
1		0	$-$	$-$	1.59
2			0	$-$	0.06
3				0	0.06
4					0

Although it can be seen that the three allowed operations for the generation of DS2, that is, $1 \to \lambda$, $2 \to \lambda$ and $3 \to 4$, all have very low costs (in fact minimum cost), two other unallowed operations also have very low cost: $1 \to 4$ and $2 \to 4$. To understand why $\mathcal{F}_1$ converged to this particular matrix, we looked at the distance matrices and found that, for the first prototype class, there is one string that, when compared with all others, requires the destruction of a

Table 1: *Data sets summary.*

	p	q	η	Σ	$P_{\text{between}}(a \to b)$	$P_{\text{within}}(a \to b)$
DS1	2	10	30	$\{1,2,3,4\}$	$\begin{cases} 0 & \text{if } b = \lambda \\ 0.25 & \text{otherwise} \end{cases}$	$\begin{cases} 0.68 & \text{if } a = b \\ 0.08 & \text{otherwise} \end{cases}$
DS2	2	10	30	$\{1,2,3,4\}$	same as DS1	$\begin{cases} 0.65 & \text{if } a = b \\ 0.35 & \text{if } a \in \{1,2\} \wedge b = \lambda \\ 0.35 & \text{if } a,b \in \{3,4\} \wedge a \neq b \\ 0 & \text{otherwise} \end{cases}$
DS3	2	30	50	$\{1,\ldots,6\}$	$\begin{cases} 0.90 & \text{if } a = b \\ 0.05 & \text{if } a,b \in \{1,2\} \wedge a \neq b \\ 0.05 & \text{if } a,b \in \{3,4\} \wedge a \neq b \\ 0.05 & \text{if } a,b \in \{5,6\} \wedge a \neq b \\ 0.01 & \text{otherwise} \end{cases}$	$\begin{cases} 0.75 & \text{if } a = b \\ 0.20 & \text{if } a,b \in \{2,3\} \wedge a \neq b \\ 0.20 & \text{if } a,b \in \{4,5\} \wedge a \neq b \\ 0.20 & \text{if } a,b \in \{1,6\} \wedge a \neq b \\ 0.01 & \text{otherwise} \end{cases}$
DS4	3	10	30	$\{1,\ldots,6\}$	same as DS3	same as DS3
DS5	6	10	30	$\{1,\ldots,6\}$	same as DS3	same as DS3

$\wedge$ is logical and

Table 2: *$\mathcal{F}_2$ values of the optimal (γ_{opt}) cost matrix obtained by GA when using fitness functions $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$ (results are in percentage and relative to $\mathcal{F}_2$).*

data set	γ_{unit}	$\mathcal{F}_1 \to \gamma_{opt}$	$\mathcal{F}_2 \to \gamma_{opt}$	$\mathcal{F}_3 \to \gamma_{opt}$		
				$(r = 10)$	$(r = 20)$	$(r = 30)$
DS1	91.9	88.4	94.4	85.5	91.9	94.7
DS2	92.8	69.3	99.5	99.7	99.5	99.3
DS3	91.8	96.8	98.6	96.7	97.4	98.4
DS4	91.1	95.6	98.8	99.1	98.8	99.3
DS5	88.5	96.0	97.1	97.3	96.7	96.7

symbol 3 even though this operation was not allowed in the within class probability matrix (the origin of this phenomenon stems from the limited precision of a float variable). Thus, although lowering the cost of operation $3 \to 1$ and $3 \to 2$ lowered both the within and between class distances, the effect on the average within distance was relatively more important. This shows that $\mathcal{F}_1$ can be very sensitive to noisy data because it will tend to draw further (closer) those between (within) prototype pairs that are easy to separate (bring closer), even when the underlying classes are already completely separated. By comparison, the optimal matrix found by $\mathcal{F}_2$ for DS2 is much less polarized:

γ_{opt}	λ	1	2	3	4
λ	–	2.3	11.0	15.1	23.9
1		0	–	3.6	23.3
2			0	–	19.4
3				0	1.4
4					0

