IJDAR (1999) 2: 90–110

International Journal on Document Analysis and Recognition IJDAR
© Springer-Verlag 1999

Offline cursive script word recognition – a survey

Tal Steinherz[1], **Ehud Rivlin**[2], **Nathan Intrator**[1]

[1] School of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel-Aviv University, Ramat Aviv 69978, Israel; e-mail: {talstz,nin}@math.tau.ac.il
[2] Department of Computer Science, Technion, Technion City 32000, Israel; e-mail: ehudr@cs.technion.ac.il

Received September 21, 1998 / Revised September 2, 1999

Abstract. We review the field of offline cursive word recognition. We mainly deal with the various methods that were proposed to realize the core of recognition in a word recognition system. These methods are discussed in view of the two most important properties of such a system: the size and nature of the lexicon involved, and whether or not a segmentation stage is present. We classify the field into three categories: segmentation-free methods, which compare a sequence of observations derived from a word image with similar references of words in the lexicon; segmentation-based methods, that look for the best match between consecutive sequences of primitive segments and letters of a possible word; and the perception-oriented approach, that relates to methods that perform a human-like reading technique, in which anchor features found all over the word are used to bootstrap a few candidates for a final evaluation phase.

Key words: Offline – Cursive – Handwritten – Word recognition – Segmentation – Survey

1 Introduction

The field of offline cursive word recognition has made great progress during the past ten years. Many methods have been developed in an attempt to satisfy the need for such systems that exists in various applications like automatic reading of postal addresses and bank checks, processing documents such as forms, etc.

Most of these methods, while presenting a large spectrum of perspectives on the problem, share a common structure, having the same modules. In the flow chart given in Fig. 1, three common alternative structures of word recognition systems are presented. The typical modules are preprocessing, then a possible segmentation or fragmentation phase, feature extraction, the core of recognition, and post-processing. Preprocessing usually includes normalization, noise reduction, reference line finding, and either contour or skeleton tracing if necessary. Next, there is the segmentation phase and its substitutes. In a segmentation process, in contrast with simple fragmentation or splitting into pieces, there is an attempt to split the word image into segments that relate to characters. Some methods prefer to avoid segmentation altogether for reasons that will be discussed later on. In the latter a new problem may arise due to the fact that most recognition modules, which come next, require a one-dimensional signal of features, and cannot handle features taken directly from the word image. Segmentation can be bypassed when the features used are global, i.e., they are located in the word resolution and therefore they can be organized in the order they appear in the word from left to right. However, when local features are preferred, one needs to divide the word image into sequential fragments, before the feature extraction stage takes place. In this case the fragmentation process substitutes the full segmentation. Next, a feature extraction process takes place. When high resolution features are used, the extraction process is more sensitive to noise. It is common to use code books in this stage when the feature space is discrete. The recognition process follows next. This process is heavily influenced by the nature of the segmentation process, as will be discussed later on. The recognition process is followed by post-processing. This process relates to lexicon lookup, string correction, and re-evaluation of a word probability with respect to syntax and context issues.

Most of this survey focuses on the algorithms that were proposed in order to realize the recognition phase. The other modules that usually constitute a word recognition system are briefly discussed in Sect. 2.

One can classify the field of offline cursive word recognition into three categories according to the size and nature of the lexicon involved: large; limited, but dynamic; small and specific. Small lexicons do not include more than 100 words, while limited lexicons may go up to 1000. Large lexicons refer to any lexicon size beyond that. When a dynamic lexicon (in contrast with specific or constant) is used, it means that the words that will be relevant during a recognition task are not available

Correspondence to: E. Rivlin

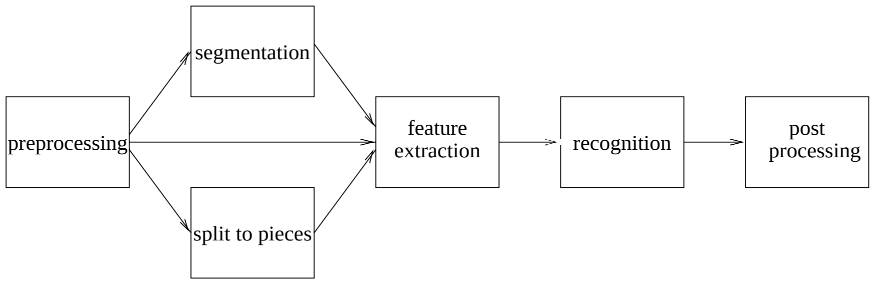

Fig. 1. Three alternative structures of a word recognition system. Each alternative differs in the way it handles the segmentation phase

