Copyright 2001 Springer-Verlag, reprinted with permission.

IJDAR (2001) 3: 150–159

International Journal on Document Analysis and Recognition IJDAR
© Springer-Verlag 2001

Experiments with adaptation strategies for a prototype-based recognition system for isolated handwritten characters

V. Vuori[1], **J. Laaksonen**[1], **E. Oja**[1], **J. Kangas**[2]

[1] Helsinki University of Technology, Laboratory of Computer and Information Science, P.O. Box 5400, 02015 HUT, Finland; e-mail: {vuokko.vuori,jorma.laaksonen,erkki.oja}@hut.fi
[2] Nokia Research Center, P.O. Box 100, 33721 Tampere, Finland; e-mail: jari.a.kangas@nokia.com

Received June 30, 1999 / Revised September 29, 2000

Abstract. This paper describes an adaptive recognition system for isolated handwritten characters and the experiments carried out with it. The characters used in our experiments are alphanumeric characters, including both the upper- and lower-case versions of the Latin alphabets and three Scandinavian diacriticals. The writers are allowed to use their own natural style of writing. The recognition system is based on the k-nearest neighbor rule. The six character similarity measures applied by the system are all based on dynamic time warping. The aim of the first experiments is to choose the best combination of the simple preprocessing and normalization operations and the dissimilarity measure for a multi-writer system. However, the main focus of the work is on online adaptation. The purpose of the adaptations is to turn a writer-independent system into writer-dependent and increase recognition performance. The adaptation is carried out by modifying the prototype set of the classifier according to its recognition performance and the user's writing style. The ways of adaptation include: (1) adding new prototypes; (2) inactivating confusing prototypes; and (3) reshaping existing prototypes. The reshaping algorithm is based on the Learning Vector Quantization. Four different adaptation strategies, according to which the modifications of the prototype set are performed, have been studied both offline and online. Adaptation is carried out in a self-supervised fashion during normal use and thus remains unnoticed by the user.

Key words: Handwriting recognition – Pattern recognition – Isolated characters – Unconstrained writing style – Online recognition – Intelligent user interface – Adaptation – Learning systems – k-nearest neighbor rule – Learning vector quantization – Dynamic time warping – Elastic matching

Correspondence to: V. Vuori

1 Introduction

Online recognition of handwritten text has been an ongoing research problem for four decades. It has been gaining more interest lately due to the increasing popularity of hand-held computers, digital notebooks, and advanced cellular phones. Traditionally, man-machine communication has been based on keyboard and pointing devices. These methods can be very inconvenient when the machine is only slightly larger or of the same size as human palm. A keyboard is very difficult to integrate in small devices and it usually determines the size of the whole apparatus. This is especially true when the number of characters is very large as in Chinese and Japanese. A pointing device, for example, a track ball or a pen, is insufficient or very slow when used alone and when also textual input is desired.

Due to these problems, new methods for input have been developed, for example, systems that recognize speech and handwriting. Because they both are very natural ways to communicate, people would easily learn to use them. Unfortunately, these recognition tasks are not that easy for computers, whose artificial intelligence is different from that of humans. Both recognition problems are still not completely solved. Very high accuracy is required for such systems before they could be commonly accepted. Handwriting recognition is a more attractive input method, especially in noisy environments and when privacy is needed.

Recognition accuracy is a key factor in determining the acceptability of a handwriting recognition system and the whole application in which it is implemented. The relationship between user acceptance of a pen interface and recognition accuracy is highly task-dependent according to experiments performed by Frankish et al. [1]. In general, the required recognition rate for handwriting recognizer is very high – even higher than humans can perform. Guyon and Warwick [2] have carried out experiments in which good typists wrote with a special keyboard that made random errors with a predefined rate. These tests showed that writers tolerate errors up to 1%, while 0.5% is not noticeable and 2% is intolerable.

These error rates are lower than error rates for human readers. An early experiment organized by Neisser and Weene [3], which involved 200 writers and nine readers, showed that the average human recognition error rate for isolated, handwritten, and context-free characters is 4.1%. Almost the same result, 4.0%, was obtained in a more recent experiment in which context-free characters from ten writers were recognized by ten readers. The best writer-reader couples had zero error percentages while the error rate for the worst couple was as high as 21% [4].

The most prominent problem in handwriting recognition is the vast variation in personal writing styles. There are also differences in one person's writing style depending on the context. The mood of the writer and the writing situation can have an effect on style. The writing style may evolve with time or practice, too. A recognition system should be insensitive to minor variations and still able to distinguish different, but sometimes very similar-looking, characters. Recognition systems should, at least in the beginning, be able to recognize many writing styles. Such multi-user systems usually have quite limited recognition accuracy. One way to increase performance is adaptation, which means that the system learns its user's personal writing style.

The work described in this paper concentrates on adaptation methods which are applied during normal use in a self-supervised fashion. The users are allowed to use their own natural styles of writing. In the beginning, the recognition performance of the system is relatively low, especially for people who have peculiar writing styles. However, after entering a few samples per each character, the error rate will be considerably decreased.

We have studied six dissimilarity measures and four adaptation strategies. First, a writer-independent recognition system was formed by finding the best combination of the preprocessing and normalization methods and dissimilarity measure. Next, the capabilities of the adaptation strategies to turn the writer-independent system into a writer-dependent one and to increase the recognition performance were examined.

2 Adaptive elastic matching classifier

2.1 General overview of the classifier

The classifier used in our work is based on prototype matching. The k-nearest neighbor (k-NN) rule [5] is used as a decision criterion. The k-NN rule is a data-driven approach and a completely parameter-free method, as it does not rely on any assumptions on the underlying probability distributions of the character classes. The classification is based only on the known classifications of the training or design samples which form the prototype set. The classification of an unknown character is made according to the majority of its k-nearest, or most similar, prototypes in the prototype set.

