Improving the Clustering Performance of the Scanning n-Tuple Method by Using Self-Supervised Algorithms to Introduce Subclasses

George Tambouratzis, *Member*, *IEEE*

Abstract—In this paper, the scanning n-tuple technique (as introduced by Lucas and Amiri [1]) is studied in pattern recognition tasks, with emphasis placed on methods that improve its recognition performance. We remove potential edge effect problems and optimize the parameters of the scanning n-tuple method with respect to memory requirements, processing speed, and recognition accuracy for a case study task. Next, we report an investigation of self-supervised algorithms designed to improve the performance of the scanning n-tuple method by focusing on the characteristics of the pattern space. The most promising algorithm is studied in detail to determine its performance improvement and the consequential increase in the memory requirements. Experimental results using both small-scale and real-world tasks indicate that this algorithm results in an improvement of the scanning n-tuple classification performance.

Index Terms—n-tuple pattern recognition method, scanning n-tuple, chain-coding, handwritten character recognition.

1 Introduction

Currently, considerable research effort is aimed at recognizing handwritten characters with a high accuracy. An important parameter in character recognition is the response time in order to 1) allow fast processing of large amounts of data and 2) provide a real-time response, which is essential in online applications. Due to the large variability of the data, several of the methods proposed recently are based on neural networks [2]. Among neural network models, the n-tuple method [3] stands out due to its readily implementable structure using RAM-type (Random Access Memory) digital circuits and its speed. The n-tuple method (hereafter named standard n-tuple) is a statistical pattern recognition method, which decomposes a given pattern into u sets of n points, termed n-tuples, and performs elementary recognition tasks on each of these sets. The results of these tasks are then combined to generate the recognition decision for the entire pattern. Considerable research activity has recently focused on the n-tuple method, both regarding theoretical issues (for example, [4] and [5]) as well as applications to real-world tasks. Several applications of n-tuple-based networks to handwritten character recognition tasks have been reported. Jung et al. [6] propose a method for selecting an optimal set of n-tuples in order to recognize classes of characters. Jorgensen [7] employs the standard n-tuple classifier incorporating negative weights to perform character recognition tasks. In contrast, Lucas and Amiri [1] propose an extension to the n-tuple method named the scanning n-tuple (hereafter termed the sn-tuple), which relies on a very small number of n-tuples repetitively scanning the input pattern. The n-tuple has been applied to image recognition tasks such as fingerprint recognition, document analysis, and texture classification ([8] provides a summary of recent research). So far, the scanning n-tuple has been applied to hand-written character recognition, though it is also expected to be applicable to general two-dimensional shape recognition and classification tasks.

This article focuses on the sn-tuple approach, in an effort to devise methods that improve its classification accuracy as compared to the results obtained so far. To this end, the pattern space defined by the training set is studied in detail, to investigate how significant overlaps between pattern classes affect the classification performance and, in particular, whether one or more classes may be automatically divided into subclasses to improve the classification performance. The results of this approach are found to be of an equivalent quality to those obtained when utilizing knowledge concerning the specific pattern space to manually split the classes into subclasses that optimize the classification performance.

In the following section, the standard and scanning n-tuple techniques are compared and contrasted. The character classification task that is used as a case study is presented in Section 3, while, in Section 4, experimental results for the standard and sn-tuple methods are described and the pattern space characteristics are examined in detail. Methods for improving the sn-tuple technique are presented in Section 5 and, in Section 6, the most promising method is selected and

• *The author is with the Institute for Language and Speech Processing, Artemidos & Epidavrou Street, Paradissos Amaroussiou, 151 25, Athens, Greece. E-mail: giorg_t@ilsp.gr.*

Manuscript received 8 Aug. 2000; revised 13 Mar. 2001; accepted 24 Oct. 2001.
Recommended for acceptance by C. Brodley.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 112679.

studied in detail. The article is concluded in Section 7 with a discussion of the results obtained.

2 Pattern Recognition via the n-Tuple Technique

2.1 The Standard n-Tuple Method

In order to classify an input pattern $\underline{p}$, the standard n-tuple method [3] relies on decomposing it into u sets of n points (termed n-tuples). In the given context of character recognition, $\underline{p}$ is a binary character image and the sets of n points are sets of binary pixels. For each pattern class to be recognized, one discriminator node is provided, which consists of a set of n-tuples. For each n-tuple, 2^n memory locations are allocated, one for each combination of pixel values, assuming each pixel can take on a value from the set {0,1}. This set of 2^n locations is collectively termed a *function*. In the nonweighted n-tuple method, each memory location stores a binary number, indicating whether the corresponding combination occurred within the training data. This article shall focus on the weighted n-tuple method, in which the number stored is an integer representing the frequency-of-occurrence of the combination. The maximum integer value is bounded by the size in bits of the memory location, which poses limits on its storage capacity. To index the memory location $a_{j,l}$ of a particular set of points $\{e_{1,j}, e_{2,j}, \ldots, e_{n,j}\}$ for the jth n-tuple of node l, we compute:

$$a_{j,l} = \sum_{i=1}^{n} p(e_{i,j}) * \sigma^{i-1}, \quad (1)$$

where $p(e_{i,j})$ denotes the value of point $e_{i,j}$ for the current pattern and σ is the number of possible values of each point (in the case of binary pixels, $\sigma = 2$). Training consists of updating the correct memory locations for each n-tuple for the given patterns. During classification, the contents of the memory locations $a_{j,l}$ are recalled (forming the results of elementary recognition tasks) and are then added to generate the final classification result. Thus, the recognition task is transformed into u elementary recognition tasks, each performed on the basis of the corresponding n points. The response r_l of node l (dedicated to recognizing class l) is:

$$r_l = \sum_{j=1}^{u} content(a_{j,l}). \quad (2)$$

Then, the pattern is classified as belonging to class c, where the cth node is the node with the highest response (as determined by formula (2)) among all the nodes in the network.

The weighted n-tuple approach has been coupled with an adaptive learning algorithm to create a self-organizing network which, when presented with patterns, is able to cluster them autonomously. This network is able to determine autonomously the classes existing in the pattern space with a high degree of accuracy, as reviewed in [9].

2.2 The Scanning n-Tuple Method

2.2.1 The sn-Tuple Method and Chain-Coding

While, in the standard n-tuple method, the sets of n points (the n-tuples) are fixed, in the scanning n-tuple [1], a mask $m = \{m_1, m_2, m_3, \ldots, m_n\}$ is used as a template to determine the sampled n points. The points in the mask are situated at a distance of f points from each other. Though this distance may vary over the mask, experiments will concentrate on a constant f (hereafter referred to as offset). The mask is applied repetitively to all points of the chain-code, in order to sample the entire pattern. The choice of a given mask determines the contents of the corresponding sn-tuple. Hence, to obtain sn-tuples reflecting different pattern properties, a different mask needs to be selected for each sn-tuple.

The application of the sn-tuple technique to character recognition tasks presupposes the chain-coding of the character patterns. For each pixel transition, the chain-code can have one of σ values corresponding to directions situated at angles of $(360/\sigma)$ degrees to each other. The chain-coding transformation consists of starting at a given point of the character area (in the present implementation, the upper-left corner) and traversing the character perimeter until the whole character is covered. This results in a one-dimensional vector of k elements

$$\underline{c} = \{c(1), c(2), \ldots, c(x), \ldots, c(k)\},$$

where $1 < x < k$ (and k may differ for each character). These elements are then encoded in a standard binary format, generating a vector of binary values. For instance, for $\sigma = 8$, each element is converted into a sequence of three binary values which are subsequently processed as an ensemble, while allowing the use of the n-tuple principle.

