Available online at www.sciencedirect.com

Pattern Recognition 40 (2007) 898–919

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/pr

A genetic framework using contextual knowledge for segmentation and recognition of handwritten numeral strings

Javad Sadri[a,*], Ching Y. Suen[a], Tien D. Bui[b]

[a]CENPARMI (Center for Pattern Recognition and Machine Intelligence), Computer Science Department, Concordia University, 1455 de Maisonneuve Blvd. West, Montreal, Que., Canada H3G1M8

[b]Computer Science and Software Engineering Department, Concordia University, 1455 de Maisonneuve Blvd. West, Montreal, Que., Canada H3G1M8

Received 8 September 2005; received in revised form 16 May 2006; accepted 2 August 2006

Abstract

For the first time, a genetic framework using contextual knowledge is proposed for segmentation and recognition of unconstrained handwritten numeral strings. New algorithms have been developed to locate feature points on the string image, and to generate possible segmentation hypotheses. A genetic representation scheme is utilized to show the space of all segmentation hypotheses (chromosomes). For the evaluation of segmentation hypotheses, a novel evaluation scheme is introduced, in order to improve the outlier resistance of the system. Our genetic algorithm tries to search and evolve the population of segmentation hypotheses, and to find the one with the highest segmentation/recognition confidence. The NIST NSTRING SD19 and CENPARMI databases were used to evaluate the performance of our proposed method. Our experiments showed that proper use of contextual knowledge in segmentation, evaluation and search greatly improves the overall performance of the system. On average, our system was able to obtain correct recognition rates of 95.28% and 96.42% on handwritten numeral strings using neural network and support vector classifiers, respectively. These results compare favorably with the ones reported in the literature.

Keywords: Handwritten digit recognition; Handwritten numeral string segmentation; Handwritten numeral string recognition; Genetic algorithm; Contextual knowledge

1. Introduction

Segmentation and recognition of unconstrained handwritten numeral strings is one of the most challenging problems in the area of optical character recognition (OCR). It has been a popular topic of research in OCR for years, and it also has many potential applications such as bank check processing, automatic mail sorting, and tax form reading [1]. One of the main challenges in this problem is the recognition and discrimination of digits from non-digit patterns (outliers). Outliers are mainly produced during the segmentation process, and they are the result of over or under segmentation of the string images. Since segmentation of numeral strings produces both valid digits and outliers, many researchers have utilized isolated digit classifiers/verifiers in order to distinguish between them [2–4]. Researchers in Ref. [4] trained different isolated digit classifiers (such as linear discriminant functions (LDF), quadratic discriminant functions (QDF), neural networks (NN), and support vector machines (SVM)) by using outlier samples in order to improve their outlier resistance/rejection. These researchers showed that improving the outlier resistance of classification module in a handwritten numeral string recognition system can improve the overall performance of the system. However, using the isolated digit classifiers for evaluation of all the outlier patterns entails a very high computational cost. More importantly, there are many cases of outlier patterns which are not correctly rejected or recognized by

* Corresponding author. Tel.: +1 514 848 2424x7950; fax: +1 514 848 2830.

E-mail addresses: j_sadri@cs.concordia.ca (J. Sadri), suen@cenparmi.concordia.ca (Ching Y. Suen), bui@cs.concordia.ca (Tien D. Bui).

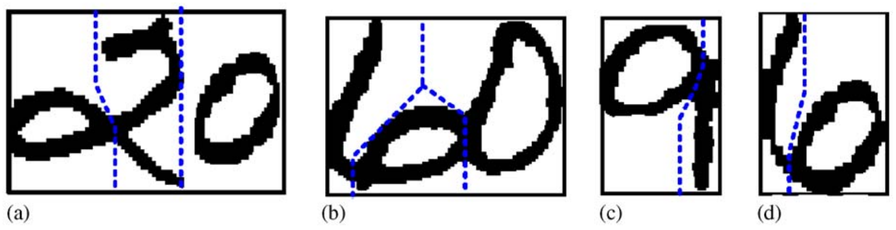


Fig. 1. With relying only on the recognition scores of the isolated digit classifiers, and without using contextual information, it is more likely that most of the numeral strings shown in this figure will be misrecognized with very high confidence values. For example, in (a), 20 will be misrecognized as 020, (b) 60 will be misrecognized as 100, (c) 9 will be misrecognized as 01, and in (d) 6 will be misrecognized as 10.

isolated digit classifiers (see examples in Fig. 1). In fact, such examples of segmented numeral strings can be easily misrecognized with a very high level of confidence scores produced by the isolated digit classifiers, regardless of whether those classifiers have been trained or not by the outlier samples. These kinds of misrecognitions can yield to very hazardous or costly consequences in applications, such as bank check processing.

Another important challenge in the segmentation and recognition of handwritten numeral strings, especially when the strings are long, is finding their optimum segmentation/recognition. Normally, during the segmentation process of long numeral strings, many redundant cutting paths are produced, so a search is conducted to find the optimum combination, among all the possible combinations of the cutting paths (so-called segmentation hypotheses). Using an exhaustive search to find the optimum segmentation hypothesis, while the number of cutting paths is large, is quite unfeasible. In order to find the optimum segmentation hypotheses in handwritten numeral strings, many researchers have used recognition scores for evaluation, and dynamic programming (DP) for searching [2–4]. However, Liu et al. [4] found that under some constraints of the evaluation scores or objective functions (e.g. non monotonicity), DP search does not guarantee finding the global optimum. To relax the constraints on the evaluation scores and objective functions, in Ref. [5], we proposed using a genetic algorithm (GA), as an alternative searching strategy for finding the optimum segmentation hypotheses in long numeral stings. However, experiments showed that our GA also must be improved in order not to get stuck by the outlier patterns which are frequently found in segmentation hypotheses.

Although many methods have been proposed for segmentation and recognition of handwritten numeral strings (such as [2,4,6,7], etc.), it seems that present techniques are not yet adequate enough for modeling the infinite variations of the handwritten strings, and much more improvements are still required in order to build robust systems for practical applications. In [5], we introduced the baseline of a handwritten numeral string recognition system, and we presented some preliminary results. Since then, we have modified the segmentation algorithm in order to produce fewer outliers, and we have also utilized contextual knowledge in our GA in order to improve the evaluation and searching of the segmentation hypotheses (chromosomes). In this paper, we present the improved version of our system which utilizes contextual knowledge efficiently in segmentation, evaluation, and searching of the segmentation hypotheses. We show that the results of these improvements yields to a higher overall performance in handwritten numeral string recognition systems, though we do not use very high computational cost digit classifiers. We also show that contextual information provides better and fewer candidate cutting paths, and it remedies the insufficient outlier resistance of the isolated digit classifiers.

The rest of this paper is organized as follows: Section 2 presents some background information, and an overall review of the previous methods for segmentation/recognition of numeral strings. Section 3 presents a mathematical model for the problem. Section 4 describes the detailed outline of our system. Section 5 presents the details and discussion of our experimental results, and finally, Section 6 offers our conclusions.

2. Review of segmentation and recognition methods for numeral strings

A lot of methods have been proposed for segmentation and recognition of handwritten numeral strings. In this section, we briefly review some of those works. Generally, character or numeral string segmentation is defined as an operation that tries to decompose an image of a sequence of characters or digits into sub-images of individual symbols [8]. For decomposing a string image into its constituent patterns, some cutting paths are introduced, and for constructing those cutting paths, usually two types of features are extracted. The first type of features are foreground features, which are extracted from the black pixels of the image. There are several techniques for extracting these features, such as contour tracing [6,9], and stroke analysis [10]. The second type of features are background features, which are extracted from the white pixels of the image. There are also several techniques for extracting background features, such as analyzing background regions [11], and analyzing background skeletons [12]. Recently, some techniques have used a

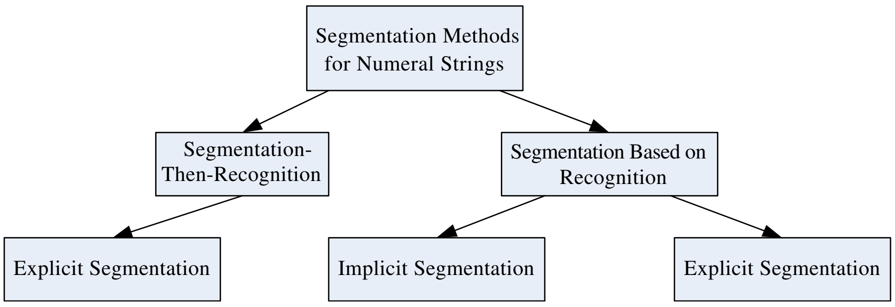

Segmentation Methods for Numeral Strings
Segmentation-Then-Recognition
Segmentation Based on Recognition
Explicit Segmentation
Implicit Segmentation
Explicit Segmentation

Fig. 2. Classification of current methods for segmentation and recognition of numeral strings.

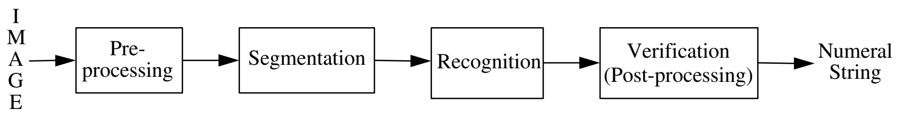


Fig. 3. A general block diagram of handwritten numeral string recognition system.

combination of both foreground and background features, such as [13–15]. Usually, approaches based on extraction of a combination of both foreground and background features have better segmentation results.

For recognition of handwritten numeral strings, there also have been two general approaches: segmentation-then-recognition, and segmentation-based-on-recognition [2]. In the first approach, the segmentation module provides a single sequence of partial images based on some heuristic rules, where each subsequence should contain an isolated character which is submitted to a recognizer [1,10,16]. Since this technique does not use any contextual or recognition feedback (for segmentation), it shows its limits rapidly when the correct segmentation does not fit the predefined rules of the segmenter. In the second strategy, first the segmenter provides a list of segmentation hypotheses, and then the recognition module evaluates each hypotheses [2,17,18]. This approach gives better results and a higher reliability than the first one. However, it entails a higher computational cost, since it has to generate all the segmentation hypotheses and then compare their recognition results. In addition, in this latter method, the correct segmentation rate depends too much on the accuracy of the isolated digit classifier in recognition or rejection of digits or outliers.

In summary, we can classify segmentation and recognition methods of numeral strings according to Fig. 2. As seen in this diagram, segmentation can be either explicit or implicit. In the explicit methods, segmentation is accomplished explicitly prior to recognition in order to provide primitive segments (candidate digits) for the recognizer, such as [1,15,19]. In these methods, cutting paths are normally found from the contour, profile, background or foreground regions or skeletons of the string image. However, in implicit methods, segmentation is embedded in the recognition process and is performed simultaneously with recognition such as in Refs. [14,20]. In these methods, the challenge consists of finding the best compromise between segmentation and recognition. As indicated in the literature, explicit segmentation methods tend to have a better performance than implicit methods [2,13]. A conventional handwritten numeral string recognition system has an architecture as shown in Fig. 3. Here, the function of each module is briefly explained. Preprocessing: accomplishes some image processing tasks such as binarization, smoothing or normalization. Segmentation: tries to separate connected or touching digits by introducing some cutting paths. Recognition: assigns class labels and confidence scores to each segment (candidate digit). Verification: an optional module which verifies or post-processes the decisions made by the recognizer. For more information on character string segmentation algorithms, see Ref. [8]. In the next section, we propose a mathematical model for segmentation and recognition of string images.