The classification rule applied in our recognition system resembles the condensed nearest neighbor (CNN) rule [6], a variant of the k-NN rule. The CNN rule classifies unknown patterns in a similar manner as the k-NN rule, but is based on a subset of the training samples instead of the whole training set. In our work, the subset of prototypes was selected by a clustering algorithm.

2.2 Prototype pruning and ordering

Computational savings can be achieved if the prototypes are pruned or ordered prior to the matching phase. In our work, connected parts of a drawn curve in which the pressure between the pen and writing surface exceeds a given value are considered as strokes. The prototypes are pruned according to their number of strokes. Arakawa [7] and Tappert [8] have used similar pruning methods successfully.

In addition to the pruning, computation time can be saved by ordering the prototypes so that distances to the well-matching prototypes will be evaluated in the early stages of the search, and distance-evaluation is interrupted as soon as it becomes clear that the prototype is not among the k best-matching ones.

Each character sample can be assigned to one of sixteen categories depending on the rough shape of its first stroke. The category is determined by the quadrants of the coordinate plane in which the starting and ending points of the first stroke are located. Prior to determining the category, the origin of the coordinate system has to be moved to the center of the character. A four-bit binary value is constructed for each category so that a change in the value of one of the bits corresponds to moving the starting or the ending point to a neighboring quadrant. The distance between two categories is then the count of bit differences in their binary representations.

2.3 Dissimilarity measures used in the matching and clustering of the characters

The number of data points per character is not fixed because the pen position is sampled with a constant sampling frequency and the size of the characters, writing speed, and style vary (see Fig. 1). Even if the pen point movements were resampled in order to get spatially equidistant data points, the number of data points would be nearly the same only for very similar-shaped strokes of the same size. Therefore, the metric used as a dissimilarity measure for the characters should be defined between curves which do not consist of the same number of data points. In addition, the dissimilarity measure should preferably be symmetric and independent of the actual number of data points.

All the six dissimilarity measures tested in the recognition system are based on the dynamic time warping (DTW) algorithm and thus are suitable for nonlinear curve matching [9]. They all are defined on stroke basis. If the number of strokes in two characters is different, the dissimilarity measure between the characters is defined to be infinite. Otherwise, it is the sum of the dissimilarity measures between the corresponding strokes. The main difference between the dissimilarity measures is the cost of matching a data point. All the matching costs and continuity and boundary constraints of the DTW-algorithm are symmetric. Therefore, all the dissimilarity measures used are symmetric. The DTW-algorithm determines the optimal matching of the data points which yields the minimum sum of the matching costs. The optimal matching of two strokes is defined by the optimal time-warping path. An example of such a matching is shown in Fig. 1. The DTW-algorithm is explained in more detail in [9].

The dissimilarity measures used in our work are called *Point-to-point*, *Normalized point-to-point*, *Point-to-line*, *Normalized point-to-line*, *Kind-of-area* and *Simple-area* distances. The continuity and boundary constraints of these dissimilarity measures are similar to each other. The continuity constraints require that the data points are matched in the same order as they have been produced. In addition, all data points are matched at least once and several data points can be matched against one. According to the boundary constraints, the first and last data points of the strokes are matched against each other or to lines interpolated between the first, or last, two data points.

Point-to-point distance uses the squared Euclidean distance between the data points as the matching cost. In the case of *Point-to-line* distance, the data points are matched to lines interpolated between the data points. The matching cost is the minimum squared Euclidean distance between the line and the point. All the data points, except the first and last of one of the strokes, are matched. *Normalized point-to-point* and *Normalized point-to-line* distances are otherwise similar to *Point-to-point* and *Point-to-line* distances, respectively, but the sums of the matching costs are divided stroke-wise by the number of matchings, i.e., the length of the warping path.

Kind-of-area(n,m) distance also matches data points against data points. However, the matching cost is a product of two terms. The first term is the nth power of the Euclidean distance between the data points. The second term is the mth power of the sum of the Euclidean distances from the matched data points to their neighboring data points. *Kind-of-area*(2,0) and *Point-to-point* distances are equivalent. As illustrated in Fig. 1, each matching of data points, except the first one, defines a new triangle or quadrilateral. *Simple-area* distance uses the areas of these polygons as the matching cost.

The main difference between the dissimilarity measures is their sensitivity to the dynamical variations of handwriting, such as speed and acceleration, and the timing of the sampling process. These factors affect *Point-to-point* distance most. *Point-to-line* distance is not so sensitive to the phase of sampling due to the interpolations. In case of *Normalized point-to-point* and *point-to-line* distances, the effects of dynamical variations are reduced by the normalization. The sensitivity of *Kind-of-area*(n,m) distance to the dynamical variations can be controlled with the parameter m. Provided that the sampling frequency is high, the area between two strokes does not depend much on the dynamics of the writing. Therefore, *Simple-area* distance depends the least on the dynamical properties of the strokes. These dissimilarity measures are discussed in more detail and their mathematical formulations are given in [10].

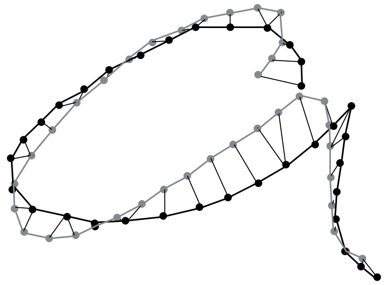

Fig. 1. The point-to-point correspondence of two characters established with a DTW-algorithm

2.4 Prototype selection algorithm

As the initial user-independent prototype set, only a subset of all training samples is used. This prototype set is formed by clustering training samples and selecting one sample from each cluster to present all samples in that cluster. The clustering algorithm is semiautomatic. This means that the number of clusters, or prototypes, must be predefined. In the current system, this number is the same for every class. The maximum number of different writing styles for a character class, which was seven, was found by manually examining the data and it is used as the total number of prototypes per class. The number of clusters per stroke number variation was selected so that it roughly corresponds to the respective share of all the writing styles of that character. For example, let us assume that there are 63, 27, and nine samples of some letter written with one, two, or three strokes, respectively. Thus, the numbers of prototypes assigned to different stroke number variations are 4, 2, and 1.