during training because they belong to an unknown subset of a much larger lexicon. This classification coincides with the different modeling techniques associated with the different recognition methods. When small or limited lexicons are involved a model-discriminant approach is often used. Using this approach, each word is represented by a unique model. Given an observation sequence O_1^T, one goes over all word models and finds the word W_i associated with the model that has maximum a posteriori probability $Pr(W_i|O_1^T)$. Using the Bayes rule

$$\max_i Pr(W_i|O_1^T) = \max \frac{Pr(O_1^T|W_i)Pr(W_i)}{Pr(O_1^T)}$$

$Pr(W_i)$ is usually assumed to have a uniform distribution due to the fact that statistics on the frequency appearance of each lexicon word is unavailable (unless specified otherwise). Since $Pr(O_1^T)$ is also independent of W_i, the a posteriori probability converges to the score given by the word model $Pr(O_1^T|W_i)$.

Practically, a system that is based on a model-discriminant method cannot handle large lexicons. In this case, usually a single model acts as a hypothesis generator that reacts to the input observations and produces a ranked list of candidate words. Note that in this hypothesis (path)-discriminant method, the generator produces spurious words (outside the lexicon), besides the legal ones with their associated matching score. During the process the lexicon is used to verify the existence of complete words or prefixes of hypotheses. However, models in both large or limited lexicon environments have a lot in common. Since the lexicon available during training is large in both cases, there cannot be a reliable training if only some word samples will be used. Therefore, models used in these scenarios are built from letter models and hence might be called *letter-oriented*. This is based on the assumption that a specific letter looks the same when it appears in different locations and neighborhoods (regarding the other letters that surround it). Empirically, this assumption is very solid if one selects valuable features that are invariant in this manner, in spite of the noise that characterizes cursive script due to the ligatures between letters.

Clearly, the larger the lexicon is, the more flexible the application that utilizes it can be, but the recognition becomes more difficult and the results get less satisfactory. Furthermore, all methods that were proposed for larger lexicons are applicable to smaller ones as well but are less suitable, meaning they will probably do worse.

Besides the lexicon size and nature, a major issue that the recognition method should relate to is the segmentation problem. This issue is one of the most important decisions one needs to make when starting to design a word recognition system. Ideally a perfect segmentation algorithm would split a cursive word image into complete characters. Then, using character recognition techniques, the word would be recognized with high confidence. Unfortunately such a segmentation algorithm is not available. Furthermore it may never be available due to ambiguity in cursive words that was best expressed by Sayre's paradox [71]: "To recognize a letter, one must know where it starts and where it ends, to isolate a letter, one must recognize it first". This conflict was reflected in the field of cursive word recognition methods resulting in a situation of having very few methods that rely on pure segmentation followed by a separate recognition phase. Three examples of methods of this kind ([88], [60] and [66]) will be discussed in the relevant section.

Based on this last observation two approaches were developed. One implements a segmentation-free approach, i.e., there is no attempt to split the word image into segments that relate to characters. Still, it is possible that the image would be split into pieces in order to produce a sequence of observations, i.e., symbols. Instead of a letter-by-letter recognition, one tries to recognize the whole word as one entity, by searching for a word with the most similar complete description to the one obtained from the whole input image.

The other approach uses a segmentation-based method. Given a sequence of primitive segments derived from a word image, and a group of possible words, we seek for a top-level segmentation that maximizes the average matching score between respective characters and segments. Top-level segmentation means that each segment is a union of one or more consecutive primitive segments, under the condition that each primitive segment appears exactly one time. In other words a top-level segmentation is a subset of the primitive segmentation points. The term primitive with respect to segments stands for elementary segments that were created by the segmentation algorithm that was used. We find this approach preserves the semantic meaning of symbols with respect to characters, while the segmentation-free approach can prevent errors caused by unsuccessful pre-segmentation.