3. Problem formulation

Segmentation and recognition of string images can be modeled as an optimization problem as follows: there is an input image I as shown in Fig. 4, and it is assumed that I contains a handwritten numeral string with an unknown length. We are looking for a segmentation or partitioning of image I denoted by $S^i (i = 1, \dots, N)$, such that: S^i has m_i partitions which are denoted by s^i_j, $j = 1, \dots, m_i$, $1 \leqslant m_i < \infty$, and is defined as follows:

$$I = s^i_1 \cup s^i_2 \cup s^i_3 \cdots \cup s^i_{m_i}, \quad s^i_j \cap s^i_k = \phi, \ \forall j \neq k.$$

There also exists an evaluating function (f), which assigns to each segment or partition s^i_j of S^i, a segmentation–recognition score of v^i_j and a class label of c^i_j. The value v^i_j is a measurement of the confidence (likelihood, or membership) of segment s^i_j to be a valid digit in the class of c^i_j

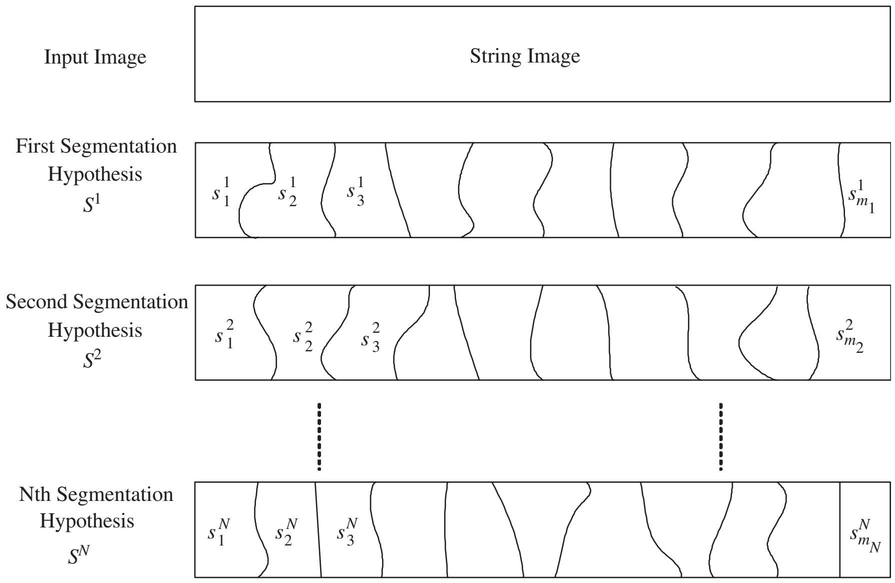

Input Image
String Image
First Segmentation Hypothesis S^1
s_1^1 s_2^1 s_3^1 $s_{m_1}^1$
Second Segmentation Hypothesis S^2
s_1^2 s_2^2 s_3^2 $s_{m_2}^2$
Nth Segmentation Hypothesis S^N
s_1^N s_2^N s_3^N $s_{m_N}^N$

Fig. 4. Illustrations of segmentation hypotheses for an input string image. $S^i : i = 1, \ldots, N$ shows segmentation hypotheses or partitionings of the numeral string, and s_k^i is the kth partition or segment of S^i.

as described below:

$$(c_j^i, v_j^i) = f(s_j^i), \quad j = 1, \ldots, m_i, \quad v_j^i \in [0, 1],$$
$$c_j^i \in \{0, 1, 2, \ldots, 9\}.$$

As shown in Fig. 4, among the set of all possible segmentation hypotheses (partitionings) of the image I, we are looking for the best segmentation or partitioning of S^i which *maximizes* our global objective function F as follows:

$$F(S^i) = \mathrm{Min}(v_1^i, v_2^i, \ldots, v_{m_i}^i).$$

This segmentation or partitioning (S^i) is called the optimal solution of our optimization problem, and the sequence of labels ($c_j^i, \ j = 1, \ldots, m_i$) corresponding to this solution will be the recognition result of the input numeral string as follows:

$$c_1^i, c_2^i, c_3^i, \ldots, c_{m_i}^i \quad \text{(A numeral string with } m_i \text{ digits).}$$

In our definition, instead of selecting the minimum objective function ($\mathrm{Min}_{j=1}^{m_i} v_j^i$), we could choose other objective functions such as product ($\prod_{j=1}^{m_i} v_j^i$), summation ($\sum_{j=1}^{m_i} v_j^i$), or average ($(1/m_i)\sum_{j=1}^{m_i} v_j^i$). In this paper, we use Min, and we try to maximize this objective function, in order to ensure that all the segments (partitions) of optimum segmentation hypothesis receive enough high segmentation–recognition scores (v_j^i). Combining partial scores (v_j^i) of the ith segmentation hypothesis by minimum function, also facilitates detection of the outlier patterns in that segmentation hypothesis. This will be explained in detail in Section 4.4.

4. Numeral string segmentation and recognition system

The general framework of our proposed system is presented in Fig. 5. Here, we briefly explain the function of each module. In the pre-processing module, images of the numeral strings are smoothed and their noises are removed according to the methods in Refs. [2,6]. Also, in order to make the segmentation task easier and the resulting cutting paths as straight and vertical as possible, a method similar to [21] is used to correct the slant of all the components of the input string. The segmentation module tries to introduce the best set of candidate cutting paths for the input numeral string, and passes those cutting paths as genes to the GA. The GA [22,23] searches the space of all possible segmentation hypotheses, and tries to find the optimum segmentation for the input numeral string. In the feature extraction module, some features and contextual information are extracted for each segmentation hypothesis. In the evaluation and classification module, each segmentation hypothesis is evaluated based on its features/contextual information, and to its segments, class labels and confidence scores are assigned. These confidence scores are used by the GA in order to find the optimum solution. The details of the main modules of this system are presented in the next subsections.

4.1. Segmentation

In Ref. [15], we proposed an algorithm to separate two touching digits in numeral strings. In this section, the algorithm is slightly modified such that it can segment numeral strings with unknown lengths. The goal of our segmentation algorithm is to introduce a super set of cutting paths in order to over-segment the string image. In fact, segmentation is a very complicated task, and there is a contradiction in its goal. For example, by introducing a set with a small number of cutting paths, there is a danger to lose or ignore some important cutting paths. On the other hand, by introducing a set with a large number of cutting paths, there is a danger to produce too many outlier or non-digit samples (which are very difficult to evaluate/reject), and also it will increase the computational cost of the system very rapidly. So in order to properly over-segment a numeral string, we require a set of cutting paths which gives us a high confidence, that contains all of the necessary cutting paths, and at the same time does not contain too many extra cutting paths which unnecessarily over-segment the components of the string. In order to reach a trade-off, a selected combination of contextual information from the string image and some local foreground/background information, extracted from each component, are utilized. The details of this process are explained in the following paragraphs.

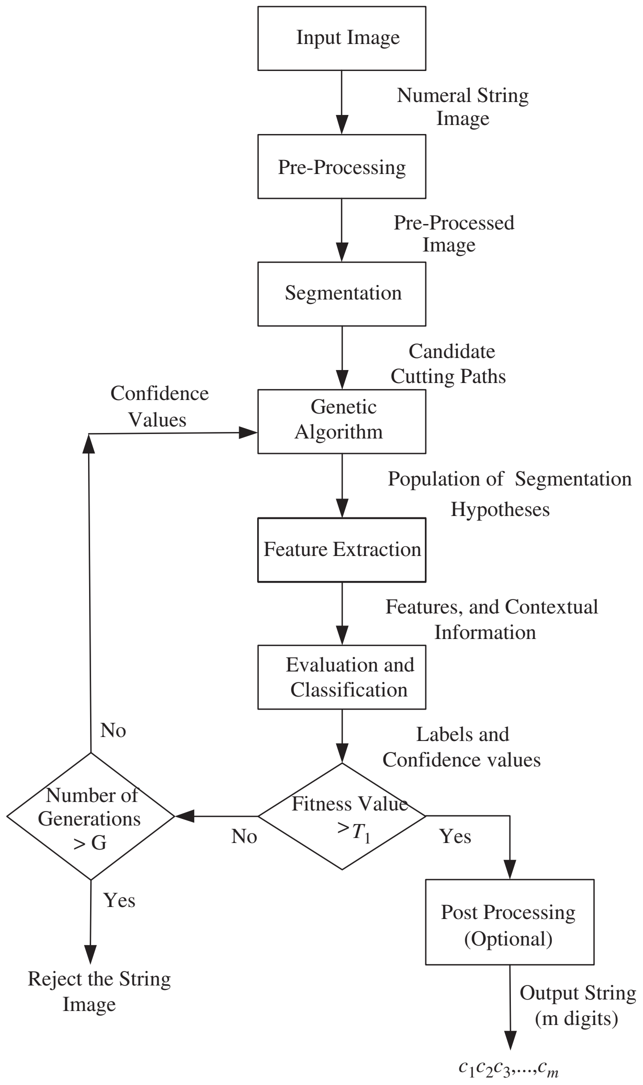


Fig. 5. General framework of our proposed system, for string recognition. T_1 and G are two threshold values, which can be adjusted by the user of our system.

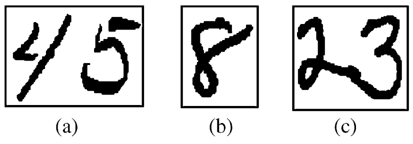


Fig. 7. Three types of connected components found in numeral strings: (a) parts of (broken) digits; (b) isolated digit; (c) two (or more) touching digits/components.

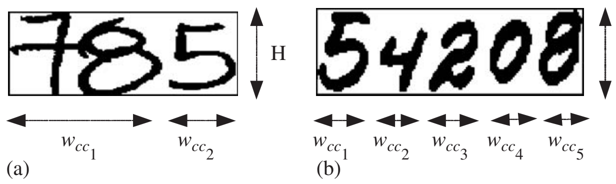


Fig. 8. Connected components whose width (w_{cc}) is less than 85% of the height of the string image (H), often do not require any further segmentation. CC_1 in (a) needs over-segmentation, but other CCs in (a), and (b) do not require any segmentation.

Our segmentation module consists of two essential parts: connected-components-analysis and touching-digits-splitting, which are shown in Fig. 6. In the connected-component-analysis, a string is separated into connected components (CCs). Our observations reveal that there are three possible types of CCs in a string image: pieces of broken digits, isolated digits, and touching components (or touching digits). Examples of these three component types are shown in Figs. 7a, b, and c, respectively. The first two types of CCs (7a and b) do not require any segmentation; however, the third type of CCs (7c) must be over-segmented by touching-digits-splitting module. Touching-digits-splitting module tries to introduce some cutting paths for each touching CC based on its foreground and background features. In the rest of this section, we explain how to identify touching CCs, and how to construct cutting paths for those CCs.

In Fig. 8, CCs of two sample numeral strings are shown, where the width of each CC is denoted by w_{cc}, and the global height of the string images (height of the bounding boxes) are denoted by H. In observing many string images from our database (such as those shown in this figure), we discovered that the third type of the CCs (touching components/digits) normally has a longer width than the two other types with respect to the height of the string image. So we employed

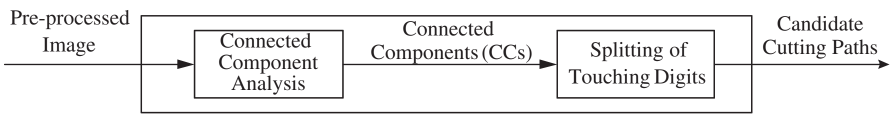

Pre-processed Image
Connected Component Analysis
Connected Components (CCs)
Splitting of Touching Digits
Candidate Cutting Paths

Fig. 6. Block diagram of the segmentation module.

a very helpful rule to avoid excess over-segmentation of the components: *If the width of a CC* (w_{cc}) *is less than* 85% *of the height of the image* (H), *then that CC is very likely to be a single digit* (*or a small piece of a digit*), *otherwise the CC is considered as a candidate of touching components*. Therefore, in the touching-digits-splitting module, all the CCs are checked based on this rule. CCs which are indicated as single digits or small CCs, using this rule, are not over-segmented. They will only be separated from their previous or next CCs in the string. CCs which are indicated as possible candidates of touching digits, will be over-segmented by the touching-digits-splitting module.

This preliminary segmentation step avoids over-segmentation of many isolated digits in the numeral strings, so it can save a great deal of computations. Also, it avoids making many mistakes resulting from the over-segmentation of the isolated digits. As an example of applying this rule, component 5 in Fig. 8a and all the components in Fig. 8b will avoid being over-segmented. Most of the other methods in the literature (such as [2,4]) do not use any similar rule in their segmentation stage, so after segmentation they are faced with a lot of pieces of wrongly over-segmented digits (outliers). Often, recognition or rejection of these pieces of digits in the later stages of the system by an isolated digit classifier is very costly or error prone.

For those CCs which are identified as possible candidates of touching digits, two types of features are extracted: foreground features, and background features. The details of our feature extraction methods, and construction of cutting paths are explained in the following sub-sections.

4.1.1. Generating foreground features

Foreground features are extracted from the black pixels of the image. To find foreground features, we introduce a new algorithm called skeleton tracing. First, by using a thinning algorithm [24], the skeleton of the CC is extracted, (see Fig. 9b). Then, on this skeleton, two points called starting and ending points (denoted by S, and E, respectively) are found, as described below. We start at the top-left corner of the skeleton image, and scan each column of the pixels from the top going downward, and we keep proceeding to the right until we encounter a black pixel on the skeleton. We declare this pixel as "the Starting point". In a similar way, we start at the top-right corner of the skeleton, and scan each column of the pixels from the top going downward, and we keep proceeding to the left, until we encounter a black pixel. We declare this pixel as "the Ending point". Then, from the starting point (S), the skeleton is traversed in two different directions: first clockwise, and then counter-clockwise, until both traversals reach the ending point (E), and they stop. In Fig. 9c, we define the traversal in the clockwise direction as top-skeleton, and the traversal in the counter-clockwise direction as bottom-skeleton. In the next paragraph, we explain how we can obtain important features of these skeletons for over-segmentation of the components.

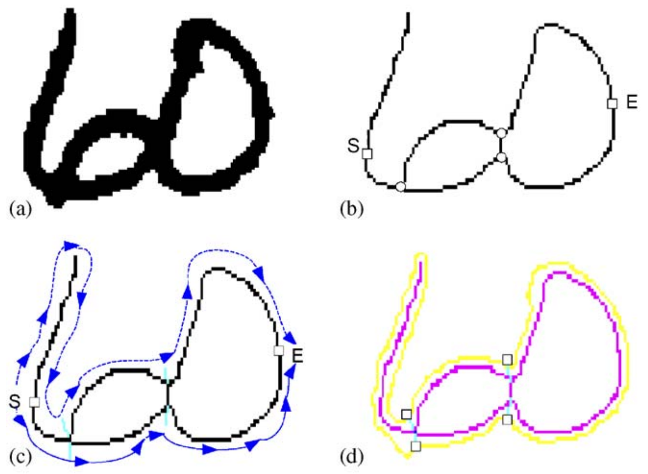


Fig. 9. (a) Pre-processed image; (b) foreground skeleton, starting point (S), ending point (E) are depicted by □, and intersection points (IPs) are depicted by ∘; (c) from starting point (S), skeleton is traversed in two different directions (clockwise: dashed arrows, and counter clockwise: dotted arrows) to the end point (E); (d) mapping of intersection points on the outer contours by bisectors to form foreground-features (denoted by □).