By its very nature, chain-coding results in a variable code length since the introduction or deletion of a single pixel in the original pattern results in the alteration of the code length. In our experiments, the chain-code length of all characters varies from 66 elements to 414 elements. The variable length prevents the application of the standard n-tuple method to chain-coded patterns, due to the need to prespecify the input vector dimensions (the chain-code length), so that each function samples a fixed part of that vector. In contrast, the scanning n-tuple can be applied in a straightforward manner to variable-size input. Each sn-tuple instance is defined as:

$$\begin{aligned} \underline{t_j} = \{c(e_{1,j}), c(e_{2,j}), \ldots, c(e_{n,j})\}, &\text{ where for } \forall i, \\ 1 < i \leq n : (e_{i,j}) = (e_{i-1,j} + f). & \end{aligned} \quad (3)$$

The first and last applications of the mask m scan the tuples $\underline{t_{1,j}}$ and $\underline{t_{k-(n-1)*f,j}}$, respectively:

$$\begin{aligned} \underline{t_{1,j}} &= \{c(1), c(1+f), c(1+2 \cdot f), \ldots, c(1+(n-1) \cdot f)\} \\ \underline{t_{k-(n-1)*f,j}} &= \{c(k-(n-1) \cdot f), c(k-(n-2) \cdot f), \\ &\quad c(k-(n-3) \cdot f), \ldots, c(k)\}. \end{aligned} \quad (4)$$

For a given position of the mask on the chain-code, the possible combinations of the sn-tuple are σ^n, where σ is the

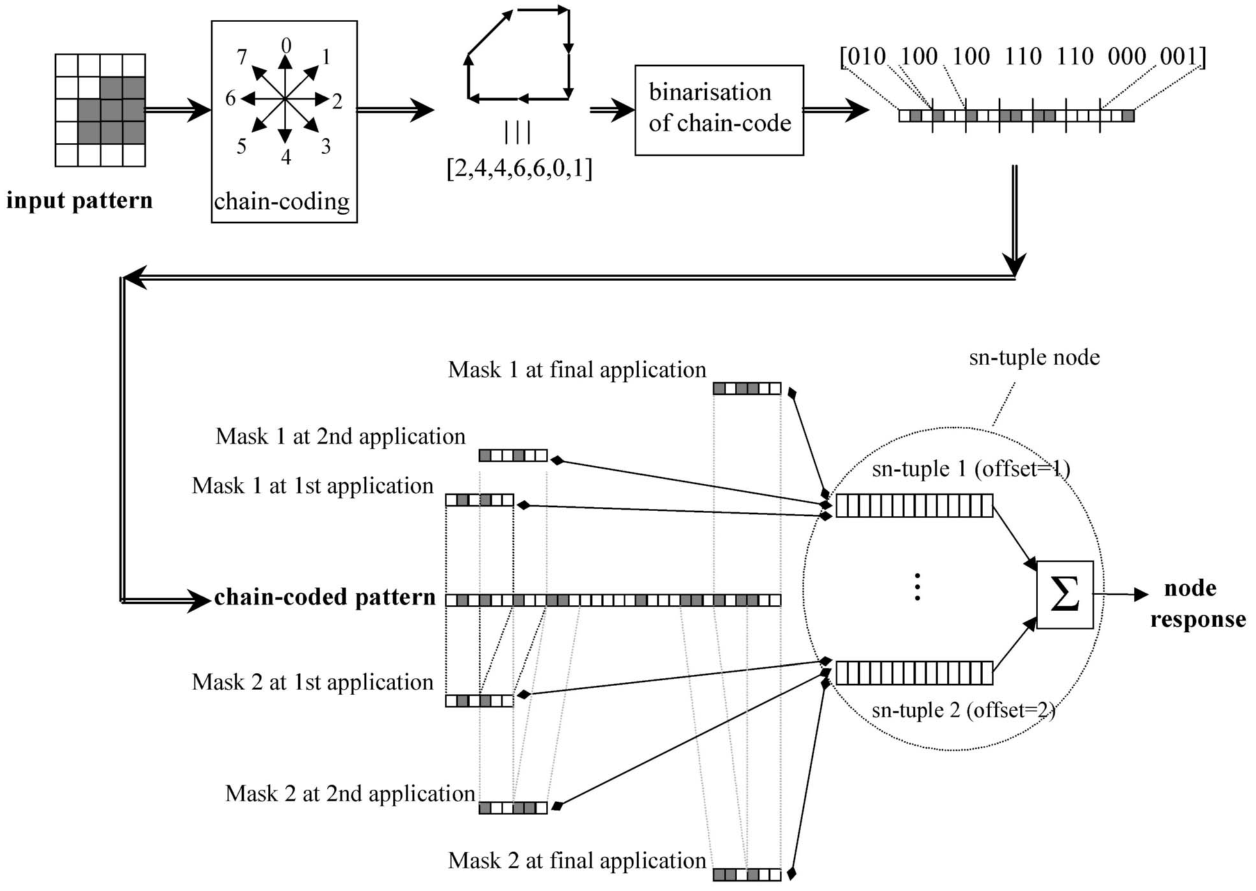

input pattern
chain-coding
7 0 1 6 2 5 4 3
[2,4,4,6,6,0,1]
binarisation of chain-code
[010 100 100 110 110 000 001]
Mask 1 at final application
Mask 1 at 2nd application
Mask 1 at 1st application
chain-coded pattern
Mask 2 at 1st application
Mask 2 at 2nd application
Mask 2 at final application
sn-tuple node
sn-tuple 1 (offset=1)
sn-tuple 2 (offset=2)
Σ
node response

Fig. 1. Schematic representation of the classification operation for a node consisting of two sn-tuples, with offsets of one (consecutive points being sampled) and two (points with a distance of 2 being sampled). Initially, the input character is chain-coded using σ directions. The chain-coded pattern is binarized and then sampled by a sliding n-tuple mask (since here $n = 2$ and $\sigma = 8$, two triplets of binary pixels are sampled). The mask is moved to the right by one point at a time, to cover the entire chain-code. For each mask position, the corresponding sn-tuple is accessed giving a partial recognition response. The partial responses for all sn-tuples of the node over all mask positions are summed to generate the node response to the input character.

number of possible directions used in chain-coding (in our experiments, σ is equal to eight) and n is the number of points sampled by the sn-tuple. Consequently, for each sn-tuple, a set of σ^n memory locations are provided where information reflecting the frequencies-of-occurrence is stored during training. Actually, the logarithm of the frequency is stored, as in [1], in order to make the summation of tuple outputs correspond to a multiplication rule (thus assuming that each n-tuple contributes independent likelihood information to the final recognition result).

The address determined by each mask application according to the chain-code values is:

$$a_{j,l} = \sum_{i=1}^{n} c(e_{i,j}) * \sigma^{i-1}. \tag{5}$$

For example, if an sn-tuple (where $n = 2$) mask samples 8-valued directions 1 and 4, (equivalent to binary-valued vector "001 100"), the location addressed shall be $2^3 + 2^2$, i.e., location 12 of the sn-tuple. The entire sn-tuple recognition process is depicted in Fig. 1.

Then, the response of the lth sn-tuple node is calculated by the following expression:

$$r_l = \sum_{j=1}^{u} \{ \sum_{q=1}^{f_{\max}} content(a_{q,j,l}) \}, \text{ where } f_{\max} = k - (n-1) \cdot f, \tag{6}$$

where $f_{\max}$ is the maximum offset value possible for the current chain-code $\underset{\sim}{c}$, so as to cover all chain-code elements. The internal summation of (6) indicates the calculation of the sum of the outputs of a single sn-tuple for all possible offset values (i.e., for the entire chain-coded pattern). The external summation of (6) represents the accumulation of the outputs of all sn-tuple functions to generate the node response.