In both approaches many methods use dynamic-programming tools in order to find the best interpretation. Optimization problems often use dynamic-programming techniques when the optimal solution is a combination of

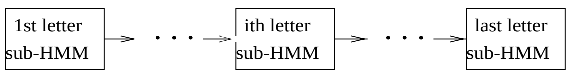

Fig. 2. A single word HMM, built by concatenating the relevant letter sub-HMMs

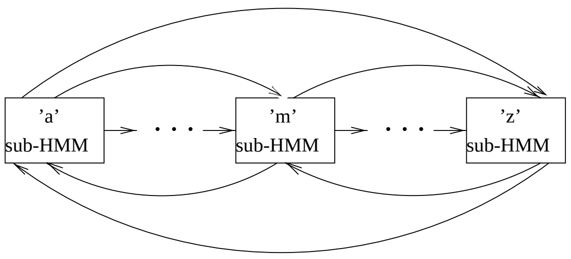

Fig. 3. A single HMM that can handle a large lexicon. Each one of the 26 letters is associated with a sub-HMM that is connected to all the others. The resulting model gains a clique topology in which different paths represent different word interpretations

optimal solutions to partial problems. In a word recognition case the optimal interpretation of a word image may be constructed by concatenating the optimal interpretations to disjoint parts of the word image. One should refer to [3] and [18] for further explanations regarding dynamic programming. A very important application that is also commonly used by methods of both approaches is a hidden Markov Model (HMM). Some find the HMMs to be a particular case of the more general dynamic-programming field.

When small or limited lexicons are used (in segmentation-free or segmentation-based approaches), each word is associated with a separate HMM that is a concatenation of the relevant letter sub-HMMs as illustrated in Fig. 2.

When large lexicons are used, a single HMM in which different paths (state sequences) represent different word interpretations is used. Such HMMs can be described as a graph with a clique topology, where each node represents a letter sub-HMM and the arcs represent transitions between them (Fig. 3). Section A briefly reviews HMMs and some observations to do with their role in word recognition.

In this paper we have decided to use the type of the segmentation scheme as the major criterion for classifying the recognition method used. As many recognition methods can be easily adapted to handle both limited and large lexicons, the size and nature of the lexicon is not used as a major classification criterion. However, the relationship between each approach and the various lexicon size and nature is discussed.

We take perception-oriented methods as forming a different/separate category. These methods differ from both the segmentation-free and segmentation-based oriented methods. The common principle to the methods that belong to this class is first to identify some characters and then continue with trial and error techniques regarding the gaps that were left – using the provided lexicon for help. These methods attempt to recognize individual characters that are present in the word image, and do not try to recognize the word as a whole. The implementation of the recognition phase is not segment-oriented and it does not follow the regular scheme of a left to right search.

For practical reasons there is a common assumption that an input word is not necessarily cursive. It is likely that there will be a mixture of cursive, hand printed, a mixture of lower case and upper case, etc. Therefore it is common to find methods that propose extra modeling for more than one style. This could be achieved by doubling each model and training each copy to handle a different writing style, or by combining redundant submodels parallel to the existing ones in parts of the word that may be written in another style. Since this treatment can be applied to any method, we do not mention specifically those who have decided to use it.

We believe that surveys are important tools that help to synchronize different research efforts around the world and make use of the knowledge and previous experience that was acquired by others. The reader might be interested in other relevant surveys involving online cursive recognition [82], OCRs [42,84], and segmentation techniques [52,8].

2 Stages in a word recognition system

In this section a global structure of a word recognition system is described. The following structure is a union of all common operations that usually appear in systems of this kind. A typical complete word recognition process consists of the following parts: preprocessing, a possible segmentation or fragmentation, feature extraction, recognition, and post-processing. The core of a recognition system is the algorithm that produces word interpretation given a sequence of observations in either one of the various word representation levels that will be mentioned. This is the main issue of this survey and it will be discussed widely in the next sections.

The preprocessing starting point depends on the environment in which the system is running. It may include external word segmentation (extraction) from a multi-word neighborhood and other various document processing techniques. Given a stand-alone word, a few normalization operations are performed, among which are:

- Skew correction – a rotation transformation that brings the word orientation parallel to the horizontal;
- Slant correction – a shear transformation that attempts to make all the vertical strokes erect;
- Smoothing – including all different kinds of noise reduction;
- Scaling – invariance to size (used in rare cases only).