Except for two cases, as expected, the allowed operations for DS2 have low costs, but not minimum cost, and other operations have relatively high, but not very high, costs. The two exceptions are $\gamma(2 \to \lambda) = 11$ and $\gamma(1 \to 3) = 3.6$. In order to test whether these values are specific to the DS2 data set or if, more generally, they model the intrinsic characteristics of DS2, 20 other data sets were generated using the same probability matrices that were used for DS2. Figure 3 shows the average separation rates for these data sets when using γ_{unit} (for reference), γ_{opt} found by $\mathcal{F}_1$ on DS2 and γ_{opt} found by $\mathcal{F}_2$ on DS2. This figure shows that, in this case, $\mathcal{F}_2$ produced a very robust cost matrix that separates almost perfectly (99.8%) the two classes even though the performance of the unit cost matrix varied greatly. As for the optimal matrix found by $\mathcal{F}_1$, its performance is always below that of the unit cost matrix and quite variable.

Table 2 also shows that the results of the $\mathcal{F}_3$ fitness function are not significantly different than those obtained with $\mathcal{F}_2$, even for a small subset of prototype pairs, while requiring approximately $\frac{q^2}{r}\left(\frac{p}{p+1}\right)$ times less string comparisons (where p is the number of classes, q is the number of prototypes per class and

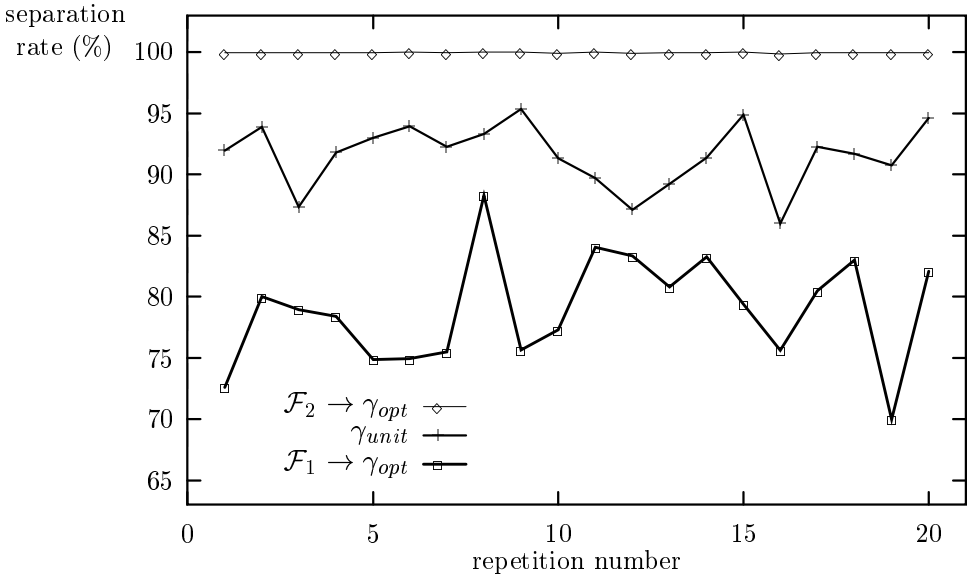

Figure 3: *Validation of the optimal matrices found by $\mathcal{F}_1$ and $\mathcal{F}_2$ on 20 repetitions of DS2.*

r is the number of comparisons per density function; we assume $pq >> 1$). For the case of DS3 and $r = 30$, this yields a fitness evaluation that is 20 times faster. In the case of DS5 and $r = 10$, it is 8.6 times faster.

For the case of DS3 optimized by $\mathcal{F}_2$, Figure 4 gives the plot of the maximum and average fitness of each of the first 150 generations of the GA. This figure shows that convergence is rapid: after approximately 50 generations, the average fitness does not increase anymore, although the maximum fitness is reached at the 111^{th} generation. Convergence is similar for all data sets.