2.2.2 Decision Margin and Classification Confidence of n-Tuple Systems

For a network consisting of two or more sn-tuple-based nodes, the decision margin r_{diff} is

$$r_{diff} = r_{\max\ 1} - r_{\max\ 2}, \tag{7}$$

where $r_{\max\ 1}$ denotes the maximum node response and $r_{\max\ 2}$ denotes the second-highest response in the network. The decision margin provides a measure of the certainty with which the network classifies the input pattern to a given class and is used frequently in standard n-tuple networks. If the correct classification result is known, then the classification confidence *cnf* is defined as:

$$cnf = \begin{cases} r_{corr} - r_{\max\,2}, & \text{if a correct classification is made;} \\ r_{corr} - r_{\max\,1}, & \text{if an incorrect classification is made,} \end{cases} \tag{8}$$

where r_{corr} is the response of the node corresponding to the actual class. According to (8), if a pattern is correctly classified, then the classification confidence is positive. On the contrary, if it is misclassified, the confidence becomes negative. Pattern recognition schemes based on the relative magnitudes of responses for n-tuple networks have been presented in [10]. Additionally, in [11], the decision margin is used as an indication of the network classification performance.

2.2.3 The sn-Tuple versus the Standard n-Tuple Method

When applying the sn-tuple network to classification operations, for all combinations occurring in the chain-coded pattern, the stored frequencies are added to generate the node response. Thus, though in the standard n-tuple method one node uses a large number of n-tuples to cover all pixels of an input pattern with fixed dimensions, in the scanning n-tuple method, for each pattern class, a node with very few sn-tuples suffices to cover a variable-length pattern.

The substitution of the standard n-tuple by the scanning n-tuple results in an increase in preprocessing time. However, the reduced computational load during recognition in comparison to standard n-tuple networks coupled with the prospect of superior results due to the use of the chain-coding justifies experimentation with the sn-tuple method. It is worth pointing out that the reliance of the sn-tuple classifiers on a small number of sn-tuples in comparison to standard n-tuples may require different techniques to optimize the classification performance. Similarly, the memory requirements of the two models differ. For an n-tuple node, the number of memory locations required is:

$$2^n \cdot u, \tag{9}$$

where n is the number of sampled pixels for each tuple and u is the number of n-tuples used per node. Achieving complete coverage of all M pixels in the input matrix requires:

$$u = \left\lceil \frac{M}{n} \right\rceil. \tag{10}$$

For an sn-tuple using σ-direction chain-coding, the required number of memory locations is:

$$2^{n \cdot \lceil \log_2(\sigma) \rceil} \cdot u. \tag{11}$$

Usually, for a given task, the *number of sn-tuples* is smaller than the *number of n-tuples*. However, as shall be shown in the next section, the use of σ-valued rather than binary data in the case of sn-tuples means that the two approaches have similar memory requirements.

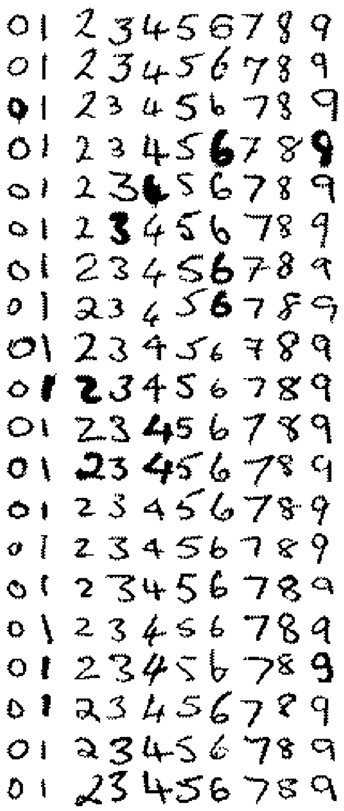


Fig. 2. The first 20 training patterns from each digit class of the ESSEX handwritten character data set.

3 The Character Classification Task

The performance of an sn-tuple network in handwritten character recognition tasks using one node per class has been studied for the ESSEX and CEDAR data sets of handwritten digits, giving recognition rates of 91.4 percent and 97.3 percent, respectively, when "training on a predefined set of training data, then reporting the classification accuracy on a disjoint predefined set of test data" [1]. This approach shall be adopted in the experiments reported here, for comparative purposes. In the ESSEX data set, the sn-tuple model using four sn-tuples is reported to have a performance superior to the standard n-tuple model, which generates a recognition rate of 90.6 percent for nodes of 262 standard n-tuples each sampling eight points.

We focus on the ESSEX data set, in an effort to improve the classification performance since this data set has been shown to be more difficult to classify accurately than the CEDAR data set. The ESSEX data set consists of 3,917 training and 1,835 test patterns. Each character is represented as a 42 x 50 matrix of binary points. The first 20 training patterns from each digit class are shown in Fig. 2. To reduce the character variation, the data set is preprocessed by centering and scaling each character so that it fully occupies the input matrix and applying a median-filter operator to suppress the existing noise. The normalized pattern is then chain-coded as reported in [1]. The character area is scanned from the upmost-left corner and the transitions along the character edges are recorded for each point as one of eight possible directions, at intervals of 45 degrees. The resulting chain-code consists of a string of elements, each

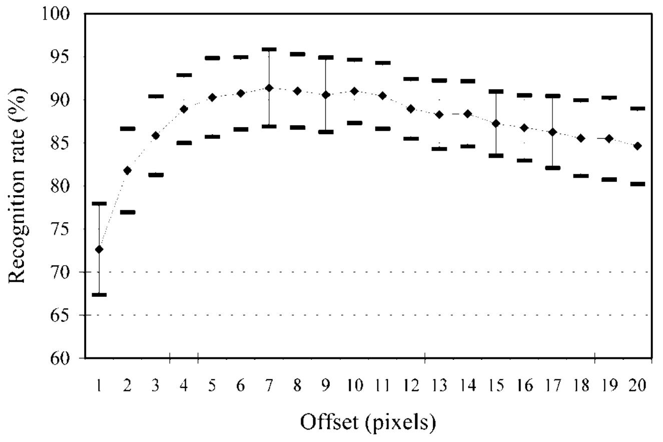


Fig. 3. Average test set accuracy over the 10 digit classes and 95 percent confidence interval using a single sn-tuple system sampling five points, for offsets ranging from 1 to 20 points. Calculations are based on one test run per offset, measured over each of the 10 classes.

with one of eight possible values. If the character consists of two or more chain-codes (for example, a noise-free "0" consists of one chain-code along the outer perimeter and one chain-code along the inner perimeter) these are simply concatenated. Though this approach may be improved upon, it has been selected here for comparative purposes since it was used in [1].

In the experiments of Lucas and Amiri [1], the sn-tuple method had a substantially higher classification performance than the standard n-tuple method. Each standard n-tuple samples eight binary-valued elements. On the other hand, each sn-tuple samples five elements, each of which may have one of eight possible values and, thus, needs three binary digits to be encoded. By using (9) and (11), it can be seen that each sn-tuple node requires a total of 131,072 memory locations, while each standard n-tuple node requires only 67,072 memory locations. To counter the increased sn-tuple memory requirements, either sparse-coding techniques may be used to remove zero-content memory locations or networks with fewer functions may be investigated. The latter solution is the focus of the next section.

4 Experimental Results

4.1 Optimizing the System Parameters

Because the classification rate of the training set for all network configurations is approximately 100 percent, we focus on the classification rate over the testing set. Additionally, the variance of the recognition rate over the digit classes is reported together with the memory requirements and the simulation time. It should be stressed that though a high-performing workstation is used in the experiments reported here, the processing times obtained are normalized by replicating the experiments of [1] on the same hardware, thus allowing for a more direct comparison.