Another important procedure in a major part of the systems is reference line finding. This procedure is essential for the feature extraction stage that comes next. Optional contour or skeleton tracing are also parts of the preprocessing phase.

In Fig. 4 one can observe some of the various preprocessing algorithms that are commonly used. First, the

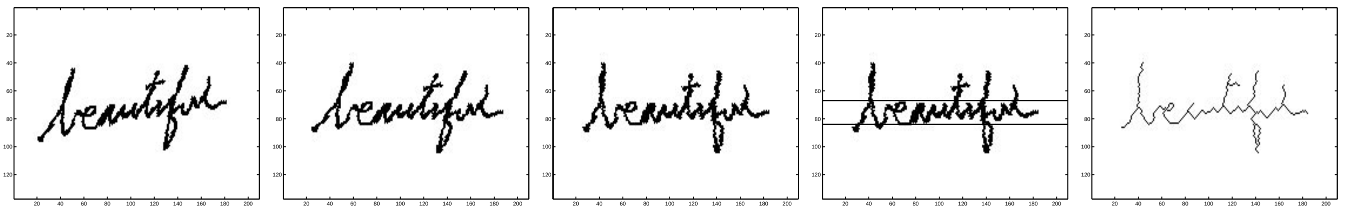

Fig. 4. Some of the various preprocessing algorithms that are commonly used. From left to right: the original image, then the image after going through skew correction, slant correction, reference line finding, and eventually a thinning algorithm that produces the word skeleton

original image is presented on the left, then the image after going through skew correction, slant correction, reference line finding, and eventually a thinning algorithm that produces the word skeleton. The algorithms were executed in a raw, i.e., each algorithm was input the image that resulted from its preceding algorithm.

Pre-segmentation is the process of word segmentation into primitive segments. Many algorithms have been proposed for this task and we recommend two surveys that review this field [52] and [8]. In the case of a segmentation-free method, this stage may be ignored or replaced with splitting the image into sequential fragments that do not attempt to match complete characters.

The role of the feature extraction stage is to retrieve observations out of the word image. There are several classes of features. Segmentation-free methods use either raw features, which are pixel-wise like strokes, or global symbolic features such as ascenders, descenders, loops, etc. Segmentation-based methods look for local features in the segment that is being evaluated. The most popular features in this case are the global features (ascenders, descenders, loops) and local irregularities such as X and T crossings, end points, sharp curvatures, etc. Other, less symbolic alternatives are different kinds of pixel moments, distribution in the different parts of the segment, etc. A feature is assumed to be more reliable if it is less sensitive to noise and to the variance in writing style, and if it has good ability to distinguish among words or letters as required.

After the recognition module has finished running, some methods use post-processing techniques to improve the recognition results. Some of these operations such as lexicon lookup and string correction are mentioned in this paper because they are built in during the recognition task. However additional syntax and context are very popular to bust or disqualify legal words according to the circumstances. A special case appears when large lexicons are involved. In this case a hypothesis generator outputs possible words that might not be present in the lexicon. Therefore, one may find it very difficult to search for an optimal solution under the constraint of being legal, i.e., to appear in the given large lexicon. A unique post-processing was developed for this purpose, based on the minimum edit-distance that will be mentioned later on in other aspects of recognition. Given a hypothesis that is the optimal interpretation of the input word image, we find the lexicon word that is the most similar to the hypothesis. Similarity between words is measured as the number of operations – insertion, deletion, substitution – necessary to transform one word into the other. From a different point of view this process is considered as correction of garbled words [62]. Note that regular post-processing of this kind requires going over all the words in the lexicon and performing non-trivial calculation for each one of them. This puts limitations on the size of the lexicon that is assumed to be large. Therefore, a proper usage of this process is in combination with some kind of lexicon reduction.

The interested reader may refer to [80] for more details on a complete handwritten text recognition system.