When the algorithm traverses the top/bottom skeletons, it looks for intersection points (IPs), which are visited on the skeletons. IPs are points that have more than two connected branches (they are denoted by ∘ in Fig. 9b). Corresponding to each visit of any IP in the skeletons, there is an angle where its bisector can be found (Fig. 9c). The intersections of these bisectors with the outer contour of the CC are obtained, and denoted by □ in Fig. 9d. These points are called our foreground feature points. In fact, in our algorithm, bisectors map IPs on the outer contour of the CCs in order to form foreground feature points. As shown in Fig. 9d, these feature points are very close to their corresponding IPs. Therefore, they can carry important information about the location of touching components or strokes.

4.1.2. Generating background features

In addition to foreground features, for over-segmenting of the touching components, we also extract background features. Lu et al. [12], and Chen and Wang [13] utilized the skeleton of the background to extract background features. In their methods, they used all the white pixels of the image as the background, including those inside the holes and inner parts of the digits. However, in our method, as seen in Fig. 10c, all the white pixels of the image are not considered as the background, and we introduce a new method of finding background features for a CC. First, vertical top and bottom projection profiles of the CC are found, as illustrated in Figs. 10d and e, respectively. Then, skeletons of these images (black regions) are extracted, which are shown in Figs. 10f and g, and they are called top-background-skeleton and bottom-background-skeleton, respectively.

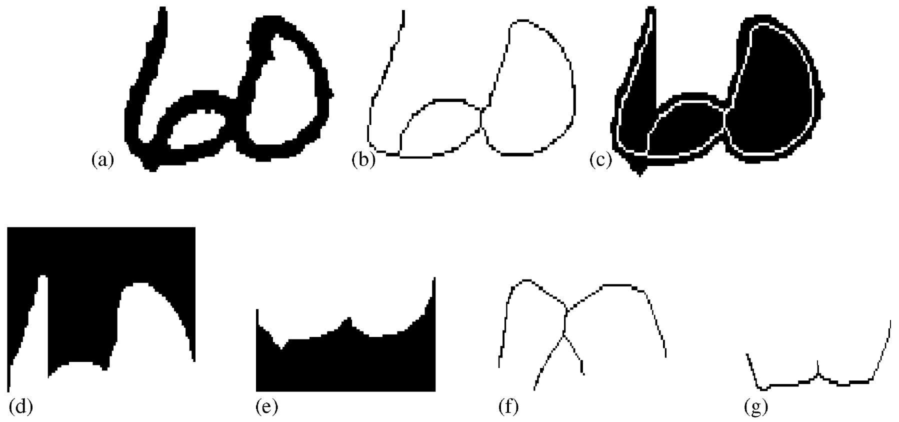


Fig. 10. (a) Pre-processed image; (b) foreground skeleton; (c) background region (white pixels outside of the black object); (d) top projection profile; (e) bottom projection profile; (f) top-background-skeleton; (g) bottom-background-skeleton.

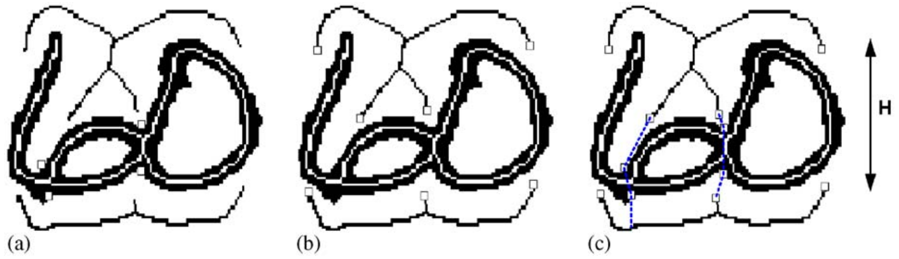


Fig. 11. (a) Foreground feature points denoted by □; (b) background feature points denoted by □; (c) feature points from the background and foreground (from top to bottom or bottom to top) are matched and assigned together to construct possible cutting paths.

In order to specify background features, on each background skeleton (top/bottom), end points are found. End points are points which have just one black neighbor pixel, and they are denoted by □ in Fig. 11b. The first and the last end points of each background skeleton will not be used and they are ignored. Compared to the methods of [12,13], the background region in our method only covers the essential part of the background of the components, so our feature points are more informative and stable, and also we have a smaller number of feature points in the background region. This facilitates the decision making for the construction of cutting paths.

4.1.3. Constructing segmentation paths

All the feature points found in the foreground and background are denoted by □ in Figs. 11a and b, respectively. These feature points from top to bottom, or from bottom to top, are assigned and connected together alternatively to construct all possible segmentation paths, according to the following rule: two feature points, A and B, are matched and assigned together if condition (1), described below is met

$$|x_A - x_B| < \alpha \cdot H, \quad \alpha \in [0.25, 0.5]. \tag{1}$$

Here, x_A and x_B are the horizontal coordinates of A and B, respectively. α is a constant parameter, which is set to 0.4 in our experiments, and H is the vertical height of the string image. Fig. 11c depicts an example of building possible cutting paths in a CC. More details about our segmentation algorithm, and the construction of cutting paths can be found in [15].

Various examples of segmentation of numeral strings, by our segmentation algorithm, will be presented in Section 5.1. These examples will show that our segmentation algorithm is an over-segmentation strategy, so its output normally provides some redundant cutting paths. Over-segmentation of a string is considered successful if it can introduce a super set of cutting paths with n cutting paths ($0 \leqslant n < \infty$), that includes the optimum set of cutting paths for that string (the optimum set of cutting paths is not yet known at this stage). After over-segmentation of a numeral string, all the produced cutting paths in the super set will be put in order from left to right based on the geometric location of their center of gravity (as shown in Fig. 12a). In the next section, by using these cutting paths, we will show how to obtain all possible segmentation hypotheses.

4.2. Generating segmentation hypotheses space

As a result of our segmentation algorithm, a string image is split into a sequence of primitive images, such that each segment of the image (primitive image) is expected to

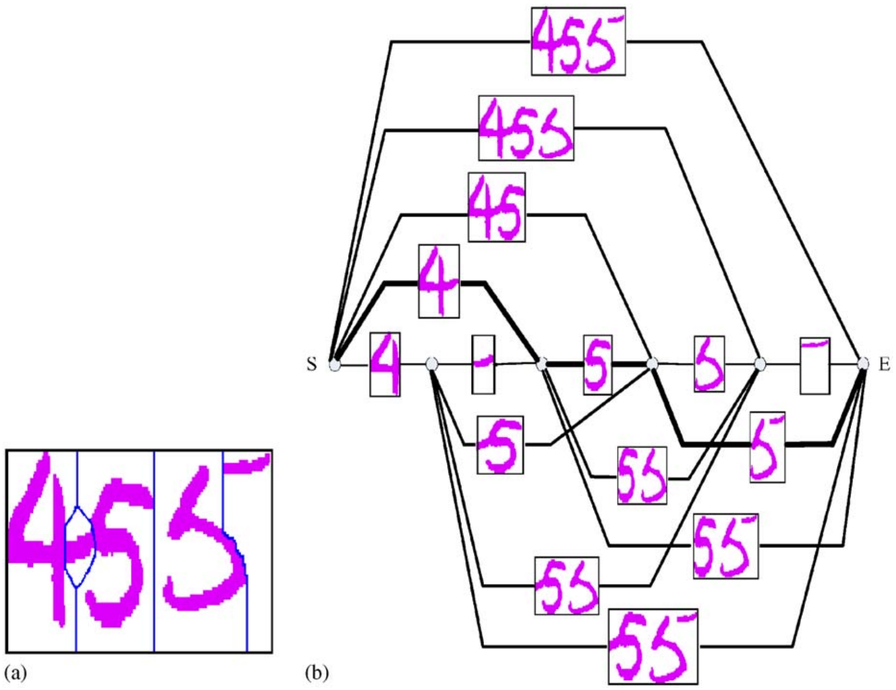


Fig. 12. (a) The original numeral string and its segmentation cutting paths; (b) segmentation candidate lattice for the numeral string in (a), where each node except for S and E, corresponds to a cutting path in (a), and each edge represents a candidate segment in (a). Terminal nodes (starting and ending nodes) are denoted by S, and E, respectively. Each path from S to E is a segmentation hypothesis; the optimum path from S to E is depicted by a thick path line. In (a), there are $n = 4$ cutting paths. Similarly, there are $n = 4$ non-terminal nodes, and $e = 15$ edges in the candidate lattice of figure (b). The total number of segmentation hypotheses (paths from S to E) is 16.

contain a digit or parts of a digit. The next task is to produce all possible sequences of these primitive images. In other words, we have to generate the space of all possible segmentation hypotheses which will be assessed/searched in the later stages. All consecutive combinations of the primitive images can be represented through a segmentation graph, called a candidate lattice. An example of this graph is shown in Fig. 12b. In a candidate lattice, nodes of the graph correspond to the cutting paths, and edges correspond to the pieces of the digits or sequences of the primitive images. There are two extra nodes which are called starting and ending nodes; these terminal nodes are denoted by S and E, respectively, in Fig. 12b.

Researchers in Refs. [2,4], also used candidate lattices to represent (generate) segmentation hypotheses in their systems, but they used heuristic rules to remove or reject some of the unlikely edges (or paths). For example, researchers in Ref. [4] made some assumptions about the maximum number of CCs in a path from S to E. They used some heuristic rules for the grouping of broken components, or they put constraints on the width or height of the candidate patterns in each edge of the graph. Since these kind of heuristic rules normally are biased to the training/testing sets, we decided not to use any heuristic rules for rejection of the paths in the segmentation lattice. After some investigations on the candidate lattices, we discovered some interesting facts about these graphs, which are explained in the following paragraph.

Since the lattice graphs in our system contain all the edges (candidate segments), we realized that these graphs are always complete graphs. In other words, each pair of the nodes of these graphs is connected by an edge. Therefore, if the total number of nodes (including starting and ending nodes) are denoted by v, then the number of edges in these graphs (e) are calculated as $e = \frac{1}{2}v(v-1)$. Here, v is equal to the number of cutting paths (n) plus two more nodes (starting and ending nodes), so v can be written as $v = n + 2$. If we substitute v in the above formula, we obtain the number of edges (e) in terms of the number of cutting paths (n) as in $e = \frac{1}{2}(n^2 + 3n + 2) = O(n^2)$. This is an important result, since it shows that in order to assign the weights or evaluation scores to all the edges of the candidate lattice graph, we have to invoke an evaluation function (such as a classifier) $O(n^2)$ times to produce the weights for all the edges of the graph. In Section 4.3, we also show that the total number of paths from S to E, (or the total number of segmentation hypotheses) in our candidate lattice graphs is always 2^n. These facts indicate the critical role of n (the number of cutting paths) in the computational cost/complexity of the problem.

4.3. Searching segmentation hypotheses space, using GA

Using an exhaustive search to find the optimum path (optimum segmentation hypothesis) in the lattice graph, while

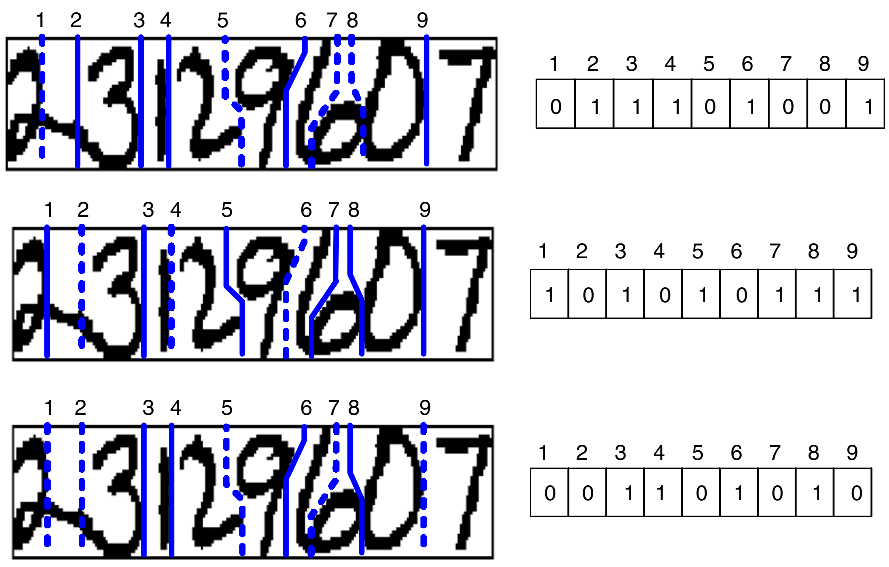


Fig. 13. Nine cutting paths can produce 512 segmentation hypotheses for the string image. Each hypothesis is encoded as a binary chromosome with length equal to 9. Dotted paths (inactive paths) are not considered in segmentation, and they are encoded as zeros in the chromosome. Solid paths (activepaths) are coded as ones, and are present in the segmentation of the numeral string. Each segmentation region or segment of the image is defined by the partial image from one solid cutting path (active gene) to the next solid cutting path (active gene). Each segment can contain either a valid digit or an outlier (over, or under-segmented pattern).

n is large, entails a very high computational cost, and it will be quite unfeasible. Many researchers used DP to find the optimum segmentation hypotheses in the lattice graphs of the numeral strings. For example, in Ref. [4], authors evaluated different paths in the graph by accumulating dissimilarity measures, produced by the isolated digit classifiers, along each path. Then, they used a DP search to find the optimal path in the graph. Since lengths of the numeral strings were unknown a priori, experiments showed that the results of the search are biased towards the short strings or short paths of the graph. Authors in Ref. [4], also attempted to use the average path scores with respect to the path lengths. Since the average path scores were not monotonic, it turned out that the search did not guarantee finding the global optimum path based on the average path scores.