After deciding the number of clusters, training samples are divided into clusters by an automatic and iterative algorithm. The number of clusters is increased iteratively until the predefined count is reached or the samples run out. The iterative algorithm starts by finding the centers of the clusters, or in other words, the items which yield the minimum sum of dissimilarity measures between themselves and all other items belonging to the same cluster. Next, all the items are ordered into an increasing series $\{x_1, x_2, \ldots, x_N\}$ according to their distance to the center item of their cluster. A new cluster is formed of the items furthest from their cluster centers. The number of items in the new cluster is determined by minimizing splitting criterion function $J(i)$ which is defined in the following way:

$$J(i) = D(x_i, \overline{x}_{old}(i)) + \max_{i \leq j \leq N} D(x_j, \overline{x}_{new}(i)), \qquad (1)$$

where x_i is the item with order index i and $\overline{x}_{old}(i)$ is the old center item of its cluster, $\overline{x}_{new}(i)$ is the center item of the new cluster consisting of $\{x_i, \ldots, x_N\}$, and function $D(x, y)$ is the similarity measure between two items x and y. A new cluster is formed in the following way: if

$$J(i^*) = \min_i J(i), \tag{2}$$

items which have order index $i \geq i^*$ are included in the new cluster. Before the next round of the clustering algorithm, the cluster centers are iteratively recalculated and each item is assigned to the cluster center nearest to it. This is repeated until a stationarity partition is reached [11].

2.5 Adaptation strategies

The initial writer-independent prototype set must be modified so that it represents the user's writing style better. We have studied four different adaptation strategies for this. The ways of adaptations include: (1) adding new prototypes; (2) inactivating confusing prototypes; and (3) reshaping existing prototypes. The adaptation strategy *Add*(k) examines the classes of the k prototypes nearest to the input character. The classification is carried out according to the k-NN rule. The input character is added to the prototype set if any one of the k-nearest prototypes belongs to a wrong class, even if the classification was correct. Problems arise if the initial prototypes represent styles of writing which the writer uses but for different characters, or, if the writer uses similar writing styles for two different classes. In these cases, the adaptation adds new prototypes in vain: the neighborhood of the k-nearest prototypes consists of prototypes of very similar writing styles but different classes no matter how many new prototypes have been added.

The adaptation strategy *Inactivate*(N, G) is used for inactivating those prototypes which are more harmful than useful. Some character classes tend to be confused, for example 'g' and '9', as some writers write them in exactly the same way. If the prototype set includes prototypes very similar to characters written by the user but which belong to wrong classes, they should be removed. After each recognition, the *Inactivate*(N,G)-strategy checks whether the prototype nearest to the input character should be inactivated: if its goodness value g is below a given limit, G, and it has been the nearest prototype for at least N times, it is removed from the set of active prototypes. The goodness value is defined as follows:

$$g = \frac{N_{corr} - N_{err}}{N_{corr} + N_{err}}, \tag{3}$$

where N_{corr} and N_{err} are the numbers of times when the prototype has been the nearest one and its class has been correct or incorrect, respectively. Parameters G and N control the strictness and reaction rate of the inactivation rule, respectively. Reasonable values of G are between -1 and 1 and for N approximately 3.

When a character written by the user is basically similar to a prototype of the correct class, for example it has the same number and order of strokes, but it is of slightly different shape, the existing prototype can be reshaped instead of adding the input character to the prototype set. This can be performed with an adaptation strategy called *Lvq*(α) based on a modified version of Learning Vector Quantization (LVQ) [12]. If the nearest prototype belongs to the same class as the input character, the data points of the prototype are moved towards the corresponding points of the input character. When the classes differ, the points are moved in the opposite direction. The classification is carried out according to the 1-NN rule.

The traditional form of LVQ cannot directly be applied, as the prototype and input characters do not generally have the same number of data points. The point-to-point correspondence can, however, be established by using the DTW-algorithm. Suppose the first stroke consists of N_1 points $p_1(1), \ldots, p_1(N_1)$ and the second stroke of N_2 points $p_2(1), \ldots, p_2(N_2)$. The modified LVQ training is defined as follows [13]: let $P(h) = (i(h), j(h))$ be the optimal time-warping path between the strokes $S_1 = (p_1(1), \ldots, p_1(N_1))^T$ and $S_2 = (p_2(1), \ldots, p_2(N_2))^T$. The *Point-to-point* distance between the strokes is

$$D(S_1, S_2) = \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))), \tag{4}$$

where $d(p_1, p_2)$ is the squared Euclidean distance between two points and H is the total number of data point matchings in the optimal time-warping path. When the data points of stroke S_2 are moved, their new locations are given by

$$S_2^{new} = \begin{cases} S_2^{old} - \alpha \nabla_{S_2} D(S_1^{old}, S_2^{old}), & \text{if } \omega_1 \neq \omega_2 \\ S_2^{old} + \alpha \nabla_{S_2} D(S_1^{old}, S_2^{old}), & \text{otherwise} \end{cases}, \tag{5}$$

where α is a positive learning coefficient and ω_1 and ω_2 the classes of the input character and prototype. In addition,

$$\nabla_{S_2} D(S_1, S_2) = -2 \begin{pmatrix} \sum_{h=1}^{H} \delta(1, j(h))(p_1(i(h)) - p_2(1)) \\ \vdots \\ \sum_{h=1}^{H} \delta(k, j(h))(p_1(i(h)) - p_2(k)) \\ \vdots \\ \sum_{h=1}^{H} \delta(N_2, j(h))(p_1(i(h)) - p_2(N_2)) \end{pmatrix}, \tag{6}$$

where $\delta(i, j)$ is Kronecker's delta function.