IV.2 Real data

A recognition experiment was also conducted on real data extracted from the international Unipen on-line handwriting data sets [13] (release 1, version 5). Section 1A of this data set was chosen because it contains only isolated digits (a total of 6519 digits; ≈ 650 per digit). This data comes from many different sources (≈ 17), was written by many different writers from different countries, using different writing styles, and was sampled using many different digitizers. By any standard, the authors feel that this data set is a very difficult one. The digits were converted into chain codes using 8 directions. The alphabet has 9 symbols: 8 symbols for the 8 directions (symbols 1 through 8) and 1 special symbol for penlifts (symbol 0). The recognition algorithm is a very straightforward fuzzy K-nearest neighbor [14] with 10 prototypes per digit (see Figure 5). The distance metric is the well-known string-to-string editing distance [4]. The 10 prototypes per digit used for training the cost matrix were chosen randomly from samples of the data set. All other samples were used for testing.

The cost matrix was defined with a total of 13 parameters: 9 different insertion/deletion costs for the 9 symbols, and 4 different substitution costs, one for each possible change of direction (45°, 90°, 135° and 180°). No substitution was permitted for symbol 0 (penlifts). The GA for optimizing that cost matrix used fitness function $\mathcal{F}_3$ with $r = 30$ comparisons. All other GA parameters were the same as in the previous experiments. The GA was run for 150 generations and the following γ_{opt} matrix was found after the fifth generation (again the – represents prohibited operations; parameter values are expressed in percentage; the matrix is symmetric):

γ_{opt}	λ	0	1	2	3	4	5	6	7	8
λ	–	6.7	8.8	10.3	10.0	12.6	9.0	1.9	8.5	3.1
0		0	–	–	–	–	–	–	–	–
1			0	0.1	11.0	9.7	8.3	9.7	11.0	0.1
2				0	0.1	11.0	9.7	8.3	9.7	11.0
3					0	0.1	11.0	9.7	8.3	9.7
4						0	0.1	11.0	9.7	8.3
5							0	0.1	11.0	9.7
6								0	0.1	–
7									0	0.1
8										0

From this matrix, we can see on the one hand that the costs for inserting or deleting most symbols are similar except maybe for directions 6 and 8 that have

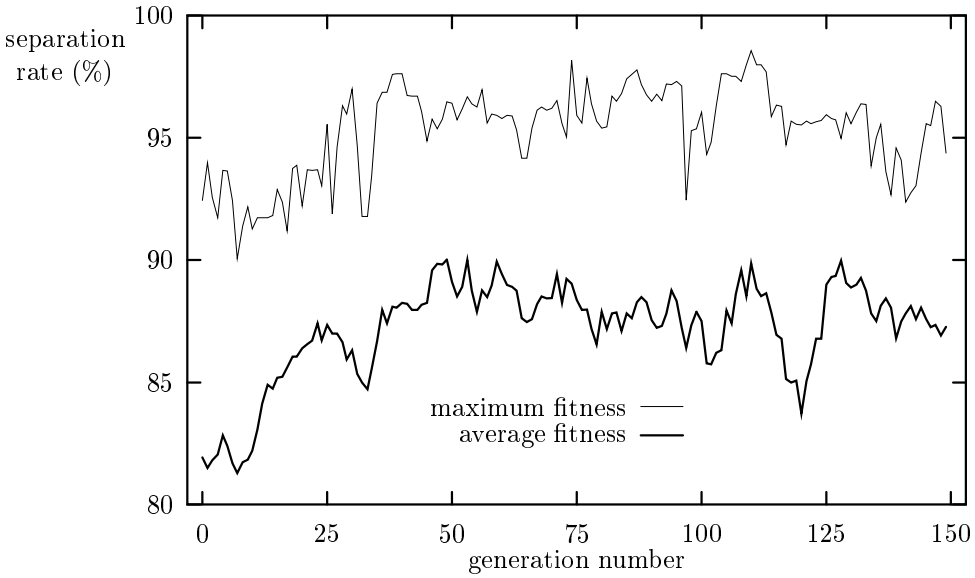

Figure 4: *Evolution of the maximum and average fitness for the GA run of DS3 using* $\mathcal{F}_2$.