In order to remedy the above-mentioned problems, we propose using a GA as a general search technique, in order to find the global optimum segmentation hypotheses in numeral strings. GAs are computationally simple and at the same time powerful, and they are known as robust search techniques for solving optimization problems in complex spaces which are not continuous, or their objective functions are multi-modal (not monotonic) [22,23]. They also have powerful operations such as selection or reproduction, crossover, and mutation, which can evolve the initial population of the candidate solutions to a better population of solutions in terms of the average fitness. However, successful application of GAs into a new problem domain generally is very dependent on the information representation, operator definition, and also on the corresponding evaluation scheme for that domain. Our information representation scheme and the operator definition are explained in this section, and the details of our evaluation scheme will be described in the next section.

All the paths from starting node (S) to ending node (E) in the candidate lattice (as shown in Fig. 12b) can be obtained by alternatively activating or deactivating the cutting paths in Fig. 12a (or corresponding graph nodes in Fig. 12b). In other words, in each path from S to E, some nodes or cutting paths are present and some are absent. This implies that each path from S to E in the graph (or equivalently each segmentation hypothesis) can be represented by a binary chromosome with n genes. Each gene corresponds to a cutting path or a node in the graph. In this representation, the starting node (S) and ending node (E) are not taken into account, since they are always present in all the paths. Another example is shown in Fig. 13, where the segmentation algorithm could find $n=9$ cutting paths, so there are $512 (=2^9)$ segmentation hypotheses in total for the string image. In this example, each segmentation hypothesis can be encoded as a binary chromosome with length $n = 9$ genes. Here, if a gene is equal to zero, then the corresponding cutting path is inactive (denoted by a dotted path), and it will not be considered in the segmentation of the image. On the other hand, if a gene is equal to one, then the corresponding cutting path is active (denoted by a solid path), and it will be considered in the segmentation of the image. In general, for n cutting paths in a string image, there are 2^n possible segmentation hypotheses in total, so the corresponding candidate lattice graph will have 2^n paths from the starting node (S) to the ending node (E). Each of those paths is represented by a binary chromosome containing n genes.

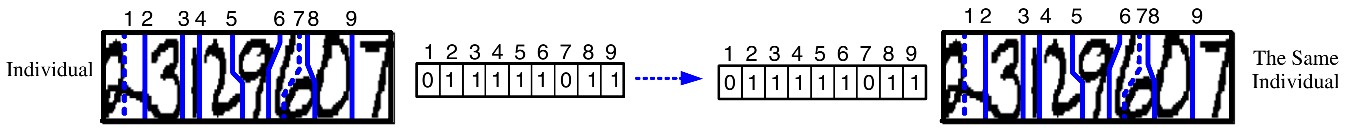


Fig. 14. Selection: very high fitness chromosomes are given a higher chance to reproduce themselves, and to be present in the new generations.

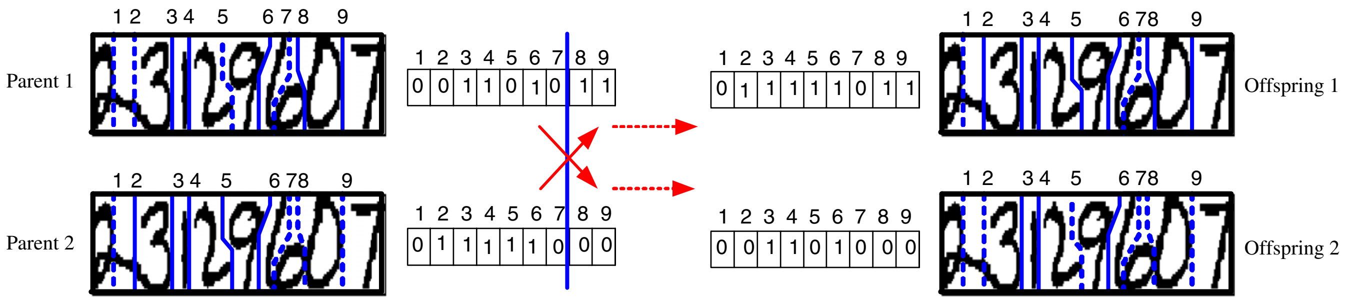


Fig. 15. Crossover: pairs of chromosomes are selected based on their relative fitness in order to perform crossover at a random position. In this example, crossover happens between the locations of the seventh and eighth genes in the two parent chromosomes, and it produces two new offsprings.

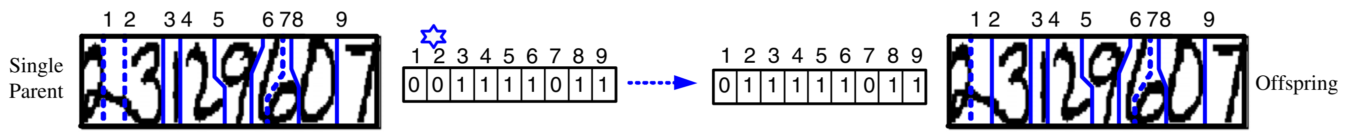


Fig. 16. Mutation: single chromosomes are selected based on their relative fitness in order to be mutated at a single random position. In this example, mutation happens on the second cutting path (second gene in the parent chromosome), and it produces one new offspring.

Representation of all segmentation hypotheses by binary chromosomes enables us to efficiently use all genetic operations such as selection, crossover, and mutation. Selection is a process by which the higher ranked chromosomes in the current generation are given higher chances to act as parents for the next generation. The crossover operator produces two offsprings (two candidate segmentations) by recombining the genetic structure of the two parents. Mutation is a random alteration of one or some genes in a single parent. In this paper, we apply the simplest versions of the genetic operations, such as fitness-proportionate (based on roulette-wheel) selection, and random single-point crossover or random single-point mutation on our binary chromosomes. Figs. 14, 15, and 16 show examples of selection, crossover, and mutation operations, respectively. As these examples show, the genetic operations are able to improve current segmentation hypotheses and change them into new offsprings with higher fitness/confidence levels. For more details on the genetic operations refer to [22,23]. The results of genetic operations in our method will always yield valid segmentation hypotheses (valid paths in the graph). As a result, we repeatedly can produce new generations of segmentation hypotheses in our genetic search. In each iteration of our GA, there is a population of chromosomes or hypotheses, which is evaluated and it goes through the genetic operations in order to generate the next population.

The representation scheme and operator definition that we presented here are very efficient and general. In fact, this representation scheme is able to generate (or represent) all the paths in a complete graph, and our genetic operators are also applicable to the paths of the graph. Therefore, our genetic search can easily be applied or adapted for solving any complicated graph searching problem in similar domains. In the next section, we will explain how to evaluate our chromosomes (segmentation hypotheses/paths in the lattice) in order to find their confidence scores.

4.4. *Evaluation of segmentation hypotheses*

Evaluation of segmentation hypotheses is a very important stage in numeral string recognition. At this stage, we have to rank segmentation hypotheses (paths in the candidate lattice). Due to uncertainty in the segmentation process, segmentation hypotheses may include isolated digits with different sizes or shapes, over-segmented (or broken) digits, or under-segmented (not-correctly separated) digits. In the literature, over-segmented and under-segmented parts of the digits are called outliers (out-of-class or non-digit patterns). An example of all segmentation hypotheses for a sample numeral string was shown in Fig. 12b, and each segmentation hypothesis was represented by a path from S to E. As seen in this figure, some paths contain outliers which are produced during the segmentation process. Those paths which contain at least one outlier pattern must be rejected or ranked with a very low confidence score.

Unlike handwritten word recognition, in the recognition of handwritten numeral strings, there is no dictionary information or linguistic context available. So, graphical models such as hidden Markov models (HMM), which are frequently used in word recognition, simply cannot be used to evaluate segmentation hypotheses in a candidate lattice for a numeral string (in some applications such as zip code recognition, there are some linguistic models used, but we do not consider these special applications here). In general, it is assumed that the constituent patterns in a segmentation hypothesis (or in a path) are independent from each other. Therefore, the majority of the researchers in the area of numeral string recognition only use the outputs of single character classifiers (similarity, dissimilarity, or probability measures) and assign scores independently to each constituent pattern (edge) in a path in the lattice graph. They combine these scores (by product, summation, average, etc.) to find a total score for ranking different segmentation hypotheses [2–4]. However, our experiments show that there are many cases where isolated digit classifiers are not able to evaluate or reject outliers in handwritten numeral strings. For example, all numeral strings in Fig. 1 have been over-segmented by the segmentation module, and the resulting outliers are very similar to actual digits. In these cases, it is very likely that an isolated digit classifier assigns higher confidence values to the outliers instead of to the correct candidates and mis-classifies outliers as valid digits. For instance, in Fig. 1a, it is possible that the wrong candidate (020) receives a higher recognition score than the correct candidate (20), or in Fig. 1b, it is possible that the wrong candidate (100) receives a higher confidence score than the correct candidate (60) by using only an isolated digit classifier.

The main reason for these kinds of mistakes can be explained as follows: isolated digit classifiers are trained to classify their inputs to one of the 10 classes (0–9). These classifiers normally classify input digits one by one and independent of the previous or the next components in their input. In other words, isolated digit classifiers normally do not pay attention to the geometric context of the digits/components in the numeral string images, such as relative positions, relative widths, or relative heights of the digits in the string. Therefore, if a high precision isolated digit classifier is fed with outliers, which are very similar to valid digits, it will very likely output valid digit class labels with high confidence values for those outliers, instead of rejecting them. In order to remedy this weakness of the isolated digit classifiers, and still use them with high reliability in numeral string recognition systems, we propose using a novel method based on using contextual information which uses a new set of scores called segmentation scores. These scores can support isolated digit classifiers, and can help them to avoid making many mistakes in numeral string recognition systems. Segmentation scores enable us to take into account the contextual geometric information from the string image, and to use this information in addition to the recognition information obtained from the isolated digit classifiers.

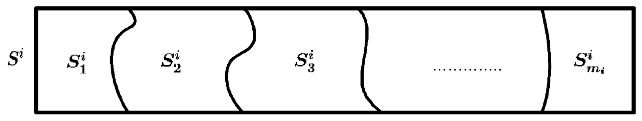


Fig. 17. A segmentation hypothesis or chromosome, which is denoted by S^i, consists of m_i regions (segments) ($1 \leqslant m_i < \infty$). Each segment is bounded by two consecutive active cutting paths. A combination of two scores (segmentation score and recognition score) are used to evaluate each segment in a segmentation hypothesis. By combining these scores, a total confidence value ($0 \leqslant \mathrm{conf}(S^i) \leqslant 1$) is assigned to this segmentation hypothesis.

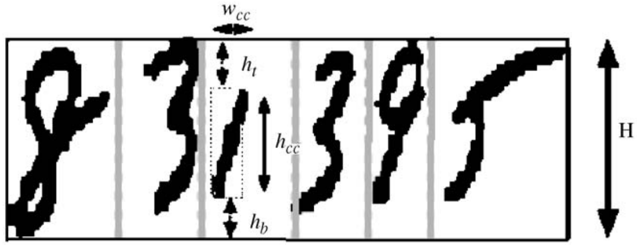
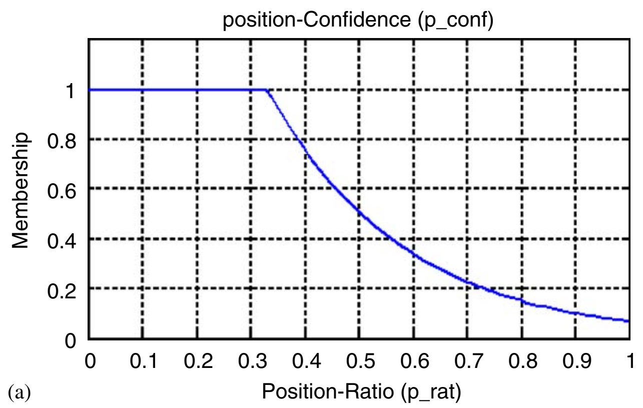


Fig. 18. Illustration of parameters used in the computations of position-ratio (*p_rat*), and aspect-ratio (*a_rat*) for the third segment of a string. For each segment (or CC) in the image, we can compute *p_rat*, and *a_rat*, and then we can calculate *p_conf*, and *a_conf*. Finally, *s_score* can be computed for each segment (or CC) according to Eq. (6).

We use Fig. 17 as a template, which shows a segmentation hypothesis for a numeral string. This segmentation hypothesis consists of m_i segmentation regions. Each region or segment is assumed to contain a valid isolated digit or an outlier, and it is assigned two different scores: a segmentation score, and a recognition score. We denote these two scores by s_score and r_score, respectively. Computation of these two scores for each region will be explained in detail in the next two sections.