Initially, the configuration proposed in [1] is simulated (one network node per class, each with four sn-tuples with offsets of two, three, four, and five points), giving a recognition rate of 89.3 percent. The deviation from the 91.4 percent rate quoted in [1] is probably attributable to different preprocessing steps. To investigate this, the performance of networks whose nodes consist of only a single sn-tuple is presented in Fig. 3 for different values of the offset. The recognition performance for a network with a single-sn-tuple node is found to rise for offsets higher than these used in [1] and for offsets between 6 and 11 gives better results than the aforementioned four sn-tuple system, while both the memory requirements and processing time are considerably reduced. As summarized in Table 1a, the optimal recognition rate is equal to 91.4 percent for single sn-tuple nodes and 91.6 percent for nodes consisting of four sn-tuples. Notably, the single sn-tuple system gives a recognition rate equivalent to the optimal value of [1] where four sn-tuples were used per class, while having a memory requirement equal to approximately one fourth of the four-tuple system. At the same time, due to the smaller number of sn-tuples per node, the time required to perform the character classification task is reduced to only 38 percent of that quoted by Lucas and Amiri [1], when simulated on the same hardware.

Another potential improvement involves the application of the sn-tuple technique. As described in [1], the sn-tuple is activated up to the point where the final element of the mask (element m_n) coincides with the end of the chain-code (see also (4)). For a given sn-tuple offset f, this introduces an edge effect, where the last q chain-code points are not fully scanned (for example, none of these points is used as the first mask element), with q being:

$$\begin{aligned} q &= f \cdot (n-1) \\ &= k - f_{\max},\ \textit{according to the definition of } f_{\max}. \end{aligned} \tag{12}$$

In the ESSEX data set, the average chain-code length is equal to 200 points and, thus, for an sn-tuple size of 5 and an offset equal to 5, 20 points are not fully scanned (10.0 percent of the pattern). If the offset is 10, then 45 points are not scanned (22.3 percent of the pattern). These points are located at the end of the chain-code, resulting in a potential loss of information, which becomes more marked for digits with shorter chain-codes. However, chain-coding is a continuous transformation whose result depends on the selection of the starting pixel as far as the initial point is concerned, though the sequence of chain-code points remains the same. For example, if the origin of a chain-code $\{c_1, c_2, c_3, c_4\}$ is shifted by two pixels, the resulting chain-code is $\{c_3, c_4, c_1, c_2\}$. Therefore, it is proposed to resample the initial chain-code points when scanning near the end of the chain-code to allow an equivalent application of the sn-tuple mask to each chain-code point, the first and last sn-tuple applications becoming $\underline{t_{1,j}}$ and $\overline{t_{k,j}}$ as compared to (4). *Resampling* removes the edge effect due to the arbitrary choice of initial point.

4.2 Investigating the ESSEX Data Set Properties

The detailed classification of test patterns for the highest-performing four-sn-tuple network reveals a large variance of the recognition rate over the digit classes (see Table 2). In particular, digit class "1" has the lowest recognition

TABLE 1
Classification Results

Method & network configuration	Recognition rate	Recognition variance	Memory requirements	Simulation time
4 x sn-tuples *(f= 2, 3, 4, & 5)* [1]	91.4 %	--	100.0 %	100.0 %
4 x sn-tuples *(f = 2, 3, 4 & 5)*	89.3 %	6.90	100.0 %	100.0 %
4 x sn-tuples *(f = 6, 7, 8 & 10)*	91.6 %	6.04	100.0 %	85.3 %
1 x sn-tuple *(f = 7)*	91.4 %	6.27	25.0 %	27.2 %

(a)

Method & network configuration	Recognition rate	Recognition variance	Memory requirements	Simulation time
4 x sn-tuples with sub-classes *(f = 6, 7, 8 & 10)*	92.5 %	3.63	110.0 %	101.1 %
4 x sn-tuples with resampling and sub-classes *(f = 6, 7, 8 & 10)*	92.6 %	4.19	110.0 %	125.2%
1 x sn-tuple with resampling and sub-classes *(f = 7)*	92.2 %	4.50	27.5 %	37.7%

(b)

(a) Classification results for different network configurations and values f *of the offset as compared to the results of [1], which are quoted in the first row of the table. (b) Classification results when introducing subclasses (in both cases, the memory requirement and simulation time entries are normalized over the values employed in [1]). All the results are obtained using the ESSEX test set, recognition rates being averaged over the 10 digit classes.*

accuracy (only 79.8 percent), which is much lower than the recognition rate for the entire data set (91.6 percent). An analysis of the ESSEX data set indicates that this is due to the existence of two fundamentally different subclasses for digit "1," without and with a horizontal line at the bottom of the character (denoted as subclasses "1A" and "1B," respectively). Within the training set for class "1," only 6 percent of the patterns belong to subclass "1B." Consequently, the node corresponding to class "1" is unable to extract the characteristics of "1B," as subclass "1A"

TABLE 2
Scatter Matrix for 10-Node Network, Indicating the Much Lower Recognition Accuracy Achieved for Class "1," in Comparison to Other Classes

		Digit pattern classification (%)									
		"0"	*"1"*	*"2"*	*"3"*	*"4"*	*"5"*	*"6"*	*"7"*	*"8"*	*"9"*
Class	*"0"*	92.9	0.0	0.0	0.5	0.7	0.0	5.3	0.0	0.9	0.0
	"1"	0.0	78.1	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0
	"2"	0.8	16.0	97.0	0.5	0.0	1.3	0.7	1.7	0.9	0.0
	"3"	0.0	2.1	1.3	94.1	0.0	5.8	0.0	0.8	3.4	0.0
	"4"	0.0	0.0	0.0	0.0	98.7	0.0	0.0	0.0	0.0	2.9
	"5"	0.0	0.2	0.0	2.3	0.0	88.4	2.0	0.0	5.9	0.0
	"6"	5.5	0.0	0.4	0.0	0.7	0.7	89.4	0.0	0.9	0.0
	"7"	0.0	2.8	0.0	0.5	0.0	0.0	0.0	92.4	0.0	0.0
	"8"	0.0	0.0	1.3	1.8	0.0	3.2	1.3	1.7	88.1	0.0
	"9"	0.8	0.7	0.0	0.5	0.0	0.7	0.7	3.4	0.0	97.1

dominates class "1." Instead, most test patterns belonging to subclass "1B" are classified as members of digit class "2," due to the horizontal line at the bottom of digits from both class "1B" and class "2."

To improve the performance of a deformable model-based handwritten character recognizer, Cheung et al. [12] propose the introduction of multiple subclasses for certain digit classes. Furthermore, Morns and Dlay [13] propose an automated algorithm to introduce new classes as required in perceptron-based networks. A similar approach is adopted here, establishing two subclasses for class "1," for which the scope for improvement is maximized. The partition of class "1" into two subclasses is achieved by a *geometry-inspired rule* examining whether the maximum width of the character at any row in the lower half of the pattern is equal to or exceeds 50 percent of the input matrix width. During classification of the test set, digits are classified to any of the 11 classes/subclasses and afterwards subclasses "1A" and "1B" are merged.

The experimental results obtained with both resampling and class splitting are shown in Table 1b. When each node consists of four sn-tuples of five points, with offsets of six, seven, eight, and 10, the recognition rate is equal to 92.6 percent, improving the results of [1] by 1.2 percent with only small increases in the memory requirements and processing time. The corresponding reduction in the error rate is considerable, being approximately 14 percent, compared to the best results previously reported. Even the single sn-tuple system gives an optimal recognition rate of 92.2 percent for an offset of seven, improving the performance of [1] by 0.8 percent, while the memory space and time required for classification are reduced by 73 percent and 62 percent, respectively. Thus, the recognition performance is substantially improved by introducing subclasses, the recognition of class "1" reaching 91 percent. On the other hand, resampling gives an improvement of only 0.1 percent to 0.2 percent.