3 Segmentation-free methods

In a segmentation-free method, one should find the best interpretation possible for an observation sequence derived from the word image without performing a meaningful segmentation first. An observation sequence can be classified into three categories according to the representation level of the word it stands for. The first category relates to observations that are based on low-level features taken directly from the word image. Such features include smoothed traces (quantized/normalized fragments) of the word contour, pieces of strokes between anchor points, edges of a polygonal approximation of the image skeleton, etc. The second category aggregates such low-level features to serve as primitives. For example, neighboring strokes can be merged into a smoothed pattern, that will constitute a primitive. The main difference between the current category and the former one is in the nature of the relevant feature space – continuous in contrast with discrete. The last category involves methods that use even higher-level features of a word image. The most popular features are the most irregular, i.e., holistic features that are hard to miss and are invariant with respect to all the different writing styles. In this case holistic means global with respect to a whole word resolution, meaning that features of this kind, such as ascenders, descenders, loops, *i* dots, *t* strokes, etc., are prominent even in an image of a complete word. These features may be sub-classified according to size, location or orientation. These features will be referred to as symbols.

In this section we discuss different algorithms that were proposed for comparison between a pair of observation sequences. Most methods use either a minimum edit-distance calculation based on dynamic-programming, or a resemblance estimation provided by HMMs.

However, there are some specialized methods that include different comparison procedures. Nevertheless, in most cases, such systems can utilize the minimum edit-distance as well. The top-level organization of this section matches the various comparison approaches (specialized, minimum edit-distance, HMMs). Each subsection is further sorted according to the feature level of the word image representation that was used. The discussion in this section will be in view of the size and nature of the lexicon that the application may use. Table 1 shows the location of each method that will be mentioned later with respect to the comparison technique and representation level.

3.1 Specialized methods

In what follows we will discuss methods in which an observation sequence is mapped to a space with a distance metric on complete sequences. All these methods are of model-discriminant nature. Since finding the most likely word is similar to other categories of model-discriminant – i.e., comparison between the input sequence and all stored references, we focus on the unique details of each of the methods.

1. *Low level*
   Govindaraju et al., [36], represented a word image as a sequence of strokes in the temporal dimension. Temporal information, i.e., a complete linear order among the strokes, is extracted by traversing the strokes between consequent anchor points such as peaks, valleys, intersection and end points. When an intersection point is encountered, the traversal proceeds along the smoothest path, determined heuristically based on orientation, trend and a gradient smoothness criterion. Finally, a feature vector associated with the word image is obtained. No details were specified regarding the matching algorithm. However, the dynamic-programming or edit-distance seem to be a possible good choice in this scenario.
   Gorsky [35] have used a *holographic representation*, meaning that each sequence of consequent stroke quanta that share approximately the same direction (which will be referred to as fragments) is mapped into a single point in a special parameter space. The following features constitute the coordinates of a point: *global x-order* (number of fragments along the x-axis), *local order* of a fragment among the neighboring fragments, and *direction*. These features are insensitive to most of the distortions produced by different writing styles and may also overcome missing or additional stroke components. The intensity of a point depends on the length of the fragment it originated from and bonuses for distinguishing properties. Both indices and values were discrete and normalized, therefore the resulting *holograph* is a three-dimensional matrix. A certain word *prototype*, i.e., model, is created by mapping a set of words written by various people to a single *holograph*, i.e., matrix. Comparison between an input word and a prototype both represented in holographic form could be carried out either by cross-correlation or by computing the percentage of word fragments that are "explained" by the prototype.
2. *Medium level*
   In [63] a Markov model with no hidden states was used to model a word. The set of legal observations derived from a word image is used to define the states of the model. Experiments with two sets of observations – 8 strokes or 42 graphemes – were made. Choosing the order of the Markov model was based on statistical criteria, and was eventually found to be 2. Therefore, each word model was trained and assigned transition probabilities considering all the possible triple combinations of observations. The a priori probability of a word was also taken into account. Hence, the probability of a word Markov model M_i given the observed sequence Q is evaluated as follows:

$$\begin{aligned}&P(M_i|Q)\\&\quad=\frac{P(Q|M_i)P(M_i)}{P(Q)}\\&\quad=\frac{P(M_i)P(X_1|M_i)P(X_2|X_1,M_i)\ldots}{P(X_1)P(X_2|X_1)P(X_3|X_1,X_2)\ldots}\\&\qquad\times\frac{\ldots P(X_3|X_1,X_2,M_i)P(X_n|X_{n-2},X_{n-1},M_i)}{\ldots P(X_n|X_{n-2},X_{n-1})}\end{aligned}$$