4.4.1. Segmentation scores

Segmentation scores (*s_score*) are tools that help contextual knowledge to be taken into account in the evaluation and selection of the segmentation hypotheses. These scores are expressed as degrees of membership that show to what degree each connected component (or segment) in a numeral string can be fitted into a bounding box of a valid digit. Segmentation scores are calculated based on two other component scores: position-confidence (*p_conf*), and aspect-ratio-confidence (*a_conf*). Computations of *p_conf*, *a_conf*, and *s_score* are accomplished according to the Eqs. (2)–(6), listed below, and the parameters which are used in these equations are illustrated for the third segment (or third CC) of the numeral string in Fig. 18. As seen in this figure, the position-confidence (*p_conf*) evaluates



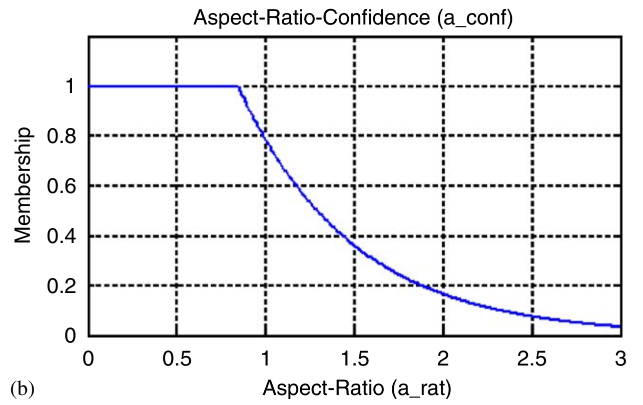


Fig. 19. (a) Graph of position-confidence (*p_conf*) membership function, (b) Graph of aspect-ratio-confidence (*a_conf*) membership function.

the position-ratio (*p_rat*) of each CC in the string image. The position-ratio (*p_rat*) is defined by Eq. (2), and it shows the relative height and vertical position of the bounding box of each CC in the string image. If a CC is very small, and its position is very close to the top or bottom of the bounding box of the image, it is assigned a low value of *p_conf*. Similarly, aspect-ratio-confidence (*a_conf*) evaluates the relative aspect-ratio (*a_rat*) of a CC, which is defined here as the ratio of the width of a CC (w_{cc}) to the height of the string image (H). If the width of a CC is greater than 85% of the height of the string (H), it is assigned a low value of *a_conf*. The membership functions of position-confidence (*p_conf*) and aspect-ratio-confidence (*a_conf*) are plotted in Fig. 19

$$p\_rat = \frac{\max(h_t, h_b)}{H}, \tag{2}$$

$$p\_conf(p\_rat) = \begin{cases} 1 & \text{if } p\_rat < \alpha, \\ \exp(-k(p\_rat - \alpha)) & \text{if } p\_rat \geqslant \alpha, \end{cases} \quad \alpha = \frac{1}{3},\ k = 4, \tag{3}$$

$$a\_rat = \frac{w_{cc}}{H}, \tag{4}$$

$$a\_conf(a\_rat) = \begin{cases} 1 & \text{if } a\_rat < \beta, \\ \exp(-t(a\_rat - \beta)) & \text{if } a\_rat \geqslant \beta, \end{cases} \quad \beta = 0.85,\ t = 1.45, \tag{5}$$

$$s\_score = \min(p\_conf(p\_rat),\ a\_conf(a\_rat)). \tag{6}$$

After computing the *s_score* for all the segments of a segmentation hypothesis (as shown in Eq. (6)), a *total_s_score* can be computed for that segmentation hypothesis, as shown in Eq. (7). Based on the definition of *s_score*, in the previous paragraph, we can easily verify that valid isolated digits or valid segmented digits in a numeral string will receive a much higher *s_score* than invalid outliers (such as those shown in Fig. 1). In fact, rejection of most of the outliers by relying only on recognition scores (like the methods in Refs. [2–4]), is very error prone. However, by using our segmentation scores, those outliers can easily be detected and rejected as follows: if the *total_s_score* of a segmentation hypothesis is lower than a threshold of T_2 (=0.45, which is determined empirically), we can conclude that it contains at least one outlier segment. Therefore, that segmentation hypothesis will be rejected, and the recognition scores (*r_score*) of its segments will not be computed (in fact those scores will be set to zero). This method also helps to avoid a great deal of computations which are required to evaluate/reject patterns in an invalid segmentation hypothesis. For the remaining segmentation hypotheses, recognition scores (*r_score*) are computed. The details of computations of recognition scores will be explained in the next section

$$total\_s\_score(S^i) = \min(s\_score(s_1^i),\ s\_score(s_2^i), \ldots, s\_score(s_{m_i}^i)). \tag{7}$$

4.4.2. Recognition scores

In addition to segmentation scores, we use recognition scores to reject the remaining outliers, and also to rank the segmentation hypotheses. By using an isolated digit classifier, recognition scores and class labels are assigned to the segments in the segmentation hypothesis. In our system, any isolated digit classifier (or a combination of classifiers), can be utilized. Achieving a very high accuracy for isolated digit classification is not the goal of our research, since this goal has already been investigated in the literature (see Ref. [25], for a survey on state-of-the-art techniques). Here instead, our goal is to improve the outlier resistance of the isolated digit classifiers, by using contextual knowledge, and to study its effect on the improvement of segmentation and recognition of numeral strings. Therefore, due to the efficiency in implementation and satisfaction in the performance, we tested our system, utilizing two different classifiers: an MLP neural network (MLP), and a support vector machine (SVM) with a radial basis function kernel (SVM_rbf). According to [25,26], MLPs, and SVMs normally have higher recognition rates in handwritten recognition than other classifiers with the same features.

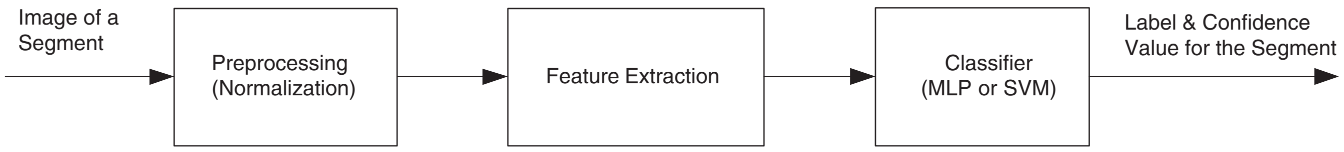


Image of a Segment
Preprocessing (Normalization)
Feature Extraction
Classifier (MLP or SVM)
Label & Confidence Value for the Segment

Fig. 20. The general structure of our recognition module. After pre-processing and feature extraction, a trained classifier assigns class labels and confidence values to each segment in the segmentation hypotheses.

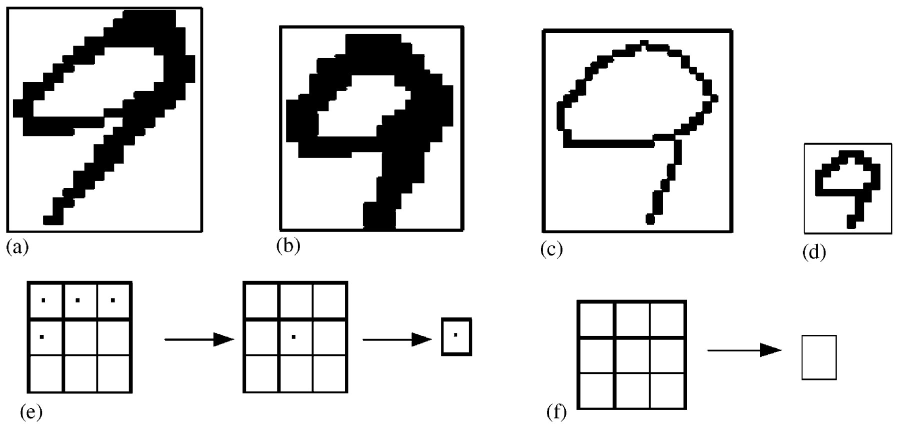


Fig. 21. An example of pre-processing and feature extraction: (a) original image; (b) pre-processed, slant corrected, and normalized image (45 × 45 pixels); (c) skeleton of part b; (d) reducing the resolution of the skeleton in horizontal and vertical directions by $\frac{1}{3}$; the resulting image is considered as a feature vector, (e and f) Two examples of the transformation which is used to reduce the variability of the pixels on the skeleton. In (e), all black pixels inside the windows are represented by a single pixel in the center of the window; in (f), all white pixels around the center are removed.

The general structure of our recognition module is depicted in Fig. 20. The details of preprocessing and feature extraction are shown in Fig. 21. Fig. 21b shows that the slants of all the segments in a segmentation hypothesis are corrected, and the size of all the segments are normalized into a matrix of size 45 × 45 [27]. Then, for feature extraction, the skeleton of each normalized segment is taken [24] (Fig. 21c), and it is divided into 15 × 15 zones, such that each zone is a window of 3 × 3 pixels. If there is at least one black pixel in a zone, its center pixel is set to black; otherwise, its center pixel will remain white. Then, all the pixels in a zone except for the center pixel will be removed. Two examples of this transformation are shown in Figs. 21e and f. Since different arrangements of the black pixels inside a zone (a window of 3 × 3) are replaced by just a single black or white pixel in the center, this transformation can greatly reduce the variations of the pixels in the skeletons of handwritten digits. Afterwards, the basic shapes of the digits are extracted, as in Fig. 21d. This image has 15 × 15 pixels, which is considered as a feature vector, and it is fed into the classifiers (MLP or SVM) for classification.

Here, we briefly explain the structure of our classifiers, while the details of their training are presented in Section 5.2. Our MLP classifier has three layers (225 neurons in the input layer, 85 neurons in the hidden layer, and 10 neurons in the output layer), and all the neurons in our network are sigmoidal neurons [28]. By using the back propagation learning algorithm [28], our MLP classifier is trained to produce labels and confidence values in the range of 0 and 1 for the segmented digits. For constructing our SVM classifier, we use 10 binary SVMs, one for separating each digit class from the other classes (so-called one-against-all strategy). Each binary SVM classifier uses a radial basis function kernel, and it produces an algebraic output as follows: a positive value corresponds to the target class samples (+1), and a negative value corresponds to the non-target class samples (−1). Similar to the methods in [3,29], these output values are mapped into confidence scores in the range of 0 and 1. The sigmoid function in Eq. (8) is used for this mapping, where $f(x)$ is the output of a binary SVM for the feature vector x, and $g(x)$ is the corresponding confidence value ($\in (0, 1)$)

$$g(x) = \frac{1}{1 + \exp(-f(x))}. \tag{8}$$

As mentioned above, each of our classifiers (MLP or SVM) has 10 outputs, corresponding to 10 classes (0–9), and each output produces a confidence value between 0 and 1 (so-called recognition score, denoted by *r_score*). For each classifier, the class which receives the highest recognition score is the winner, and it determines the label for the input segment. Recognition scores of all the segments in a segmentation hypothesis are combined based on Eq. (9), in order to produce a *total_r_score*. Having a low *total_r_score* for a segmentation hypothesis indicates that it contains at least one outlier segment, or a segment which our classifier cannot recognize very well

$$total\_r\_score(S^i) = \min(r\_score(s_1^i),\ r\_score(s_2^i), \ldots, r\_score(s_{m_i}^i)). \quad (9)$$

In summary, in Sections 4.4.1 and 4.4.2, we showed how to compute the $total\_s\_score$ and $total\_r\_score$ for each segmentation hypothesis. By combining these two scores, we are able to compute a confidence value for each segmentation hypothesis (S^i) based on Eq. (10). This confidence value expresses the total segmentation–recognition confidence for a segmentation hypothesis (or a chromosome S^i). It is used by our GA to evaluate the individuals in the population of segmentation hypotheses. Having a low confidence value for a segmentation hypothesis indicates that it contains at least one segment with a very low *s_score*, or a very low *r_score* (or both), and in all these cases that segmentation hypothesis must be rejected

$$Conf(S^i) = \min(total\_s\_score(S^i),\ total\_r\_score(S^i)). \quad (10)$$

5. Experimental results

The general outline of our system has been shown in Fig. 5. Our system has three main modules: segmentation, GA, and evaluation. In this section, we show the results of our experiments with each of these three parts, and we explain how to adjust their parameters. We show the results of our experiments on handwritten numeral string segmentation and recognition, and we compare our results with those that exist in the literature. All of the images used for our experiments have been taken from the NIST NSTRING SD19 Database, which contains unconstrained handwritten numeral strings with various lengths, and from the CENPARMI (Center for Pattern Recognition and Machine Intelligence) Database, which contains unconstrained handwritten isolated digits.