These experiments indicate that the sn-tuple recognition accuracy may be further improved by introducing additional subclasses. It would be desirable to introduce such subclasses in an automated manner rather than by inspection, using an algorithm based exclusively on the training set (that is, without resorting to the testing set). To this end, four approaches, originally introduced in [14], are investigated. The first two involve studying the characteristics of each pattern class in an effort to determine cases where frequent misclassifications might be expected. The remaining two utilize the decision margin of the network for each pattern as an indication of its state. Thus, the decision margin is monitored following the completion of the training phase to determine the relative position of classes in the pattern space and, thus, better discriminate between closely-positioned pattern classes.

5 Improving the sn-Tuple Classification Accuracy

5.1 Method 1—Using ART-Type Top-Down Information Techniques

The first method employs a self-organizing scheme to separate each class into subclasses. This scheme is based on the provision of several discriminator nodes for every pattern class, for which each node represents one subclass. As a new pattern is presented to the network, it is compared to the knowledge already accumulated in each discriminator node (which forms a top-down expectation of the pattern). This approach shares principles of ART-type networks [15] in that it generates a stable partition of patterns into clusters based on the match between previous patterns and new patterns. A conceptually similar mechanism termed the distribution constraint [9] has been shown to allow a self-organizing network consisting of standard n-tuple nodes to cluster pattern classes in an autonomous manner. In contrast, when using a network with only a few sn-tuples per node, such a mechanism does not provide an improvement.

5.2 Method 2—Studying the Frequencies-of-Occurrence of Chain-Code Orientations

As in the previous method, each class is examined in isolation, in an effort to determine pattern clusters that differ considerably from the majority. Following the chain-coding operation, the pattern space consisting of the frequencies-of-occurrence of each direction in the chain-code is examined. Initially, the class average is determined for all directions. As a measure of the pattern difference, the distance from the class average in terms of the sn-tuple's contents is used. The direction of maximal variability is determined and then the pattern class is split along this direction. This results in an improved clustering performance, the best classification rate being equal to 91.98 percent when focusing in class "1" and trying to split subclasses "1A" and "1B." However, the method proves to be sensitive to the exact displacement from the class centroid where the boundary between the current class and the new subclass is set.

5.3 Method 3—Reinforcement Learning Techniques

This method applies a type of reinforcement learning to improve the network response. Reinforcement learning is required for training patterns which a) are misclassified (i.e., classified to class j though they belong to class i) or b) are correctly classified, but possess a very low decision margin. Though cases a) and b) differ, training for both types of effect can be expected to improve the network response. Evidently, the training for patterns of type a needs to be more intensive than for patterns of type b. Experiments have indicated that this method improves the network performance, giving a classification rate of up to 91.89 percent. However, the results depend strongly on determining the optimal amount by which the network is trained for each pattern of type a and type b. Also, this method is susceptible to overtraining since, by adapting the network excessively to "problematic" patterns, its response to other patterns is adversely affected and the collective recognition rate suffers accordingly.

5.4 Method 4—Class-Splitting Based on the Classification Confidence

The fourth method is also based on the confidence with which the network classifies the training patterns and is the method that has been selected for further experimentation. Initially, all the training patterns are classified by the system, which consists of P nodes, recording the confidence

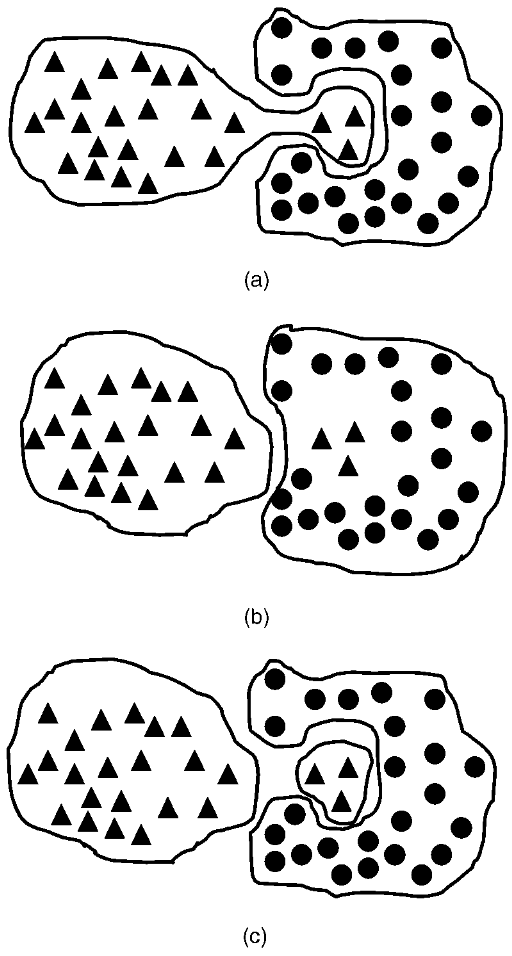


Fig. 4. Pattern space with two overlapping classes (a), together with the clustering result before (b) and after (c) applying method 4.

of each classification. Out of these, the N patterns with the lowest confidence are examined (in our initial experiments, $N = 100$). These patterns correspond to the negative- and low-confidence classifications (where patterns are correctly classified, but with a low confidence). From this set, the method selects the pattern class C_k to be divided into two subclasses as the one with the highest fraction of negative-confidence and low-confidence decisions. The pattern class C_l, which causes these low-confidence classifications for elements of C_k, is also determined. Then, a new network node is introduced, by splitting class C_k into subclasses C_{k1} and C_{k2}. The nucleus of subclass C_{k2} consists of the patterns belonging to C_k, which have a low or negative classification confidence due to C_l (in the example of Section 4.2, C_l would be class "2" while C_{k2} would be the subclass of "1" with a lower horizontal line). Following the introduction of the new node, the training set is classified by the expanded network, which consists of $P+1$ nodes. This process is repeated until the transfer of patterns between C_{k1} and C_{k2} either ceases or is stabilized (so that the same group of patterns is exchanged between the two subclasses in subsequent iterations). This procedure is graphically described in Fig. 4.

The class-splitting algorithm can be described formally by the following steps:

0. While the maximum number of classes has not been created:
1. LOOP 1: Train the network according to the existing classes.
2. Classify all training patterns according to the trained network and determine their respective classification confidence.
3. Order all training patterns according to the classification confidence and create a list L containing only the N patterns with the smallest classification confidence.
4. Select the class X whose patterns have the highest frequency-of-occurrence within the list L due to a specific class Y.
5. Split class X into classes X_1 and X_2, where X_2 consists of all elements of X which are in L due to class Y and the following properties hold:
   $X_1 \cap X_2 = \{\}$ and $X_1 \cup X_2 = X$.
6. LOOP 2: Train the network according to the existing classes.
7. Classify all training patterns according to the trained network and determine their respective classification confidence.
8. Order all training patterns according to the classification confidence and create a revised list L' containing only the first N patterns.
9. Create a list L'_{X_1} containing all patterns of class X_1 which are included within list L' due to class X_2 and a list L'_{X_2} containing all patterns of class X_2 within list L' due to class X_1.
10. If the system has not settled (i.e., at least one of the sets L'_{X1} and L'_{X2} contains new elements), return to step 6, replacing classes/sets X_1 and X_2 with (respectively):
    $X_1^{New} = (X_1 - L'_{X1}) \cup L'_{X2}$ and
    $X_2^{New} = (X_2 - L'_{X2}) \cup L'_{X1}$,
    where the sign "-" denotes the set difference.
11. If the number of existing classes (subclasses) is not equal to the predefined number of classes, return to step 0, to generate a new subclass (END OF LOOP 2).
12. Terminate the procedure (END OF LOOP 1 and of the procedure).