The adaptation strategy *Hybrid* combines *Add*- and *Lvq*-strategies. The k nearest prototypes are examined. If any one of them belongs to the same class as the input character, the nearest prototype is modified with *Lvq*. Otherwise, the input character is added to the prototype set. In this way, the increase in the size of the prototype set should be moderate compared with the pure

Add-strategy. Nevertheless, the system is able to learn completely new writing styles in a similar way to the *Add*-strategy. This behavior cannot be obtained by only modifying existing prototypes with the *Lvq*-strategy.

3 Data used in the experiments

3.1 Data collection setup

All the characters were collected with a pressure-sensitive Wacom ArtPad II tablet attached to a Silicon Graphics workstation. The resolution of the tablet is 100 lines per millimeter and the sampling rate is at maximum 205 data points per second. The loci of the pen point movements consist of the x- and y-coordinates, the pen's pressure against the writing surface, and a time stamp. The writing area was a rectangle of size 50 mm × 20 mm placed at the center of the tablet. The characters were written one at a time. Writers were advised to use their natural handwriting style. All the data were saved in UNIPEN format [14]. The essential details of the databases are summarized in Table 1.

The first database (database 1) consists of characters which were written without any visual feedback: the trace of the pen was shown neither on the tablet nor on the screen. Therefore, the pressure level thresholding the pen movements into either pen-up or pen-down movements was set individually for each writer afterwards. Some of the characters were written in alphabetical order, but most of them were written according to the dictation of a short story. The distribution of the classes (a–z, A–Z, å, ä, ö, Å, Ä, Ö, 0–9, (,), /, +, −, %, $, @, !, ?, :, ., and ,) is somewhat similar to that of the Finnish language. This means that some of the lower-case letters, for example, 'a', 'i', 't', and 'n', are overrepresented when compared to upper-case letters and digits. However, at least two samples of each class were collected from each writer.

The rest of the databases (databases 2–4) were collected with a program which showed the pen trace on the screen and recognized the characters online. The minimum writing pressure for showing the trace of the pen on the screen and detecting pen-down movements was the same for all writers. The characters were written in the same random order by all subjects. The number of collected samples was the same for all the character classes (a-z, A-Z, å, ä, ö, Å, Ä, Ö, and 0-9). When a character was misrecognized, the writer had to determine whether the character was not properly written or whether the error should be considered as a failure of the recognizer. The user interface of the data collection program is illustrated in Fig. 2. Either of the two buttons in the bottom of the window labeled 'Writer Mistake' and 'Recognizer Mistake' was used after each misrecognition for indicating the type of the error. All the samples were checked and those which were reported as the writer's mistakes or clearly incorrect were abandoned. None of the writers of database 1 appeared in any of the databases 2–4.

Table 1. Summary of the databases used in the experiments

Data	*Subjects*	*Left-handed*	*Females*	*Items*
DB1	22	1	1	∼ 10 400
DB2	8	2	5	∼ 13 200
DB3	16	2	5	∼ 21 200
DB4	8	0	5	∼ 8 100

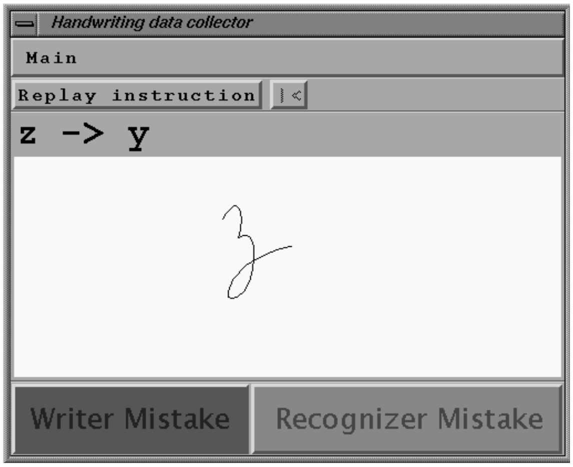

Fig. 2. The user interface of the data collection program

3.2 Preprocessing and normalization operations

Prior to clustering and matching, the characters were preprocessed. The preprocessing operations are very simple as they were mainly used for choosing a suitable sampling method and frequency. Characters were collected with a system whose properties, such as the sampling rate and resolution, are beyond the capabilities of the existing hand-held devices. Therefore, it is important to examine how sensitive the recognition method is to the amount of data for each character.

The original data consisted of sample points equidistant in time and the sampling rate was relatively high. The sampling frequency was altered with operations called *Decimate*(n) or *Interpolate*(n). *Decimate*(n) keeps every $(n+1)$th data point and abandons the intermediate ones. *Interpolate*(n) does just the opposite – it interpolates n equally spaced points between every original data point pair. The former operation reduces both the sampling rate and the amount of information in the data. The latter operation only increases the sampling rate as the data points added do not contain any additional dynamic information and are slightly misplaced from the actual smooth path of the pen. The data points can be made spatially equidistant with the *EvenlySpacedPoints*(d)-operation, where d is the requested distance between the adjacent points.

In Fig. 3, the original and two preprocessed versions of an example character are shown. It can be seen in the figure that the original sampling rate is at least sufficient because no peculiar corners appear when the adjacent data points are connected with lines.