5.1. Testing the segmentation module

In the experiments, we first examined the performance of our segmentation module independent of other parts of the system (without using classification information). In order to have a comparison of our algorithm with similar algorithms in the literature, we did two sets of experiments. In the first set, we took 5000 touching pairs of digits from the NIST Database, and we input them to the algorithm. Experiments showed that in 96.5% of the touching cases, our algorithm was able to produce the correct cutting path. In the next set of the experiments, we randomly selected 1800 images, from the NIST NSTRING SD19 Database (for each string length of: 2, 3, 4, 5, 6, and 10, we took 300 string images). The lengths of the strings were not given to our algorithm, and they were considered unknown. After inputting these images, we looked at the output of the segmentation module, in order to verify the results. We defined segmentation (or over-segmentation) of a numeral string as successful if the set of cutting paths produced for that numeral string contained all the necessary cutting paths. Otherwise, segmentation was considered unsuccessful. A visual analysis revealed that in 98.04% of the cases, our segmentation algorithm could successfully over-segment the numeral strings. Figs. 22 and 23, respectively, show some successful and unsuccessful segmentation results produced by our segmentation algorithm. As shown in Fig. 23, most unsuccessful segmentations were due to a large amount of overlaps between the CCs in the strings. Most of the algorithms in the literature (such as [1,6,13,19], etc.) were proposed to split only touching pairs of digits, and they were not used for the segmentation of longer numeral strings (with unknown lengths). In contrast, our algorithm assumes an unknown number of isolated or touching components in the numeral strings. In addition, the majority of the segmentation algorithms in the literature use recognition information to select the best cutting paths. By contrast, the goal of our algorithm is to over-segment the numeral strings by introducing a super set of best cutting paths, and it does not make any final decision about those cutting paths. Although our algorithm uses neither recognition information, nor information about the lengths of the strings, Table 1 shows that the results of our segmentation algorithm compares favorably to similar algorithms in the literature.

After producing all the segmentation hypotheses for our numeral strings, we were able to collect a set of over and under-segmented components. These components are called outliers (out-of-class or non-digit) samples. In total, 1640 outlier samples were collected: 1154 samples of over-segmented outliers and 486 samples of under-segmented outliers. Some of these outlier samples are shown in Fig. 24. These outliers are used for two different purposes. Firstly, by using them we can adjust the parameters (α, k, t, and β) in Eqs. (3) and (5) in Section 4.4 for our segmentation scores. Secondly, these outlier samples are utilized to improve the training of our classifiers, which are explained in the next section.

5.2. Implementation of the classifiers

Using the CENPARMI handwritten isolated digit database and our collected outlier samples, our MLP and SVM classifiers were trained and their parameters were adjusted. The CENPARMI database has 4000 training samples (400 samples per digit) and 2000 testing samples (200 samples per digit). Some samples of the CENPARMI isolated digit database are shown in Fig. 25. Compared to other similar handwritten databases, samples in the CENPARMI database have more variations in their shapes and styles. We therefore selected this database for the training of our isolated digit classifiers. Researchers in Refs. [4,31], showed that adding

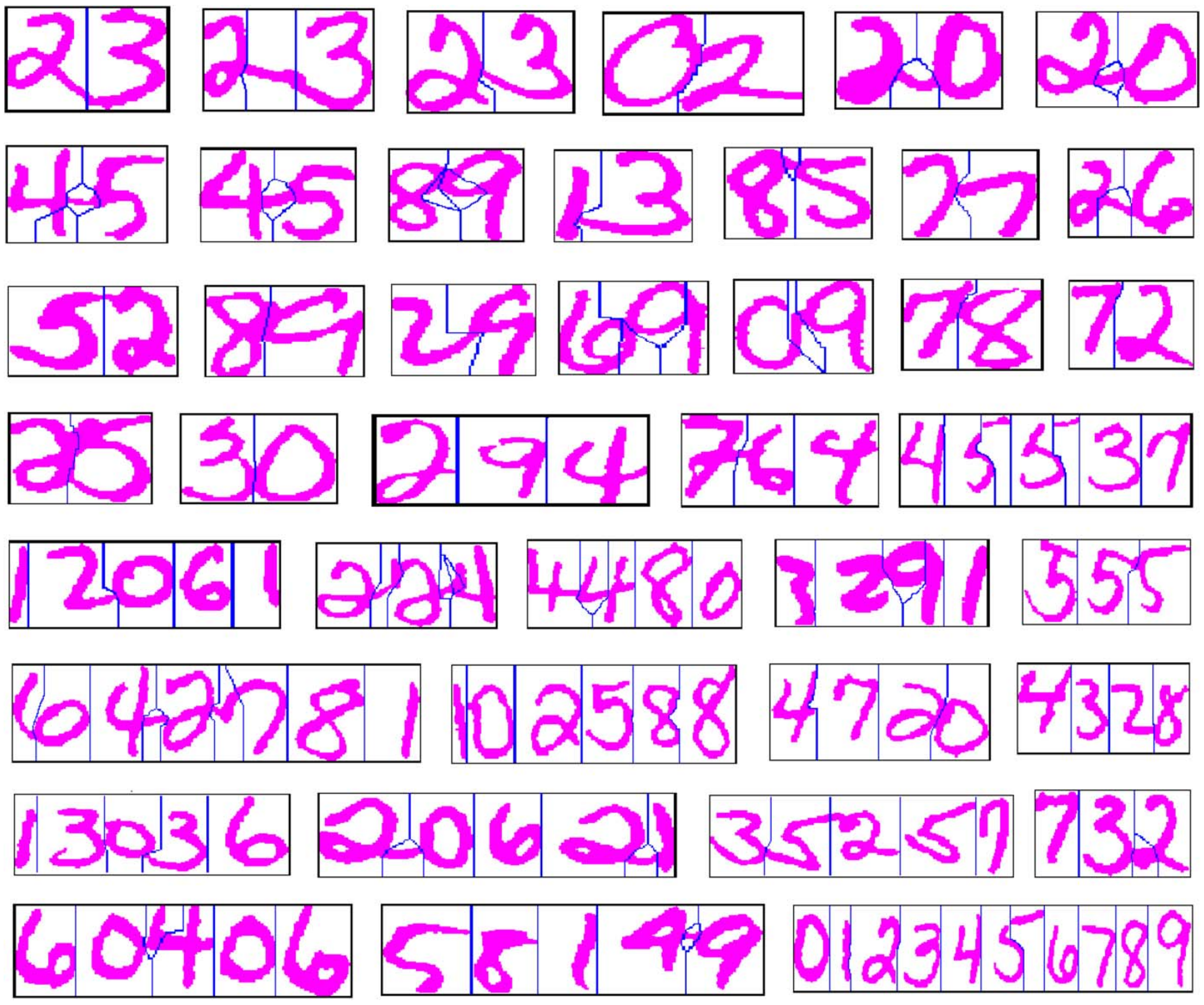


Fig. 22. Examples of successful segmentations (or over-segmentations) produced by our segmentation algorithm. Here, we do not use any recognition information. Bounding boxes contain real life string images from the Nstring SD19 Database with different lengths (2 to 10 digits). Each cutting path is a path from the top to the bottom of the bounding box or vice versa. As shown in this figure, some cutting paths have two or more branches.

outlier (or non-character) samples to the training set of isolated digit classifiers (such as MLP, SVM, ...) yields to higher performances in the classifier's rejection of outliers. In order to improve the outlier resistance of our classifiers, 1640 outlier samples (produced in the previous stage of our experiments) were added to the training samples of the CENPARMI database. However, unlike [4,31], we did not require a very large number of outlier samples to train our classifier to become outlier resistant. This is because most of the outlier samples were handled/rejected by our segmentation scores (which were introduced in Section 4.4.1), and not by our isolated digit classifier.

After some trial training of our classifiers (MLP, and SVM) the number of neurons in the hidden layer of the MLP was set to 85 (it has 225 and 10 neurons in the input and output layers, respectively). The parameters σ and C of the SVM classifier were also set to 1.15 and 10, respectively. The performances of our classifiers after training on the CENPARMI isolated handwritten digit database is presented in Table 2. As it shows, our MLP, and SVM classifiers could reach the recognition rate of 98.03% and 98.90% on the testing set of the CENPARMI isolated digit database. Table 2 also shows that the performances of our classifiers on the testing set of the CENPARMI database are a little lower than the results reported by [25] (using SVM_rbf). However, in the recognition of handwritten numeral strings, our system shows outlier resistance and a very good overall performance. We will discuss this issue and compare the performance of our system with other systems in the recognition of numeral strings in Sections 5.4 and 5.5.

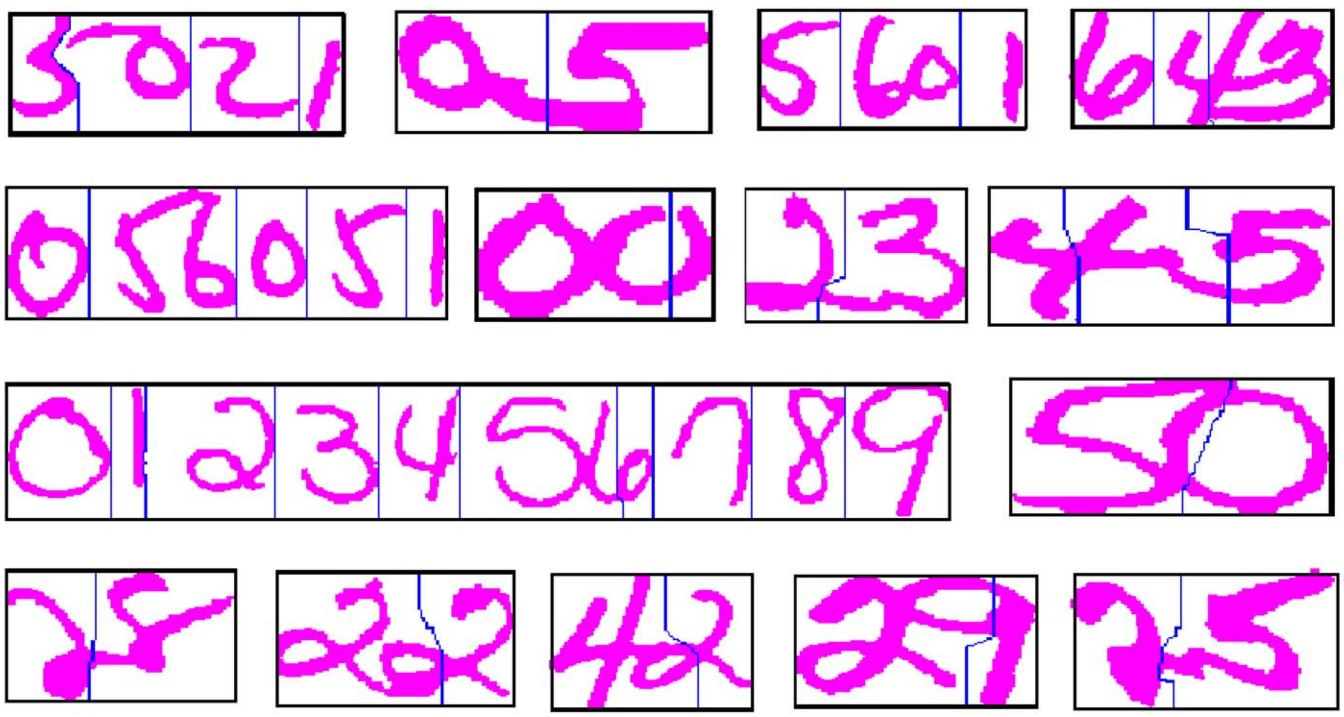


Fig. 23. Examples of unsuccessful cases of segmentation (without using any recognition information) by our segmentation algorithm.

Table 1
Performance comparison of different segmentation algorithms

Approach	Correct rate (%)	Error rate (%)	Reject rate (%)	Database	Using recognition information?
Strathy et al. [6]	89	5.8	N.A.	191 images of touching digits from USPS Database	Yes
Chi et al. [9]	95.1	4.9	After 32.7	3355 images of touching digits from NIST Database	Yes
Chi et al. [9]	89.2	10.8	After 2.8	3355 images of touching digits from NIST Database	Yes
Cheriet et al. [11]	80.8	19.2	0	120 images of touching digits from their own collection	No
Lu et al. [12]	97	3	After 28.6	3355 images of touching digits from NIST Database	Yes
Lu et al. [12]	92.5	7.5	After 4.7	3355 images of touching digits from NIST Database	Yes
Shi et al. [30]	95	5	0	495 zipcode images from CEDAR CD-ROM Database	No
Shi et al. [30]	85.7	14.3	0	2579 US zipcode images from US Postal Database	Yes
Oliveira et al. [19]	98.5	1.5	0	900 touching digits from Brazilian Bank Checks	No
Oliveira et al. [19]	95.24	2.14	2.62	900 touching digits from Brazilian Bank Checks	Yes
Chen and wang [13]	96	4	After 7.8	4178 images of touching digits from NIST Database and 332 images from their own collection	Yes
Our approach	96.5	3.5	0	5000 images of touching digits from NIST Database	No
Our approach (*Over segmentation*)	98.04	1.96	0	1800 images from NIST NSTRING SD19 Database (touching and non-touching strings with lengths of: 2,3,4,5,6, and 10 digits)	No

5.3. Implementation of GA and adjusting its parameters

The same samples of numeral strings that were used in the first stage of our experiments (1800 numeral strings) were also used to adjust the parameters of our GA. After some trial adjustment, the parameters of our GA were determined as shown in Table 3. In this table, the parameters P_r, P_c, P_m, M, and G represent the following: probability of reproduction (P_r), probability of crossover (P_c), probability of mutation (P_m), size of the initial population (M), and maximum number of generations (G). Here, n is the number of cutting paths generated by our segmentation algorithm. For short numeral strings where n is less than or equal to 5, the total number of possible segmentation hypotheses (2^n) will be less than or equal to 32. In these cases, our search algorithm conducts an exhaustive search on the population of segmentation hypotheses, instead of a GA search, but in the other cases (where $n \geqslant 5$) it conducts a GA search.