According to this algorithm, each repetition of LOOP1 creates a new node (and a new subclass) until the desired maximum number of classes is formed. On the other hand, each repetition of LOOP2 redistributes the elements in the existing classes to reduce the discrepancies of the classification, following the introduction of a new node. Whenever a new node is created, several iterations of LOOP2 are required in order to reach a stable partition of class X into classes X_1 and X_2. Initially, class X_2 contains the elements that are misclassified or are very close to being misclassified, while class X_1 contains the remaining elements of X. Thereafter, all elements of the $N+1$ classes are reclassified to determine whether splitting X can be fine-tuned toward the natural border between the two subclasses. The split of class X into the two classes is finalized when either 1) no

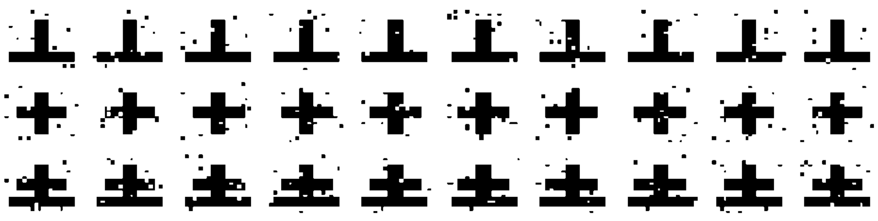


Fig. 5. Samples from the "inverted_T" (first row), "cross_A" (second row), and "cross_B" (third row) classes.

patterns are exchanged between the two constituent classes of X (indicating well-separated classes) or 2) a closed set of patterns are continuously exchanged between the classes (indicating less well-separated but distinct classes).

Here, the class-splitting operation is studied briefly using an sn-tuple network using four sn-tuple functions per node without resampling with offsets of 6, 7, 8, and 10. For the ESSEX data set, the class with most misclassifications is class "1," due to class "2." Therefore, class "1" is split into two new subclasses. In the first application of the aforementioned procedure, seven patterns form the nucleus of the new subclass, resulting in an improved recognition rate of 92.04 percent. The network converges after nine refinement iterations, when the new subclass consists of 29 patterns and the network recognition rate reaches 92.52 percent. This is very close to the recognition rate obtained using the rule to manually split class "1" (92.55 percent), as described in Section 4.2. Furthermore, the classification result is stable, even when the system is allowed to perform additional refinement steps. Since the class-splitting method has the highest performance among the four techniques, it is further studied in the following section.

6 Optimizing the Scanning n-tuple Performance

6.1 Solving a Small-Scale Problem via Class-Splitting

To illustrate the effectiveness of the class-splitting algorithm, we first examine its performance on the task of recognizing two classes, namely, crosses and inverted T-shaped objects. Both classes are generated using prototypes to which 5 percent random noise is injected. For each class, both the training set and the testing set consist of 20 patterns each. The cross class comprises two subclasses, one where the cross has a bottom edge (archetype "cross_B") and one where the bottom edge is missing (archetype "cross_A"). More specifically, the training set of the cross class consists of 18 patterns generated by "cross_A" and two patterns by "cross_B," while in the test set 12 patterns are generated by "cross_A" and eight patterns by "cross_B." On the contrary, the "inverted_T" class consists of a single archetype to which noise is added. As can be seen by Fig. 5, "cross_B" has a relatively high degree of similarity with the "inverted_T" class. Furthermore, in the training set, the "cross_A" patterns dominate the "cross" class. Consequently, the "cross_B" patterns are not recognized correctly using an sn-tuple network consisting of two nodes, as shown in Table 3a. Since the frequency of "cross_B" patterns is higher in the testing set, a low recognition rate is obtained (only 60 percent for the entire cross class). A study of the decision margins for all training patterns does indeed indicate that the two training patterns belonging to the "cross_B" subclass are misclassified by the trained network, these being the only misclassified patterns in the training set. By applying the proposed class-splitting algorithm and introducing one additional node, the network succeeds in separating the classes, even in the presence of noise. This is achieved via self-organization, without any explicit information about the specific classes.

6.2 Application of the Class-Splitting Algorithm to the ESSEX Data Set

The class-splitting algorithm has been applied to the ESSEX data set using an sn-tuple network with four sn-tuples per node employing offsets of 6, 7, 8, and 10 without resampling. In our experiments, five class-splitting iterations were run (allowing a 50 percent increase in the network size), resulting in a total of 15 classes. The aim of this experiment is to study whether the recognition performance would be improved using a limited number of classes, rather than to achieve a very high performance by creating a large number of smaller classes. Therefore, five class-division steps have been allowed. These are determined by the algorithm of Section 5.4 as:

1. the splitting of class "1" into classes "1A" and "1B" in order to improve the separation of classes "1" and "2";
2. the splitting of class "1A" into "1AA" and "1AC" to improve the separation of classes "1A" and "7";
3. the splitting of class "2" into "2A" and "2B" to better separate "2" from "3";
4. the splitting of class "6" into "6A" and "6B" to better separate "6" from "0";
5. the splitting of class "3" into "3A" and "3B" to better separate "3" from "5".

The system recognition rate through each step is plotted in Fig. 6 (recall that the class-splitting algorithm relies only on the training set, but we report the accuracy of the classifier on the test set). When a new class is selected for splitting, the collective recognition performance may deteriorate temporarily. However, following that reduction, the recognition rate is steadily improved in subsequent iterations. As can be seen, even though the decisions of the class-splitting algorithm are based on the training set, all class-splitting operations result in an improvement of the clustering performance over the testing set. The algorithm can be seen to settle successfully to new classes, the network performance reaching a plateau for a given population of classes after a number of iterations. A typical example is shown in Fig. 7, which displays the system performance

TABLE 3
Classification Results for the Cross and Inverted-T Classes

		actual class	
		cross	inverted_T
classification Result	Cross	60%	0%
	Inverted_T	40%	100%

(a)

		actual class		
		cross_A	cross_B	inverted_T
classification result	cross_A	100%	0%	0%
	cross_B	0%	100%	0%
	inverted_T	0%	0%	100%

(b)

(a) Before and (b) after performing the class-splitting operation.

when 14 nodes are being used. Certain classes (such as class "1") are split up more than once, as required by the characteristics of the training set. Another interesting point involves the evolution of the class-splitting algorithm, as typified by Fig. 8. In this case, the system evolves so that when the algorithm settles with 13 classes, the new class C_{k2} contains the majority of the patterns from the original class C_k while the old class C_{k1} contains fewer patterns. Notably, the algorithm converges to a solution even though the initial partition following the class-splitting step is not the most favorable.

6.3 Application of the Class-Splitting Algorithm to the NIST Data Set

Our last experiment was performed using data from the NIST database #19 of handwritten characters. More specifically, the first data set (hsf_0) was selected for processing.

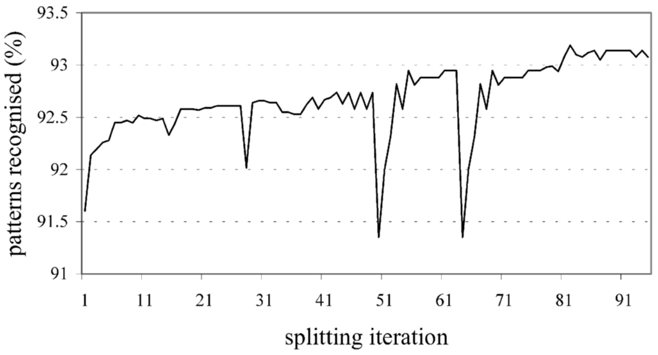


Fig. 6. The recognition rate of the system as it evolves from a 10-node system into a 15-node system. New nodes are introduced at iterations 1 (to separate class "1" from "2"), 15 ("1A" from "7"), 28 ("2" from "3"), 50 ("6" from "0"), and 64 ("3" from "5").