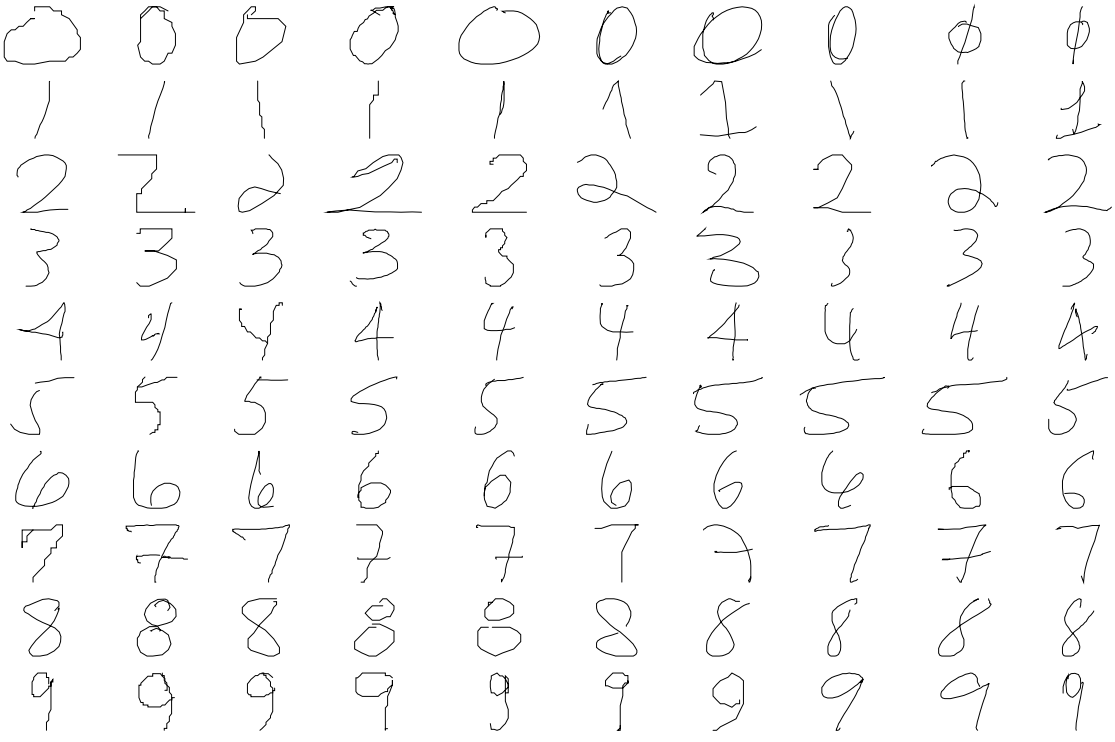

Figure 5: *Random prototypes used by the fuzzy k-nearest neighbor classifier.*

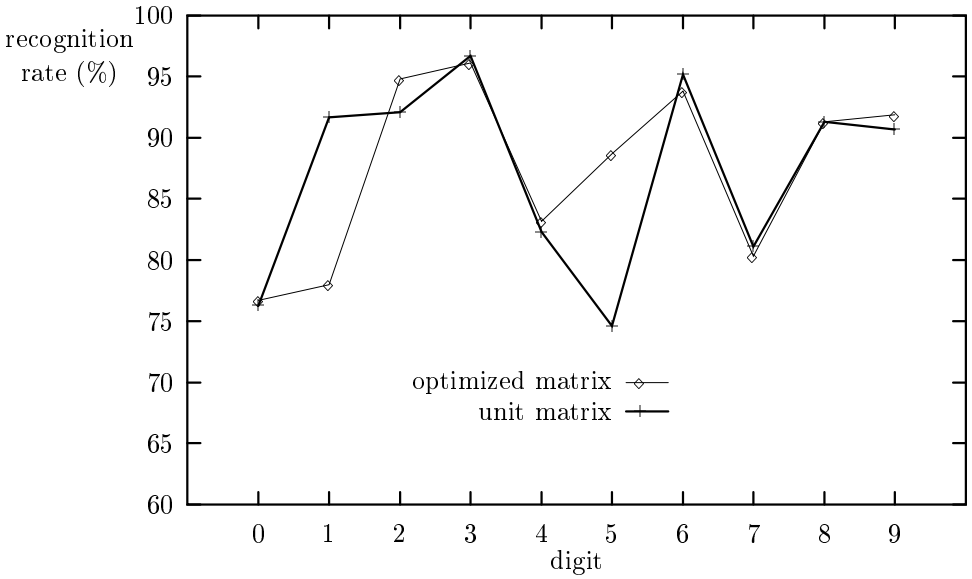

Figure 6: *Digit recognition rates when using 4 neighbors.*

somewhat lower costs (this difference might not be very significant). On the other hand, there is a very clear penalty for substituting non-adjacent directions (those that have an angle difference of 90°, 135° or 180°) compared with adjacent ones (those that have angle differences of 45°). For instance, $\gamma_{opt}(1 \rightarrow 3)$ is a hundred times higher than $\gamma_{opt}(1 \rightarrow 2)$. Thus, one may expect that this matrix should yield different (better?) recognition rates than those obtained using the γ_{unit} matrix. But this is not the case. In fact, these two matrices yield almost identical average recognition rates ($\gamma_{opt} = 87.5\%$ and $\gamma_{unit} = 87.2\%$) although individual rates for each digit (see Figure 6 for results with 4 neighbors) do show considerable fluctuations. In particular, the 1 digits are not well recognized by γ_{opt} and the 5 digits are not well recognized by γ_{unit}. This suggests that it may not be possible to find a single cost matrix for optimal discrimination of all digits in a difficult multi-writer context like the one of the Unipen database. And indeed, several very different matrices with equivalent fitness values and recognition rates were produced by the GA. In particular, one of these matrices was much better than γ_{opt} for recognizing the 1 digits but also much worse on the 2 and 5 digits (its average recognition rate was 87.4% compared with 87.5% for γ_{opt}). The most noticeable difference between that matrix and γ_{opt} is a higher substitution cost for adjacent directions.

When analyzing the overall results obtained with this digit recognition system, the reader should not forget that the objective was only to test the optimization method, not to design the best possible system. Nevertheless, the obtained performance (87.5%) is quite respectable considering the simplicity of the approach, and the fact that the prototypes were chosen randomly. Moreover, the authors estimate that approximately 5% of the digits in the testing data set are either mislabeled or very badly written. Thus, although the present experiment did not lead to an improved average recognition rate over what could be obtained with a unit matrix, it has still demonstrated the feasibility of the approach on a very difficult real data set were classes are mostly incoherent (i.e. they contain sub-classes where many within-class sample pairs may be just as dissimilar as between-class sample pairs). Furthermore, it has revealed an interesting property illustrated in Figure 7. This figure shows that the recognition rates for γ_{opt} seem more independent of the number of neighbors in the fuzzy nearest neighbor classifier than those of γ_{unit}. This probably stems from the definition of our evaluation function $\mathcal{F}_3$ whose associated optimization process takes into account both the within and between-class distance distributions, leading to a more robust discrimination.

V Conclusion

This paper has described an original approach for optimizing the cost matrix of any approximate string matching algorithm based on the Levenshtein distance. This approach uses a genetic algorithm to