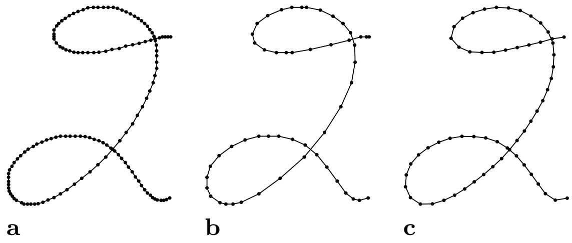

Fig. 3a. An original character and its two preprocessed versions when **b** *Decimate*(2), and **c** *EvenlySpacedPoints*(60) has been applied

The writers were allowed to write anywhere on the restricted writing area. Therefore, the unknown character and the prototypes must be moved to same location before they can be matched. This is carried out by moving their center points to the origin of the coordinate system. The normalization method *MassCenter* moves the mass center of the character to the origin, and the normalization method *BoundingBoxCenter* does the same according to the center of the character's bounding box. The size variations in the characters are normalized with an operator called *MinMaxScaling* which scales the size of the character so that the length of the longer side of the bounding box is set to 1 000 units while the aspect ratio remains unchanged. The *MinMaxScaling*-operation was performed in all the experiments.

4 Experiments and their results

4.1 Selection of the dissimilarity measure, normalization, and preprocessing operations

In the first experiments, the six dissimilarity measures were compared with each other without any prototype adaptation. The most suitable preprocessing and normalization operations were selected separately for each of them. These selections were based on the recognition performance of the system. The initial prototype sets were formed by using database 1 and the same dissimilarity measure, normalization, and preprocessing operations as in the recognition phase. As several parameter values were tried for the preprocessing methods and all the combinations of the preprocessing and normalization methods were tested with each of the dissimilarity measures, only the lower-case letters and digits of database 2 were used as a test set. This way, the recognition task was still challenging, and the experiments could be carried out in a reasonable time.

The best recognition results and the corresponding preprocessing methods are presented in Table 2. It can be seen that *Point-to-point* distance yields the lowest total error percentage E_{tot}. The best preprocessing method seems to be *Decimate*(2) which abandons two data points out of three and does not distort the dynamic information implicitly stored in the remaining data points. *MassCenter* is a better normalization method for all the dissimilarity measures. The implicit dynamic information is evidently of paramount importance, as data points distributed unevenly in space and the dissimilarity measure most sensitive to the point distribution yield the lowest error rate. In addition, the dissimilarity measure totally independent of the data point distribution, namely *Simple-area* distance, was significantly worse than the other measures. The fact that the most suitable value for the parameter m of *Kind-of-area* distance is zero also emphasizes the importance of the data point density.

Table 2. The lowest error percentage for each dissimilarity measure. The lower-case letters and digits of database 2 were used as a test set

Dissimilarity measure	*Preprocessing*	E_{tot}
Point-to-point	*Decimate*(2)	15.93
Point-to-line	*none*	17.07
Normalized point-to-point	*Decimate*(1)	16.13
Normalized point-to-line	*Decimate*(2)	17.07
Simple-area	*none*	29.75
Kind-of-area(1,0)	*Decimate*(2)	18.25

4.2 Comparison of the adaptation strategies

After choosing the best combination of dissimilarity measure, preprocessing, and normalization operations, several parameter values were tried for the adaptation strategies. Once again, all the experiments were carried out by using lower-case letters and digits. The initial prototype set was the same as used in the previous experiments. The characters of each writer in database 3 were recognized one by one and in the same order as they were collected. The adaptation was performed after each classification. The system was re-initialized for every user. The parameter values which yielded the lowest total error rates were selected. The recognition results were verified with writers in database 4.

The selected parameter values are presented in Table 3. The figures in the first four columns are the parameter values used for the adaptation strategies. The other figures are various error percentages. $E^{tot}_{des.}$ and $E^{tot}_{verif.}$ are the total error rates for all the characters of the design and verification set. $E^{final}_{verif.}$ is the error rate for the last 200 characters of each writer. From Table 3 it can be seen that all the adaptation strategies can improve the recognition accuracy significantly. The adaptation strategy *Add* yields the lowest total error. It reduces the total error rate from 14% down to 3%. However, the size of the prototype set increases considerably. The total error obtained with the *Hybrid*-strategy is also quite good, about 4%. The adaptation strategy *Lvq* is not sufficient when used alone as it cannot learn writing styles which are fundamentally different from those represented by the initial prototypes. Nevertheless, it is quite useful in combination with *Add*, as good recognition results can be obtained by using the *Hybrid*-strategy which adds only few new prototypes.

The probabilities that a new sample will be added to the prototype set are illustrated in Fig. 4 for both *Add*-

Table 3. Error percentages for the design set database 3 and the verification set database 4. Only lower-case letters and digits have been recognized

Add	*Lvq*	*Hybrid*	*Inact.*	$E^{tot}_{des.}$	$E^{tot}_{verif.}$	$E^{final}_{verif.}$
				14.33	14.13	14.06
4				2.93	3.12	1.81
	0.3			6.81	9.89	8.63
4			3,0	2.95	3.04	1.56
		3,0.3		3.06	4.18	2.50
		3,0.3	16,0	3.33	4.26	2.75

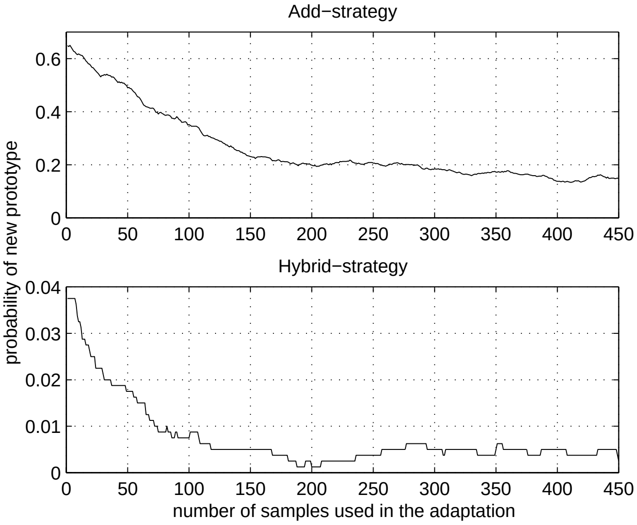

Fig. 4. Probability that new samples will be added to the prototype set when adaptation strategies *Add*(4) and *Hybrid*(3,0.3) are used. These results have been obtained by averaging the results for the eight writers of database 4. Only lower-case letters and digits have been recognized. Notice that the plots are scaled differently