Similar to earlier experiments, the 10 digit classes were chosen and, initially, the characters in each class were shuffled randomly using the NIST-specified "shuflmis" utility. Then, the first 600 characters from each class formed the training set, the following 200 characters from each class forming the test set. The patterns of the two sets were chain-coded and presented to a system consisting of 10 sn-tuple nodes, one being assigned to each digit class. The system classification accuracy for the test set was equal to 90.05 percent. Though this is relatively low, it should be stressed that emphasis here is placed on evaluating if this performance can be improved by the class-splitting algorithm rather than fine-tuning the recognition performance of the original system. Thus, parameters such as the number of sn-tuples used per node and the offset were not modified as compared to the values used with the ESSEX data set.

The class-splitting algorithm was then activated, adding initially one node to split class "0" so as to reduce the confusion to class "8." This choice was once again based on the maximum confusion between any two classes within the 100 patterns with the lowest classification confidence margin. The 11-node network settled after a total of 25 iterations, resulting in a classification rate of 90.45 percent. Subsequently, additional splitting steps were performed, of class "2" (to reduce the confusion with "3") resulting in a classification rate of 90.55 percent and of class "7" (to reduce the confusion with "9") resulting in a classification rate of 90.65 percent. Thus, the class-splitting algorithm succeeded in improving the NIST classification accuracy. It should be noted that alternative test sets were also formed using the patterns of the NIST data set. All of them showed similar gains when using the class-splitting operation with the aforementioned training set.

In concluding the experimental results section, it is worth briefly studying the storage capacity implications of

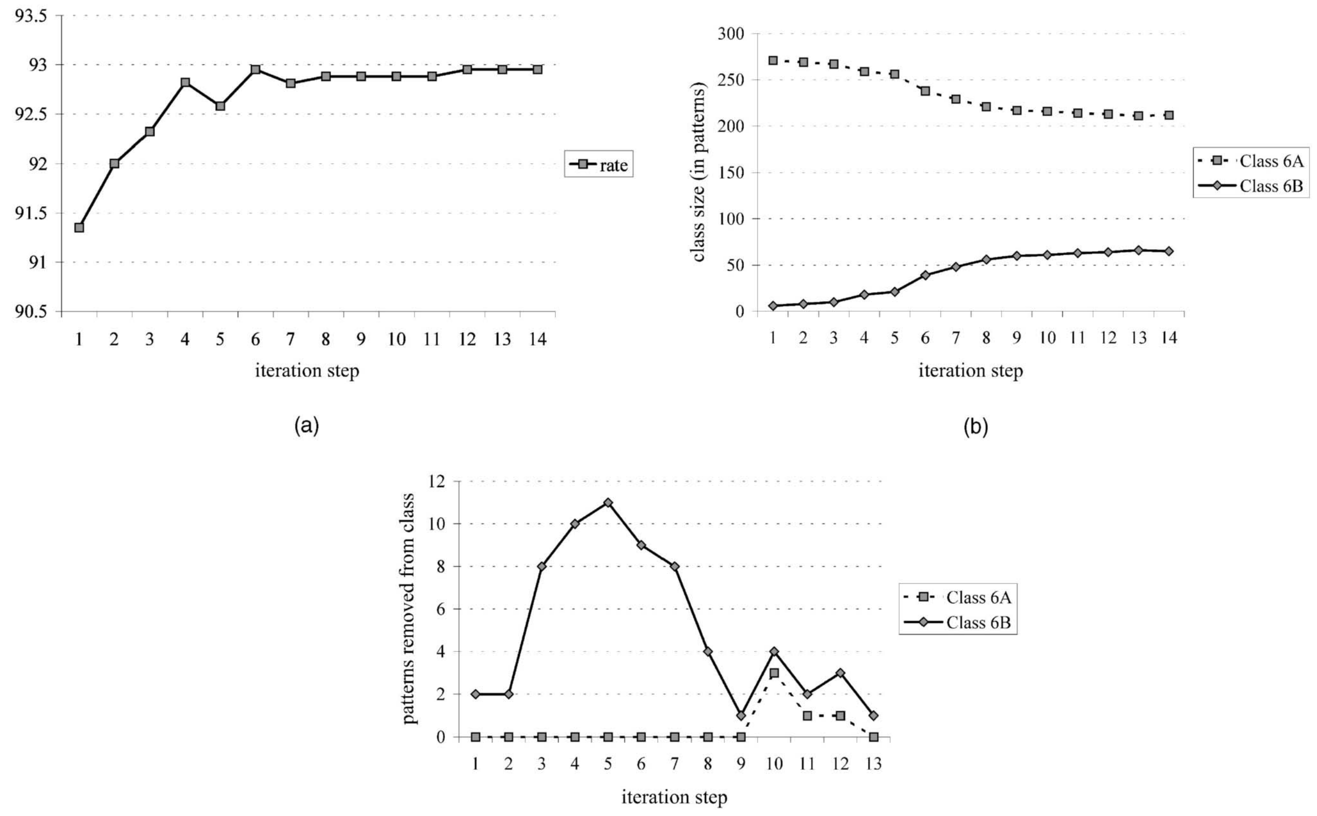


Fig. 7. (a) Classification accuracy of the test set of a 14-class system, (b) number of patterns in classes 6A and 6B, and (c) number of patterns exchanged between the two classes at each step.