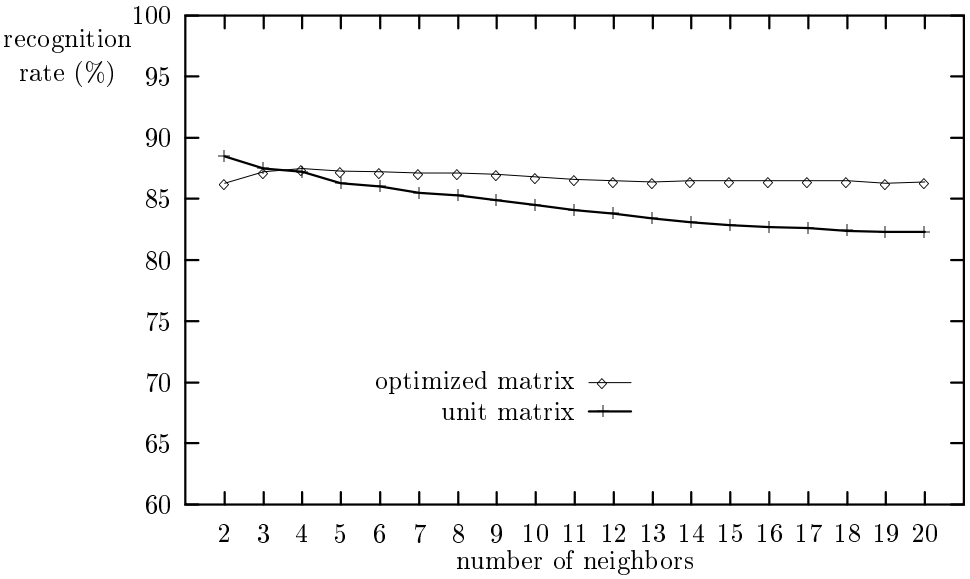

Figure 7: *Average recognition rates for different number of neighbors.*

efficiently explore the solution space using an objective evaluation function that measures the fitness or quality of a cost matrix. The problem is posed in the context of discriminating between a set classes. Three functions are proposed: the first computes the average between-class distance over the average within-class distance, the second estimates the average separation rate of classes using all within and between-class distances, while the third also estimates the average separation rate of classes but using only a small subset of the worst distances.

Experiments were conducted on both synthetically generated strings of symbols and on chain codes extracted from handwritten digits found in the international Unipen data sets. The synthetic strings were generated using variable within and between-class deletion/substitution probability matrices. Results have demonstrated that the proposed approach is both efficient and robust for discovering the hidden costs in those synthetic strings.

For the real chain code data, a simple recognition system was designed using a K-nearest neighbor classifier. With only 10 prototypes per class, an optimal matrix was obtained that correctly classified 87.5% (with K=4 neighbors) of the digits found in section 1A of the Unipen training data set. This system led to the observation that many different cost matrices can have almost identical discrimination power, especially if classes are non-homogeneous. This was the case for the Unipen digits. For instance, the optimal matrix found and the unit matrix gave almost equal average recognition rates for K=4 neighbors (87.5% versus 87.4%) although their individual digit recognition rates varied a lot. This suggests that if a single cost matrix cannot optimize simultaneously the recognition of each class, then maybe several distinct cost matrices should be looked for.

VI Acknowledgements

The authors would like to thank Christian Genest and the anonymous reviewers whose suggestions have greatly improved the presentation of this paper. This work was supported in part by NSERC grant OGP0155389 to M. Parizeau and in part by NSERC grant OGP0155639 to N. Ghazzali.

References

[1] R.C. Gonzales, M.G. Thomason, *Syntactic Pattern Recognition: An Introduction*, Addison-Wesley, 1978.

[2] A. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Sov. Phy. Dohl.*, Vol. 10, pp. 707-710, 1966.

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

[4] R.A. Wagner, M.J. Fischer, "The String-to-String Correction Problem", *Journal of the ACM*, Vol. 21, No. 1, pp. 168-173, 1974.

[5] E.W. Myers, W. Miller, "Approximate Matching of Regular Expressions", *Bulletin of Mathematical Biology*, Vol. 51, No. 1, pp. 5-37, 1989.

[6] H. Bunke, U. Bühler, "Applications of Approximate String Matching to 2D Shape Recognition", *Pattern Recognition*, Vol. 26, No. 12, pp. 1797-1812, 1993.

[7] A. Marzal, E. Vidal, "Computation of Normalized Edit Distance and Applications", *IEEE Trans. on Pattern Recognition and Machine Intelligence*, Vol. 15, No. 9, pp. 926-932, 1993.

[8] E. Vidal, A. Marzal, P. Aibar, "Fast Computation of Normalized Edit Distances", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-17, No. 9, pp. 899-902, 1995.

[9] H. Bunke, J. Csirik, "Inference of Edit Costs using Parametric String Matching", *Proc. of the 11th International Conference on Pattern Recognition*, The Hague, Vol. II, pp. 549-552, 1992.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[11] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

[12] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.

[13] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, S. Janet, "UNIPEN Project of On-Line Data Exchange and Recognizer Benchmarks", *Proc. of the 12th International Conference on Pattern Recognition*, Jerusalem, Vol. II, pp. 29-33, 1994.

[14] J.M. Keller, M.R. Gray, J.A. Givens, "A Fuzzy K-Nearest Neighbor Algorithm", *IEEE Trans. on System, Man and Cybernetics*, Vol. SMC-15, No. 4, pp. 580-585, 1985.