and *Hybrid*-strategies. The probabilities are average values for all the writers of the verification set and were calculated by using a forwarded moving window of 100 samples. It can be seen from both plots that more prototypes are added at the beginning of adaptation. In the case of the *Add*-strategy, the initial probability is nearly 0.7 and at the end of the experiment it is still remarkably high, approximately 0.15. For the *Hybrid*-strategy, the initial probability is about 0.04 and the final probability is less than 0.01. As a result, the final number of prototypes with the *Hybrid*-strategy is only 2% higher than the initial, whereas for strategy *Add* it is 66% higher.

It should be noted that the probabilities mentioned above and shown in Fig. 4 are average values and there are considerable variations between character classes and writers. If the *Add*-strategy is applied and the user writes certain characters, say, '0' and 'O', in an identical way, new prototypes will be added for these character classes *ad infinitum*. On the other hand, for those character classes in which the writer uses distinct styles, no new prototypes will be added if there are already k prototypes. At the end of the adaptation, the easily confused character classes will be overrepresented. In the case of the *Hybrid*-strategy, this problem is less likely to arise as new prototypes will be added only until there is one good prototype for each character class and writing style. One way to avoid adding new prototypes and thereby increasing the recognition time in vain, is to set an upper limit for the number of prototypes per class.

The adaptation strategy *Add* applied together with *Inactivate* works better than *Add*-strategy used alone for the verification set. The effects of the *Inactivate*-strategy seem to be more prominent for the last 200 characters as the relative difference between the final error rates is larger than between the total error rates. However, the *Inactivate*-strategy is rarely used, in total, 40 times with the *Add*- and 11 times with *Hybrid*-strategy, and it does not really limit the size of the prototype set. This was not a totally unexpected result as bad samples had been removed from the databases.

4.3 Online experiments and nonadaptive simulations

In the collection of database 4, the *Add*(4)-strategy was applied. This can be considered as a genuine online test of that adaptation strategy. To evaluate the effects of the adaptation, the characters were also recognized offline with a nonadaptive system. These recognition results are presented and analyzed next. As the results were obtained with all the character classes, not only with digits and lower-case letters, they cannot be directly compared to the previous results. However, they give valuable insight into the problems which emerge when there are more classes. In addition, some of the classes, such as 'l' and '1', 'g' and '9', 'S' and '5','O' and '0', are quite easily confused even by humans when no contextual information is available.

The following observations were made on the basis of the experiments: (1) a low final error rate for the adaptive experiment does not necessarily follow from a good total error rate for the nonadaptive experiment. The initial accuracy of the recognizer is determined by the writer-independent prototype set. It is high for those writers whose writing style is covered by the initial prototype set. On the other hand, the benefits of the adaptation, and thus the final accuracy, depend on the consistency and nature of the writing style. On the average, adaptation was able to drop the error rate from 23% down to 4.0%; (2) the initial rate of learning is high if the initial accuracy is low; (3) the increase in the size of the prototype set is more prominent if the final error rate is high. Due to the adaptation, the average final number of prototypes was 1.8-fold compared to the initial number; (4) the writers cannot change their writing styles so that the characters would more closely resemble the initial writer-independent prototypes and thus be better recognized. On the contrary, the error percentage for the last 200 characters was on the average slightly higher than the total error rate for all characters.

Some interesting figures for both the true adaptive and simulated nonadaptive data collections are shown in Table 4. $E_{tot,A}$ and $E_{tot,NA}$ stand for the total error

Table 4. Error percentages for the true collection of database 4, in which adaptation strategy $Add(4)$ has been applied, and for the nonadaptive simulation

Writer	$E_{tot,A}$	$E_{tot,NA}$	$E_{final,A}$	$E_{final,NA}$
DB4:8	3.25	22.27	0.50	24.50
DB4:6	4.87	30.62	1.00	31.50
DB4:5	6.55	18.87	2.00	20.50
DB4:4	6.31	37.27	2.00	36.50
DB4:2	4.61	15.59	2.50	18.00
DB4:3	6.94	21.11	6.00	18.50
DB4:1	9.51	19.92	8.00	21.50
DB4:7	11.25	20.14	10.00	17.50
avg	6.66	23.19	4.00	23.56

Table 5. Percentages of different error types that occurred during the true collection of database 4 in which adaptation strategy $Add(4)$ has been applied (TC), and in the nonadaptive simulation (SC)

Test	E^d	E^l	E^u	E^1	E^2	E^3	E^4
TC	4.10	7.74	6.46	2.97	1.28	0.42	2.00
SC	12.80	25.06	24.93	7.86	4.23	2.75	8.34

percentages of the true and simulated data collections, respectively. $E_{final,A}$ and $E_{final,NA}$ are the error percentages for the last 200 characters. It can be seen that the variations between the writers are significant.

4.4 Analysis of typical errors

To obtain a deeper analysis of the recognition errors, the characters were divided into three groups: lower- and upper-case letters, and digits. The error rates shown in Table 5 are denoted with superscript d, u, and l for the digits, upper-, and lower-case letters, respectively. Abbreviations TC and SC stand for the true and simulated collections of database 4. All the error percentages are calculated over the whole test. Superscripts 1, 2, 3, and 4 correspond to different types of errors, namely: (1) a confusion between the lower- and upper- case versions of same letter; or (2) between a number and letter; or (3) between lower- and upper-case versions of different letters; and (4) all other mistakes.