3. *High level*
   Guillevic et al., [38] have extracted seven types of features from an input word image. Each feature is associated with its relative position in the image (in percents). In the training process, each lexicon word goes through the same process but the feature locations are relative to the characters they belong to. Thus, in the recognition process, for every lexicon word, we translate positions of features in the input image, that were given in percents, to character locations of the word being compared and rate the matching. The distance between an input feature vector and a lexicon word feature vector is computed as the minimum shift in feature locations needed in order to match between the two.
   A mixture of segmentation-free principles and a segmentation algorithm takes place in the methods presented by Madhvanath et al. [53–56]. The produced segments are used only to define the location of a feature inside a word image. In the first case ([53]), a concatenation of all features found results in a feature vector used for comparison with references – using Euclidean distance. A few supporting mechanisms were supplemented; feature equivalence rules enabled one to define interchangeable sets of features. *Macro-features*, i.e., a combination of features, lead to the recognition of a specific phrase with very high confidence. In addition a new feature – *point of return* – is included. However, this feature seems less robust and more author-dependent. On the other hand if a small lexicon of the order of ten words was engaged, the ascender and descender features suffice to distinguish lexicon entries. In a more sophisticated method, all features are sorted according to their type and sub-sorted by their position from left to right [54–

Table 1. A summary of segmentation-free methods

Representation level	Specialized	minimum edit-distance	HMMs
Low	[36,35]	[65,23,55,57]	[39,70,5–7,29,59,17]
Medium	[63]	[64]	
High	[38,53–56]	[50,60,66,21]	[32–34,30,31]

56]. When comparing the observations derived from a word image with a similar reference of a lexicon word, one attempts to find the best bipartite matching between the two, using a dynamic-programming algorithm. Clearly, only features of the same type can be matched. In addition, features' order should be kept, i.e., edges of the resulting graph may not cross. Moreover, there is another logical constraint on the matching, that is, a pair of features cannot be matched if the difference between their positions exceeds a threshold. A possible bipartite match does not have to be complete since some features might have been missed and others may be spurious. Each valid match is given a score depending on the positional difference between matched features but also on the weight each feature type is associated with. In addition, a match confidence may also be measured by a similar interpolation over the confidence associated with each matched feature. A feature confidence is given during its extraction from the image, representing the amount of certainty that a feature of this type really exists in that position. In [56], letter models also specify optional features in order to model alternative ways of writing the same letter. These features, though given lower confidence, are very useful in case of spurious or missing strokes. This method utilizes common features like ascenders, descenders etc., together with the word length in segments. In addition, some a priori valuable facts, like the existence of an ascender (descender) at the beginning (end) of a word, are handled as independent features.

3.2 Minimum edit-distance

Several methods rely on minimum edit-distance (a simple dynamic-programming approach which is described in Sect. B in the appendix) for comparison in the different levels of a word representation.

1. *Low level*
   Since low-level features are usually measurable, any matching between a pair of them may be considered as a substitution with a penalty relative to the mutual distance in the feature space.
   – Parisse [65] used the upper and lower profiles of the word image. A profile in this case means a series of vectors describing the respective contour.
   – Eliaz et al., [23], used polygonal approximations of word images for comparison. The word's skeleton is divided into fragments and each one of them is compared to all fragments of a lexicon word model that are relatively close when sharing the same coordinates (all words were first normalized to enable the computation of relative distance). The similarity between a pair of fragments of the input word image and a pre-stored model respectively take into account the difference between the properties of themselves. In addition, the similarity between their neighborhoods, i.e., the minimal edit-
   distance between the two sets of fragments that are close to the compared fragments in the image and in the model respectively, are also part of the calculation.
   – In Madhvanath et al., [55,57], given an input cursive word, all down-strokes were extracted. Medium-level features resulting from classification of these strokes into five discrete types – ascender, descender, f-stroke, medium, and unknown – was found unsatisfactory in [55]. Therefore, the extracted strokes were associated with their normalized position above (below) the half-line (baseline) in [57]. However, the descriptions of the lexicon words were still given in a discrete form. The minimum edit-distance algorithm was performed simultaneously for all the lexicon words that were ordered in a trie.