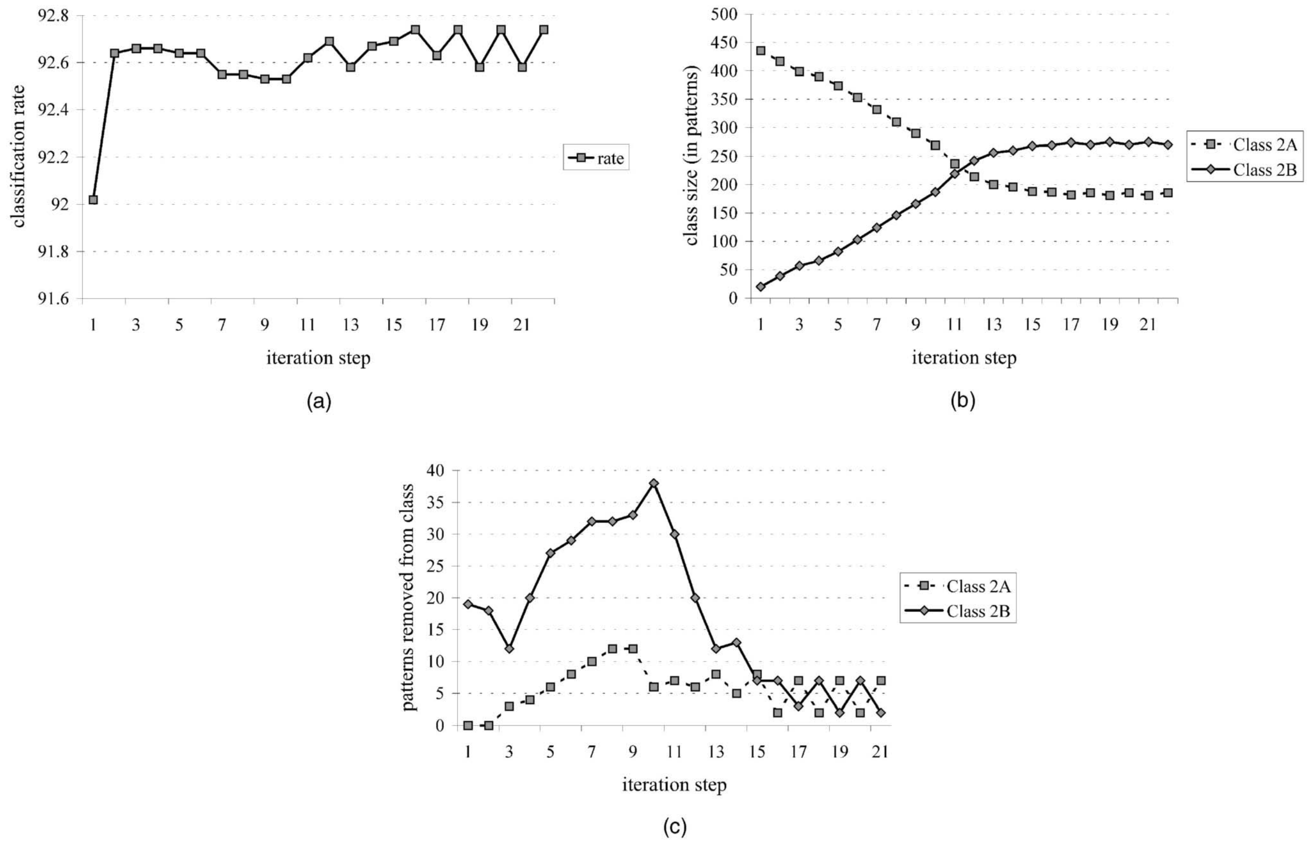


Fig. 8. (a) Classification accuracy of the test set of a 13-class system, (b) number of patterns in classes 2A and 2B, and (c) number of patterns exchanged between the two classes at each step.

employing the class-splitting algorithm. By introducing additional nodes, the sn-tuple performance is improved, though the storage requirement is unavoidably increased. In our experiments, an upper limit of 50 percent was placed on the additional storage capacity. However, it is possible that the introduction of additional nodes via the class-splitting algorithm may allow the use of a network with fewer sn-tuples per node, to generate results of the same accuracy or an even higher one while allowing a reduction in the total memory requirements. This issue is the subject of ongoing research.

7 Conclusions

In this paper, the application of the sn-tuple to character recognition tasks has been studied. More particularly, the focus has been to evaluate a number of methods to improve the scanning n-tuple performance. Therefore, initially, issues such as the optimal sn-tuple offset and the optimal number of sn-tuples have been studied with respect to the classification accuracy, the processing time, and the memory requirements. Following this study, possible shortcomings of the sn-tuple have been examined, using a handwritten character recognition task as a case study. A number of automated algorithms have been evaluated as possible vehicles for solving these shortcomings. It has been found that, though the sn-tuple and standard n-tuple techniques are closely related, different mechanisms are needed to improve the sn-tuple performance. The most promising algorithm has been presented in detail and applied to three distinct recognition tasks. This algorithm has been shown to markedly improve the sn-tuple classification performance. Since this is a general-purpose self-supervised algorithm, it considerably increases the sn-tuple's effectiveness in real-world problems. The experimental results illustrate that this algorithm gives a recognition performance equivalent to that obtained using structural knowledge regarding the different classes, thus indicating its usefulness when applied to large-scale recognition tasks with the scanning n-tuple technique.

In this article, the class-splitting algorithm is used in conjunction with the sn-tuple technique. However, class-splitting should also be applicable to the standard n-tuple technique, improving the classification performance of n-tuple networks via the introduction of additional subclasses. Due to space restrictions, this application has not been studied in detail here.

Acknowledgments

The author wishes to thank Dr. A. Amiri for his assistance in replicating the chain-coding technique used in the original ESSEX data set experiments as well as for stimulating discussions. Furthermore, the author wishes to acknowledge the role of the anonymous referees, who have contributed—via their constructive comments—to the improvement of this manuscript.

References

[1] S. Lucas and A. Amiri, "Statistical Syntactic Methods for High-Performance OCR," *IEE Proc. Vision, Image, and Signal Processing,* vol. 143, no. 1, pp. 23-31, 1996.

[2] R. Plamondon and S.N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Study," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, pp. 68-85, Jan. 2000.

[3] I. Aleksander, "Emergent Intelligent Properties of Progressively Structured Pattern Recognition Nets," *Pattern Recognition Letters,* vol. 1, pp. 375-384, 1983.

[4] R. Rohwer and M. Morciniec, "The Theoretical and Experimental Status of the n-Tuple Classifier," *Neural Networks,* vol. 11, no. 1, pp. 1-14, 1998.

[5] T.M. Jorgensen and C. Linneberg, "Theoretical Analysis and Improved Decision Criteria for the n-Tuple Classifier," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 4, pp. 336-347, Apr. 1999.

[6] D.-M. Jung, M.S. Krishnamoorthy, G. Nagy, and A. Shapira, "N-Tuple Features for OCR Revisited," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 7, pp. 734-745, July 1996.

[7] T.M. Jorgensen, "Classification of Hand-Written Digits Using a RAM Neural Net Architecture," *Int'l J. Neural Systems,* vol. 8, no. 1, pp. 17-25, 1997.

[8] *RAM-Based Neural Networks,* J. Austin, ed. Singapore: World Scientific Publishers, 1998.

[9] G. Tambouratzis and D. Tambouratzis, "Self-Organisation in Complex Pattern Spaces Using a Logic Neural Network," *Network: Computation in Neural Systems,* vol. 5, pp. 599-617, 1994.

[10] I. Aleksander and T.J. Stonham, "Guide to Pattern Recognition Using Random-Access Memories," *Computers and Digital Techniques,* vol. 2, no. 1, pp. 29-40, 1979.

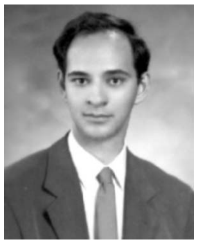
[11] M. Morciniec and R. Rohwer, "Benchmarking n-Tuple Classifier with Statlog Datasets," *RAM-Based Neural Networks,* J. Austin, ed., World Scientific Publishers, pp. 53-60, 1998.

[12] K.-W. Cheung, D.-Y. Yeung, and R.T. Chin, "A Bayesian Framework for Deformable Pattern Recognition with Application to Hand-Written Character Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 12, pp. 1382-1388, Dec. 1998.

[13] I.P. Morns and S.S. Dlay, "The DSFPN, a New Neural Network for Optical Character Recognition," *IEEE Trans. Neural Networks,* vol. 10, no. 6, pp. 1465-1473, 1999.

[14] G. Tambouratzis, "Improving the Classification Accuracy of the Scanning n-Tuple Method," *Proc. 15th Int'l Conf. Pattern Recognition (ICPR-2000),* vol. 2, pp. 1050-1053, 2000.

[15] G.A. Carpenter and S. Grossberg, "ART 2: Self-Organisation of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics,* vol. 26, pp. 4919-4930, 1987.



George Tambouratzis obtained the Diploma in electrical engineering (specialization in electronics) from the Electrical Engineering Department of the National Technical University of Athens (N.T.U.A.), Greece, in July 1989. From October 1989 to September 1993, he pursued his postgraduate studies at the Department of Electrical Engineering of Brunel University, United Kingdom, which led to the award of an MSc degree in digital systems (September 1990) and a PhD degree in neural networks and pattern recognition (September 1993). After completing his military service (1993-1995), he lectured at the Hellenic Naval Academy (1995-1996). Since 1996, he has been associated with the Institute for Language and Speech Processing, Athens, Greece, where he currently holds a research post. He is a member of the Technical Chamber of Greece, the IEEE, and IEEE Systems, Man, and Cybernetics Society. His research interests cover the area of neural networks, with emphasis in self-organizing logic neural networks and their application to real-world tasks. Other areas of research activity include pattern recognition, image processing, speech synthesis, and computational linguistics.