Our GA starts with a random population of chromosomes, and each chromosome has n binary genes. Then it searches

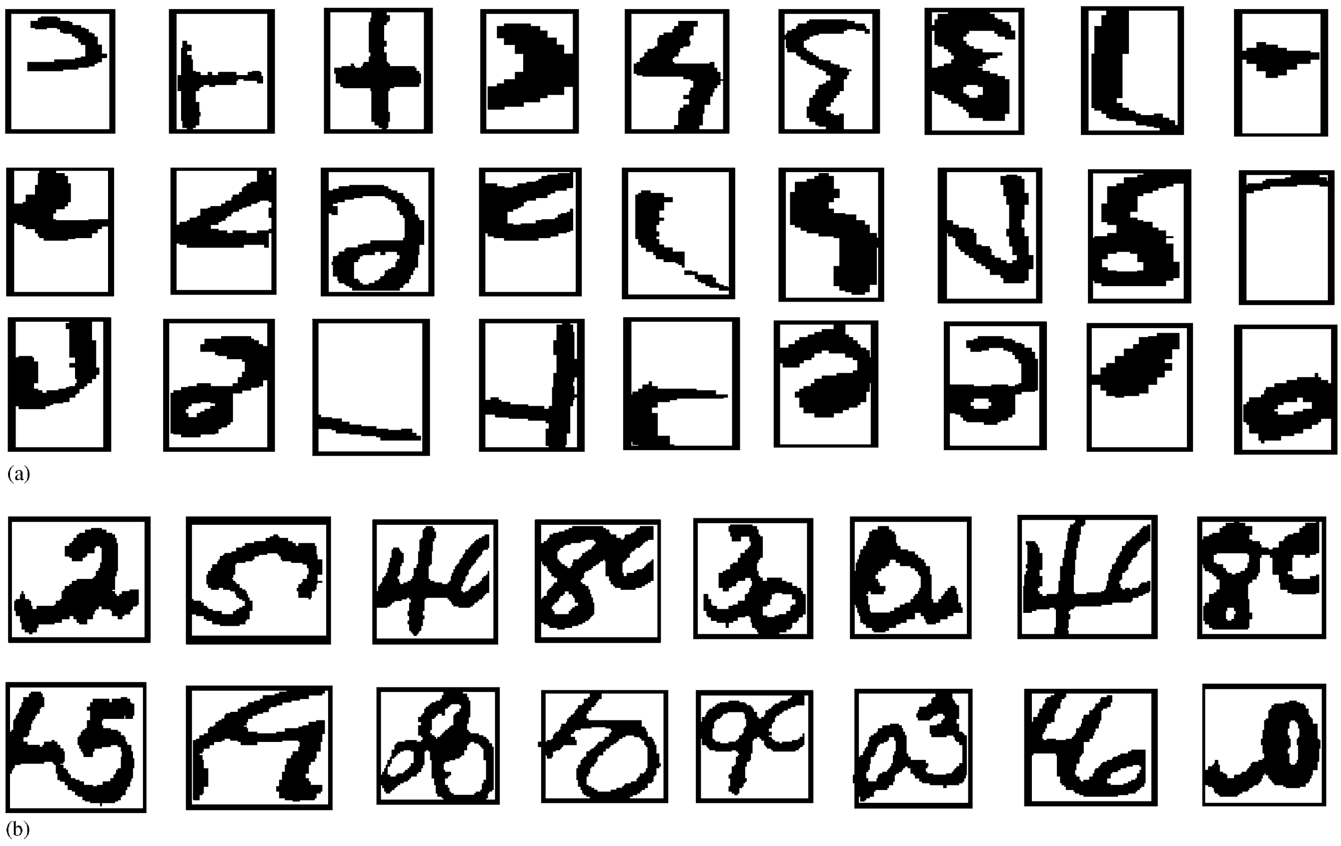


Fig. 24. Outlier samples: (a) over-segmented samples; (b) under-segmented samples.

for the optimum solution by updating the population using genetic operations. As shown in the flowchart in Fig. 5, the GA terminates in two cases: first, it terminates if a solution is found with desired fitness (a solution where its confidence is greater than a threshold of T_1). Second, it terminates if the number of iterations (generations) reaches the maximum number generations (G), and no other good solution has been found. For the former case, the algorithm accepts the string, and it outputs the labels and corresponding confidence values for the string. For the latter case, the algorithm rejects the input string. In our experiments, the rejection level of the system was controlled by adjusting the acceptance/rejection threshold (T_1). In our experiments, a typical value for T_1 was set to 0.80. In the next section, we will present the overall numeral string recognition results produced by our system.

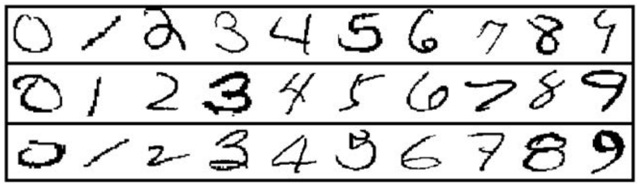


Fig. 25. Some samples of the CENPARMI isolated handwritten digit database.

5.4. Numeral string recognition results

Many researchers have used the NIST NSTRING SD19 database in order to evaluate their segmentation/recognition systems. For example, this database was used by researchers in Refs. [2,3,32]. Authors in Ref. [4], also used the NIST NSTRING SD19 database for some of their experiments, but they only reported the results on numeral strings with lengths 3 and 6, and they used fewer string images (1471 images, for each string length). In order to compare our results with all these researchers, we also used the same database for our experiments, and we used the same test images and string lengths which were used in Refs. [2–4,32] (Ref. [4] used only a subset of those test images). Table 4 summarizes the segmentation/recognition rates of our system on numeral strings of lengths 2, 3, 4, 5, 6, and 10 digits. In this table, we report the results of our system with two different isolated digit classifiers: MLP, and SVM and under two different conditions: first, without using segmentation scores, and using only the recognition scores (columns 3–6); second, using segmentation scores in addition to the recognition scores (columns 7–10). In the first case, segmentation hypotheses were evaluated based only on their recognition scores produced by the isolated digit classifiers (MLP, or SVM). In the latter case, in addition to the recognition

Table 2
Performance evaluation of our classifiers on CENPARMI isolated digit database

Database: CENPARMI	Our approach						Approach in [25]		
	MLP			SVM_rbf			SVM_rbf		
	Rec. (%)	Err. (%)	Rej. (%)	Rec. (%)	Err. (%)	Rej. (%)	Rec. (%)	Err. (%)	Rej. (%)
Training set (4000 samples + 1640 outlier samples)	99.81	0.19	0	99.92	0.08	0	N.A.	N.A.	N.A.
Testing set (2000 samples)	98.03	1.97	0	98.90	1.10	0	99.05	0.95	0

N.A. stands for not applicable.

Table 3
Parameter values for our genetic algorithm

P_r	P_c	P_m	M	G
0.4	0.55	0.05	32	$2^{\lfloor n/2 \rfloor}$

scores, contextual information (segmentation scores) were also used for the evaluation of segmentation hypotheses. In Table 4, the results of these two cases at zero rejection level can be compared for the MLP (in columns 3 and 7); and for the SVM (in columns 6 and 10). This comparison shows that using segmentation scores (contextual knowledge) improves the performance of the numeral string recognition system. The average improvement for the case of MLP was 7.85%, and for the case of SVM was 6.35%. As seen in Table 4, on average, our system could correctly segment, and recognize 95.28% of the test samples with MLP, and 96.42% of the test samples with SVM_rbf at zero rejection level. For the sake of simplicity and space, we also present the results of the recognition at two different levels of errors: less than 1%, and less than 0.5% (only for the MLP classifier). More importantly, we also compare our results using segmentation scores with the best results of similar approaches in the literature in Table 5. As this table shows, our results using segmentation scores (contextual knowledge) compare favorably to those existing in the literature [2–4,32].

Some successful and unsuccessful segmentation–recognition results of our system on handwritten numeral strings from NSTRING SD19 (hsf_7) are also shown in Figs. 26 and 27, respectively. As these two figures indicate, there are four possible outcomes for the output of a handwritten numeral string segmentation–recognition system: correct segmentation–correct recognition, wrong segmentation–wrong recognition, correct segmentation–wrong recognition, and wrong segmentation–correct recognition. The first and the last of these cases produce correct recognition results.

5.5. Discussion and comparison

Based on the results in Section 5.4, we herein discuss and compare the effects of classifier structure-training versus using contextual knowledge, on the performance of numeral string recognition systems. Columns 4–7 of Table 5 show recent examples of numeral string recognition systems where researchers focused only on improving isolated digit classifiers, in order to improve the overall system performance. For example, researchers in Ref. [2] used MLP neural networks to function as an isolated digit classifier and verifiers for the recognition of handwritten numeral strings, and later in Ref. [3] they substituted these MLP neural networks for support vector machines with radial basis function kernels (SVM_rbf). Comparison of their results in columns 4 and 5 of Table 5 shows that although SVMs have a higher generalization power (and a higher computational cost) than MLP neural networks in isolated digit recognition [25], substituting MLPs for SVMs could only improve the performance of a numeral string recognition system by an average of around 1%. Researchers in Ref. [4] also found that using different isolated digit classifiers or using different training strategies can yield different performances in handwritten numeral string recognition. Comparison of their results for two sample classifiers (MLP, and SVM_rbf) in columns 6, and 7 of Table 5, show that SVM_rbf has about 1.16–1.22% higher performance in numeral string recognition than MLP Neural Networks. These results are consistent with the results of [2,3]. These two examples of systems for numeral string recognition show that there is a possibility to improve the performance of the system slightly, by enhancing the isolated digit classifier module. However, in our experiments, unlike [2–4], in each case we kept the isolated digit classifier (MLP, or SVM) unchanged, and we only added the segmentation scores (as a source of contextual knowledge) to the system, and we were able to increase the performance of our system, to about 7.85% in the case of MLP, and 6.35% in the case of SVM, which are very significant improvements. The reason for these improvements can be explained as follows: isolated character classifiers (such as SVM, MLP, etc.) are designed, and trained in order to recognize single and independent components one at a time, so normally they do not use the contextual knowledge (such as comparisons of adjacent components, or comparison of relative sizes or relative positions of different components) in their decisions. The lack of using contextual knowledge, which is easily available in numeral strings, can cause isolated digit classifiers

Table 4
Recognition rates of our system on the numeral strings from NSTRING SD19, using two different classifiers: MLP and SVM, and in two different situations: without and with segmentation scores (contextual knowledge)

Col no. 1	Col no. 2	Col no. 3	Col no. 4	Col no. 5	Col no. 6	Col no. 7	Col no. 8	Col no. 9	Col no. 10
String length	Number of strings	Without segmentation scores				With segmentation scores			
		MLP			SVM	MLP			SVM
		0% (rej)	1% (err)	0.5% (err)	0% (rej)	0% (rej)	1% (err)	0.5% (err)	0% (rej)
2	2370	92.06	87.05	85.56	95.05	97.87	92.20	90.77	98.94
3	2385	89.39	84.79	83.04	91.43	96.43	90.06	87.18	97.23
4	2345	88.20	83.63	81.00	91.07	95.17	87.91	85.04	96.16
5	2316	85.75	81.91	79.05	88.05	94.91	86.97	84.38	95.86
6	2169	85.64	81.02	78.69	88.69	94.26	83.01	80.41	96.10
10	1217	83.51	79.75	78.13	86.13	93.01	81.12	79.05	94.25
Average rates	–	87.43	81.56	80.91	90.07	95.28	86.88	84.47	96.42

Table 5
Comparison of the recognition rates of our approach (using segmentation scores) with similar approaches on the test samples of NIST NSTRING SD19

Col no. 1	Col no. 2	Col no. 3	Col no. 4	Col no. 5	Col no. 6	Col no. 7	Col no. 8	Col no. 9
String length	Number of of strings	Results by [32]	Results by [2] (MLP)	Results by [3] (SVM)	Results by [4]		Our approach	
					MLP	SVM	MLP	SVM
2	2370	94.8	96.88	97.67	–	–	97.87	98.94
3	2385	91.6	95.38	96.26	95.60	96.82	96.43	97.23
4	2345	91.3	93.38	94.28	–	–	95.17	96.16
5	2316	88.3	92.40	94.00	–	–	94.91	95.86
6	2169	89.1	93.12	93.80	95.58	96.74	94.26	96.10
10	1217	86.9	90.24	91.38	–	–	93.01	94.25
Average rates	–	90.33	93.57	94.57	–	–	95.28	96.42

(even with a very high recognition power) to misrecognize many outliers, and to treat them as valid digits. This can explain why boosting the isolated digit classifiers in a numeral string recognition system is not the most effective, nor the most unique solution in order to improve the overall performance of the system.

Researchers in Ref. [4] also showed that training the isolated digit classifiers with many outlier samples (in addition to the original training samples), can improve the outlier resistance of those isolated digit classifiers, and subsequently this can improve the performance of those classifiers in handwritten numeral string recognition systems. However, our experiments showed that there were many outlier samples that could not be recognized/rejected by isolated digit classifiers, no matter whether those classifiers have been trained or not by the outlier patterns. Also, in order to train classifiers with outlier samples, a set with a very large number of outlier patterns is required. Currently, such a standard set of outlier samples is not available. Therefore, researchers have had to use their own collection of outlier samples.