From Table 5 it can be seen that, as a group, digits are best recognised and lower-case letters are worst recognized. The error probability for digits is naturally the lowest as the group contains the least classes. The relative proportions of misrecognitions in the three character groups (digits, lower-, and upper-case letters) were roughly the same for the adaptive (22%, 42%, and 35%, respectively) and nonadaptive (20%, 40%, and 40%, respectively) experiments. Therefore, the initial prototypes of the three character groups can be considered to be equally representative.

In both the adaptive and nonadaptive experiments, the 1-type error is the most probable of the errors in which two character groups are confused. This was not an unexpected result as many of the subjects used very similar shapes but different sizes for the lower- and upper-case versions of a letter and the size information is lost due to *MinMaxScaling*-normalization. The following character pairs were very prone to 1-type error in the adaptive experiments: 's' and 'S' (44 cases), 'z' and 'Z' (42), 'c' and 'C' (40), and 'x' and 'X' (36). The most typical examples of the 2-type error were: '1' and 'I' (11), '9' and 'g' (11), '2' and 'Z' (9), '1' and 'l' (9), and '0' and 'O' (8). The 3- and 4-type errors were more evenly distributed over the classes and thus typical errors cannot be clearly pointed out. The character pairs mentioned above for the adaptive experiments were also among the most troublesome in the nonadaptive experiments. The classification method and the adaptation strategy do not seem to be sufficient for separating these pairs – there is no sense in having and no gain in adding almost identical prototypes for two different classes.

4.5 Experiments with separate prototype sets

The easiest and commonly used way to avoid confusions between character groups is to demand that the user explicitly specifies the group in which the input character belongs. In such systems, the prototype sets of the groups can be kept separated. For example, specific input areas can be used for each character group. It is less convenient for the user but can be, at least partially, justified by the reduced error probability. In Tables 6 and 7, the results for simulated adaptive ($Add(4)$) and nonadaptive collections of database 4 are shown. It is now assumed that the group of the character is known prior to its recognition. All the notations have similar meanings as in the previous tables. In this case also, subscript *final* means that the error percentage is calculated for the last 200 samples of each writer.

The figures in Table 6 are the combined results of the three character groups and they can be compared with the figures in Table 4. It is evident that the use of separate prototype sets for the character groups is beneficial: the total error rate of the nonadaptive system drops from 23% to 9.8% and the final error rate of the adaptive system drops from 4.0% to 0.63%. In addition, the final number of prototypes is significantly lower: only 1.5-fold the initial. Specific results for the character groups are presented in Table 7.

The evolution of the error rate during the experiments is illustrated in Fig. 5. The lower plot corresponds to the adaptive test and the upper plot to the nonadaptive test. It can be seen from the lower plot that the error rate improves quite steadily during the first 300 characters and then fluctuates between 0.5% and 2%. These 300 characters were collected in approximately 20 min. Hence, on the average, a single character was prompted, written, and processed, and if requested, the type of the recognition error was declared in 4 s.

5 Conclusions

In this work, adaptation methods for a handwriting recognizer based on elastic matching have been studied.

Table 6. Simulated collection of database 4 in which adaptation strategy *Add*(4) and separate prototype sets for different character groups have been used. The error percentages have been calculated by combining the group-wise results

$E^{all}_{tot,A}$	$E^{all}_{tot,NA}$	$E^{all}_{final,A}$	$E^{all}_{final,NA}$
2.16	9.83	0.63	10.69

Table 7. Simulated collections of database 4 in which adaptation strategy *Add*(4) and separate prototype sets have been used. Error percentages for the three character groups are shown individually

$E^{d}_{tot,A}$	$E^{l}_{tot,A}$	$E^{u}_{tot,A}$	$E^{d}_{tot,NA}$	$E^{l}_{tot,NA}$	$E^{u}_{tot,NA}$
0.42	2.93	1.95	2.60	13.78	10.13

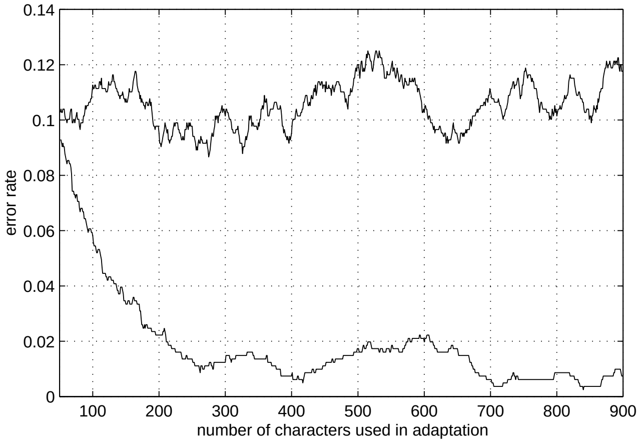

Fig. 5. Evolution of the error rate in the simulated collections of database 4 in which three separate prototype sets have been used for digits and lower- and upper-case letters. The lower plot is obtained with adaptation strategy *Add*(4) and the upper plot with a nonadaptive system

First, a writer-independent recognizer was constructed by choosing the combination of preprocessing and normalization methods and a dissimilarity measure most suitable for a set of writers. The major conclusion made in this phase of the work was that the implicit dynamic information, which is contained in data points sampled with a constant frequency, is beneficial. The recognition accuracy improves if not only the shapes of the characters but also the implicit dynamic features, such as the velocity and acceleration, of their generation are considered in the dissimilarity measure.

This is an interesting result, as explicit dynamic features are often used only for character or stroke segmentation and the data points are preprocessed so that they become evenly spaced in the spatial domain. In our experiments, the dissimilarity measure (*Point-to-point*) and centering normalization (*MassCenter*) most sensitive to the data point distribution were found to be better than the dissimilarity measures and centering normalization insensitive to the point density. These results speak for the benefits of the implicit dynamic information.

The main goal of the research was to study the adaptation of the system to the individual user. Several prototype set adaptation strategies were used for this. After adaptation, a recognition accuracy high enough to be acceptable for real-world applications was attained for most of the writers. Due to the adaptation, the recognition error rate for digits and upper- and lower-case letters was reduced from 23% down to 4%. In addition, when separate prototype sets were maintained for the three character groups, the final error rate was less than 1%. It should be noticed, however, that the data used in the experiments was collected from a fairly limited set of subjects and the recognition results varied considerably between the writers. In order to get more general results, the experiments should be repeated with a large public database.

An important phenomenon which can be observed in the true online test of the adaptation strategy *Add* is the user's adaptation to the performance of the recognition system. Instead of trying to improve the recognition results by themselves by writing more carefully and consistently, some of the users tend to demand more and more from the system as it learns and their writing styles become less careful. In such cases, the users probably had no other motivation than to get away from the tiresome test as soon as possible. User adaptation might be quite different in a more realistic experiment in which the user has to perform some real tasks, for example, write a letter, and the time consumed in the test directly depends on the recognition accuracy. It would be very interesting to test the recognition method and learn the users' opinions of it with a portable device and in realistic tasks. Before this, the supervision method of the learning process, the user interface, all error situations, and the system's sensitivity to bad learning samples must be considered more deeply. These are the topics of our recent work.

References

1. C. Frankish, R. Hull, and P. Morgan, "Recognition accuracy and user acceptance of pen interfaces," in Proceedings ACM CHI'95 Conference on Human Factors in Computing Systems, 1995
2. I. Guyon and C. Warwick, "Joint EC-US survey of the state-of-the-art in human language technology," http://www.cse.ogi.edu/CSLU/HLTsurvey.htm, 1996
3. U. Neisser and P. Weene, "A note on human recognition of hand-printed characters," Information and Control, no. 3, pp. 191–196, 1960
4. M. Parizeau and R. Plamondon, "Machine vs humans in a cursive script reading experiment without linguistic knowledge," in Proceedings of International Conference on Pattern Recognition, 1994, vol. 2, pp. 93–98
5. E. Fix and J.L. Hodges, "Discriminatory analysis–nonparametric discrimination: Consistency properties," Tech. Rep. Number 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas, 1951
6. P.E. Hart, "The condensed nearest neighbor rule," IEEE Transactions on Information Theory, vol. 14, no. 3, pp. 515–516, May 1968

7. H. Arakawa, "On-line recognition of handwritten characters – alphanumerics, Hiragana, Katakana, Kanji," Pattern Recognition, vol. 16, no. 1, pp. 9–16, 1983
8. C.C. Tappert, "Speed, accuracy, and flexibility trade-offs in on-line character recognition," International Journal of Pattern Recognition and Artificial Intelligence, vol. 5, no. 1&2, pp. 79–95, 1991
9. D. Sankoff and J.B. Kruskal, Time warps, string edits, and macromolecules: the theory and practice of sequence comparison, Addison-Wesley, 1983
10. Vuokko Vuori, "Adaptation in on-line recognition of handwriting," M.S. thesis, Helsinki University of Technology, 1999
11. Jorma Laaksonen, Vuokko Vuori, Erkki Oja, and Jari Kangas, "Adaptation of prototype sets in on-line recognition of isolated handwritten Latin characters," in Advances in Handwriting Recognition, Seong-Whan Lee, Ed., pp. 489–497. World Scientific Publishing, 1999
12. Teuvo Kohonen, Self-Organizing Maps, vol. 30 of Springer Series in Information Sciences, Springer-Verlag, 1997, Second Extended Edition
13. Jorma Laaksonen, Jarmo Hurri, Erkki Oja, and Jari Kangas, "Comparison of adaptive strategies for on-line character recognition," in Proceedings of International Conference on Artificial Neural Networks, 1998, pp. 245–250
14. I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "Unipen project of on-line data exchange and recognizer benchmark," in Proceedings of International Conference on Pattern Recognition, 1994, pp. 29–33

Vuokko Vuori received her M.SC. degree in system analysis and operation research from Helsinki University of Technology, Finland, in 1999. Currently, she is a graduate student and working as a researcher at the Laboratory of Computer and Information Science in the same university. Her research interests are pattern recognition and usability issues in general, emphasis being on adaptive handwriting recognition methods. MS Vuori is the secretary of Pattern Recognition Society of Finland.

Jorma Laaksonen received his D.Sc. degree in 1997 from Helsinki University of Technology, Finland, where he is presently Senior Research Scientist at the Laboratory of Computer and Information Science. He is an author of several journal and conference papers on pattern recognition, statistical classification, and neural networks. His research interests are in content-based image retrieval and recognition of handwriting. Dr. Laaksonen is a founding member of the SOM and LVQ Programming Teams, PicSOM Development Group, and a member of the International Association of Pattern Recognition (IAPR) Technical Committee 3: Neural Networks and Machine Learning.

Erkki Oja received his D.Sc. degree in 1977 from Helsinki University of Technology, Finland, where he is presently Professor of Computer Science and Director of the Neural Networks Research Centre. His research interests are in the study of principal components, independent components, self-organization, statistical pattern recognition, and applying artificial neural networks to computer vision and signal processing. Dr. Oja is an IEEE Fellow, IAPR Fellow, and President of the European Neural Network Society. He is member of the editorial boards of several journals, including "Neural Computation", "IEEE Transactions on Neural Networks", and "Int. Journal of Pattern Recognition and Artificial Intelligence".

Jari Kangas received his M.Sc. degree in computer science from Helsinki University of Technology, Espoo, Finland, in 1988, and his D.Sc. degree from the same university in 1994. He is currently a R&D manager and Principal Scientist at Nokia China R&D Center, Beijing, China. His research interests are in pattern recognition in general, emphasis being in methods and techniques to enhance the User Interface functions of mobile terminals by using, for example, handwriting recognition, speech recognition and image analysis.