In summary, in systems which do not use segmentation scores (sources of contextual knowledge), correct segmentation–recognition of numeral strings only relies on the accuracy, and the outlier resistance of the classifier. So, these systems are very dependent on the structure and training method of the isolated digit classifier. In comparison, in our system, correct segmentation–recognition of numeral strings depends on two different factors: first, the accuracy/outlier resistance of the classifier, and second, the segmentation scores. Since segmentation scores are independent of the classifier and they come from another source (contextual information), in our system the classifier has a less crucial role in the recognition performance, and this can also improve the reliability of the system.

6. Conclusion and future works

Segmentation and recognition of unconstrained handwritten numeral strings is one of the most difficult problems in the area of optical character recognition (OCR). In this paper, we formulated the segmentation and recognition of handwritten numeral strings as an optimization problem, and we proposed three novel contributions for solving this problem. First, a genetic framework was proposed for the segmentation and recognition of unconstrained handwritten numeral strings. Our framework allows for flexibility, for instance, any algorithm or a combination of algorithms can be chosen for each task such as the segmentation task,

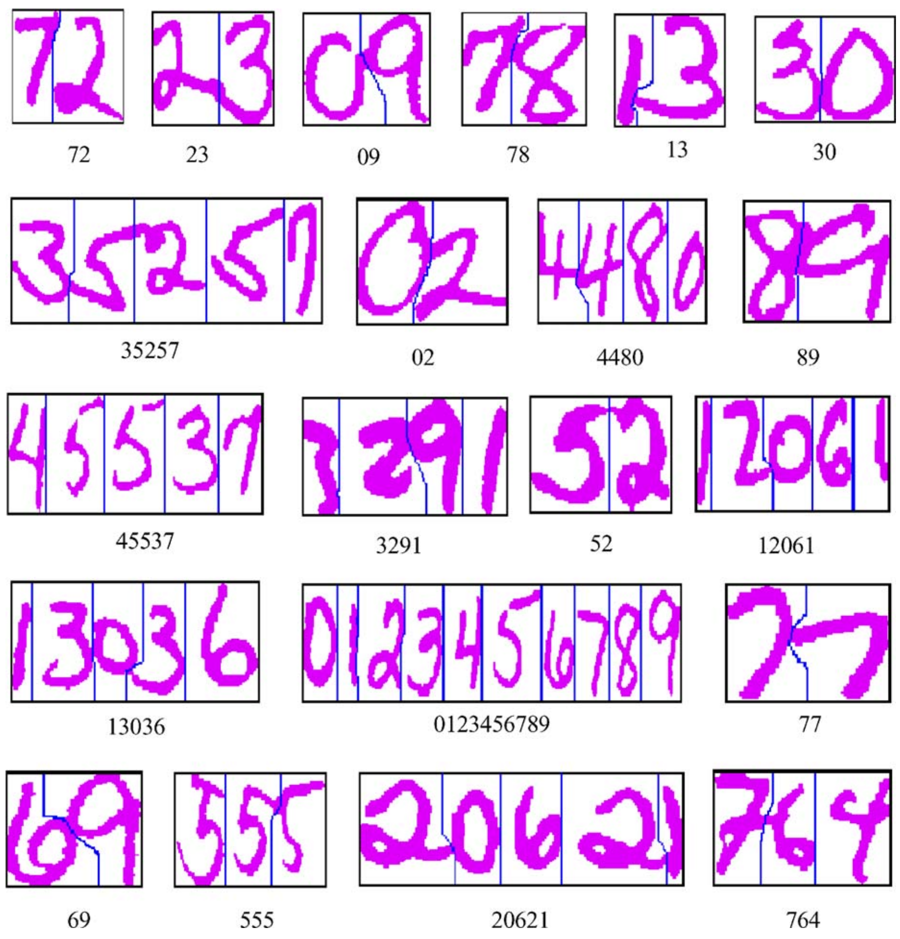


Fig. 26. Successful examples of segmentation/recognition produced by our system. Numeral strings are taken from the NSTRING SD19 Database.

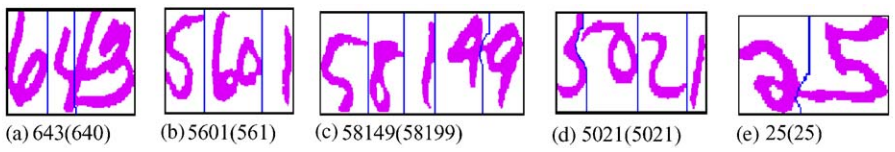


Fig. 27. Unsuccessful examples of segmentation/recognition produced by our system. (a) and (b) Wrong segmentation–wrong recognition; (c) correct segmentation–wrong recognition, (d) and (e) wrong segmentation–correct recognition.

feature extraction task, or evaluation task. Compared to a conventional model that does not provide any recognition feedback for segmentation, our method provides a feedback loop of recognition/segmentation and enables us to use different sources of recognition and contextual information for the evaluation of segmentation hypotheses. Our novel genetic representation scheme is very efficient, and it can also be utilized in solving very large complicated graph searching problems. Secondly, we introduced new algorithms to extract segmentation features, in order to segment numeral strings with unknown lengths. For extracting foreground features, we introduced a new technique called skeleton tracing, and for extracting background features, we introduced a new method based on the thinning of projection profiles. Compared to similar algorithms in the literature, our segmentation algorithm is more sophisticated, and in most of the cases it is able to produce a set of the best candidate cutting paths for the input strings. These cutting paths are used as building blocks (genes) of the final solution. Finally, we introduced segmentation scores, which allow us to efficiently extract contextual information from the string image. These scores are used in addition to recognition scores, and they help us to incorporate the information about the shapes, relative sizes or positions of the CCs in the evaluation of the segmentation hypotheses. We showed that by employing segmentation scores (using contextual knowledge), conventional isolated digit classifiers can reach a very high performance in handwritten numeral string segmentation/recognition. Segmentation scores can also help us to avoid considerable computation, which is normally required to reject outlier patterns.

Our string segmentation/recognition results on the numeral strings from NSTRING SD19 compare favorably to the ones reported in the literature. Our experiments showed that having a good segmentation algorithm, and also utilizing both recognition and contextual information from the string image, are the keys to success in handwritten numeral string recognition systems. In the future, we would like to improve the performance and efficiency of our segmentation algorithm and the robustness of our evaluation method by better exploiting all sources of contextual knowledge, and by applying advanced machine learning techniques such as combining classifiers.

Acknowledgements

The authors would like to thank NSERC (Natural Sciences and Engineering Research Council of Canada), and FQRNT (Fonds de Recherche sur la Nature et les Technologies du Quebec) for their financial support, Ms. Shira Katz for her valuable editorial assistance, and also the anonymous reviewers of this paper for their helpful comments in improving the earlier draft of this paper.

References

[1] U. Pal, A. Belaid, Ch. Choisy, Touching numeral segmentation using water reservoir concept, Pattern Recognition Lett. 24 (2003) 261–272.
[2] L.S. Oliveira, R. Sabourin, F. Bortolozzi, C.Y. Suen, Automatic segmentation of handwritten numerical strings: a recognition and verification strategy, IEEE Trans. Pattern Anal. Mach. Intell. 24 (11) (2002) 1438–1454.
[3] L.S. Oliveira, R. Sabourin, Support vector machines for handwritten numerical string recognition, Proceedings of the International Workshop on Frontiers in Handwriting Recognition (IWFHR-9), Tokyo, Japan, 2004, pp. 39–44.
[4] C.L. Liu, H. Sako, H. Fujisawa, Effects of classifier structure and training regimes on integrated segmentation and recognition of handwritten numeral strings, IEEE Trans. Pattern Recognition Mach. Intell. 26 (11) (2004) 1395–1407.
[5] J. Sadri, C.Y. Suen, T.D. Bui, New approach for segmentation and recognition of handwritten numeral strings, document recognition and retrieval XII, Proceedings of SPIE-IS&T Electronic Imaging, SPIE, vol. 5767, 2005, pp. 92–100.
[6] N.W. Strathy, C.Y. Suen, A. Krzyzak, Segmentation of handwritten digits using contour features, in: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR93), 1993, pp. 577–580.
[7] K.K. Kim, J.H. Kim, C.Y. Suen, Segmentation-based recognition of handwritten touching pairs of digits using structural features, Pattern Recognition Lett. 23 (2002) 13–24.
[8] R.G. Casey, E. Lecolinet, A survey of methods and strategies in character segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 18 (7) (1996) 690–706.
[9] Z. Chi, M. Suters, H. Yan, Separation of single and double touching handwritten numeral strings, Opt. Eng. 34 (1995) 1159–1165.
[10] Z. Shi, V. Govindaraju, Segmentation and recognition of connected handwritten numeral strings, in: Progress in Handwritten Recognition, World Scientific, Singapore, 1996, pp. 515–518.
[11] M. Cheriet, Y.S. Huang, C.Y. Suen, Background region based algorithm for the segmentation of connected digits, in: Proceedings of the International Conference on Pattern Recognition (ICPR92), 1992, pp. 619–622.
[12] Z. Lu, Z. Chi, W. Siu, P. Shi, A background-thinning-based approach for separating and recognizing connected handwriting digit strings, Pattern Recognition 32 (1999) 921–933.
[13] Y.K. Chen, J.F. Wang, Segmentation of single or multiple touching handwritten numeral strings using background and foreground analysis, IEEE Trans. Pattern Anal. Mach. Intell. 22 (2000) 1304–1317.
[14] S. Choi, I. Oh, A segmentation-free recognition of two touching numerals using neural networks. in: Proceedings of 5th ICDAR (ICDAR'99), Bangalore, India, 1999, pp. 253–256.
[15] J. Sadri, C.Y. Suen, T.D. Bui, Automatic segmentation of unconstrained handwritten numeral strings, in: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR 2004), Tokyo, Japan, October 2004, pp. 317–322.
[16] M. Shridhar, A. Badrelin, Recognition of isolated and simply connected handwritten numerals, Pattern Recognition 19 (1) (1986) 1–12.
[17] H. Nishida, S. Mori, A model-based split-and-merge method for character string recognition, in: P.S.P. Wang, (Ed.), Document Image Analysis, World Scientific, Singapore, 1994, pp. 209–226.
[18] O. Matan, C.J.C. Burges, Recognition of overlapping handprinted characters by centered-objects integrated segmentation and recognition, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN'91), 1991, pp. 504–511.
[19] L.S. Oliveira, E. Lethelier, F. Bortolozzi, R. Sabourin, A new approach to segment handwritten digits, Seventh International Workshop on Frontiers in Handwritten Recognition (IWFHR-7), Amsterdam, September 2000, pp. 577–582.
[20] X. Wang, V. Govindaraju, S. Srihari, Holistic recognition of touching digits, in: The Proceedings of Sixth International Workshop on Frontiers in Handwriting Recognition (IWFHR'98), 1998, pp. 295–303.
[21] T. Yamaguchi, Y. Nakano, M. Maruyama, H. Miyao, T. Hananoi, Digit classification on signboards for telephone number recognition, Proceedings of the Seventh International Conference on Document Analysis and Recognition, 2003, pp. 359–363.
[22] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Longman Inc., Reading, MA, New York, 1989.
[23] J.R. Koza, Genetic Programming—On The Programming of Computers by Means of Natural Selection, The MIT Press, Cambridge, 1992.
[24] T.Y. Zhang, C.Y. Suen, A fast parallel algorithm for thinning digital patterns, Commun. ACM 27 (3) (1984) 236–239.
[25] C.L. Liu, K. Nakashima, H. Sako, H. Fujisawa, Handwritten digit recognition: benchmarking of state-of-the-art techniques, Pattern Recognition 36 (2003) 2271–2285.
[26] J. Dong, A. Krzyzak, C.Y. Suen, Fast SVM training algorithm with decomposition on very large training sets, IEEE Trans. Pattern Anal. Mach. Intell. 27 (4) (2005) 603–618.
[27] C.L. Liu, K. Nakashima, H. Sako, H. Fujisawa, Handwritten digit recognition: investigation of normalization and feature extraction techniques, Pattern Recognition 37 (2004) 265–279.
[28] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 2003.
[29] G. Wahba, X. Lin, F. Gao, D. Xiang, R. Klein, B. Klein, The bias-variance trade-off and the randomized GACV, Proceedings of the 13th Conference on Neural Information Processing Systems (NIPS), Vancouver, Canada, 2001, pp. 8–31.

[30] Z. Shi, S.N. Srihari, Y.-C. Shin, V. Ramanaprasad, A system for segmentation and recognition of totally unconstrained handwritten numeral strings, Proceedings of the International Conference on Document Analysis and Recognition (ICDAR97), vol. 2, 1997, pp. 455–458.
[31] C.L. Liu, H. Sako, H. Fujisawa, Integrated segmentation and recognition of handwritten numerals: comparison of classification algorithms, Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR'02), Ontario, Canada, 2002, pp. 303–308.
[32] A.S. Britto, R. Sabourine, F. Bortolozzi, C.Y. Suen, Recognition of handwritten numeral strings using a two-stage HMM-based method, Int. J. Doc. Anal. Recognition 5 (2–3) (2003) 102–117.