2. *Medium level*
   In Pacquet et al., [64], a set of seven possible primitives (occlusions, different upper and lower strokes, or connexions) resulting from a guiding point were defined. The respective guiding points were intersection points between strokes of the word and the intermediate base-line – the line that divides the body zone between the base-line and the half-line into two strips of equal height. The weights of the three editing operations (deletion, substitution, insertion) were fixed in advance. However, in some exceptional cases, combinations of primitives called shapes could be replaced by "similar shapes" without increasing the total edit-distance. Eventually the distance calculated for each reference was normalized by the length of the observation sequences compared.

3. *High level*
   – In Leroux et al., [50], ascenders and descenders derived from the primary contour were supported by closed loops, *i* dots and disconnected *t* crosses evidenced from secondary contour components. According to the experiments performed, recognition rate was not improved when the edit costs were learned. However, the amount of lexicon examination was successfully limited based on a rough estimation of the word length.

– Moreau et al., [60] used an extended reference codebook of the lexicon including many orthographic deviations, that usually appear due to poor semantic meaning of the features involved. Normalization of the edit cost was based upon a rough estimation of the word length in letters.
– In Plessis et al., [66], an edit-distance between symbolic description chains was one of the three methods that were combined before final recognition results were given. The authors decided to use two separate encoding systems of each word – depending on whether loops were taken into consideration in the global feature list or were ignored. The reason behind this distinction is the huge variance in loop occurrence among different writers.
– A higher level of implementation appears in a paper dedicated to optimization of the minimal edit-distance parameters [21], where an application to handwritten word recognition was presented. Given connected components derived from a word image (in a fully connected word that would be the whole image), the number of characters L present in the component is estimated using the width/ height ratio and the number of crossings with the center line. Then L special characters are inserted in the recognized character string. The special characters reflect the presence/absence of an upper stroke, lower stroke, or inner loop, in the relevant part of the signal. The optimal lexicon word was found using an improved edit-distance between the string of special characters (including real letters, if they were naturally segmented from the rest of the word) and each possible word. The paper presents a method for training and optimizing the character substitution parameters, and applies some additional rules (e.g., the substitution of one character with two and vice versa). This method is considered a higher-level method as the comparison is performed between a sequence of pseudo-characters and the real letters of the word being estimated. It seems that this method does not perform well on fully connected words.

The above segmentation-free methods that calculate minimum edit-distance between two sequences of observations or perform a specialized comparison technique, appeared as holistic methods and were used only for a small static lexicon environment. In that case a word reference is an interpolation of all training samples available for the specific word. However, these methods might be used in limited dynamic lexicons. This can be achieved by creating an artificial observation sequence for each lexicon word by concatenating the observations associated with each of its constituting letters. This requires training samples in the letter level. The performance of methods of this kind depends on the representation level that the observations relate to. Thus methods that belong to the first category of the low-level observations, contain a lot of information that belongs to ligatures and therefore may not be suitable for letter-oriented. On the other hand, the features that are considered in high-level observations appear mostly in the letter area. For the persistent user even the former observation class may be adjusted to limited dynamic lexicons by using training samples of letters derived from cursive words without eliminating the ligature fragments that are attached to it. Only [66] [21] [53] have followed the proposed scheme and enabled the usage of dynamic (limited) lexicons.

3.3 HMMs

HMMs of segmentation-free methods share a common structure and the same training scheme. The basis for the following methods is a statistical estimation of the edit-distance that is more sophisticated than the one presented above. This is achieved by an HMM that gives higher probability to observation sequences that are "closer" to the training samples. "Close" in this case relates to an edit-distance metric. The common method is a combination of a specific structure of an HMM and an appropriate training algorithm. The structure of a typical HMM consists of a sequential backbone path and several branches that represent local alternatives. A null transition from one state to the following, or a transition that skips the next state, represent a missing symbol. The probabilities of the different symbols associated with a backbone transition[1] are relative to the penalty of a symbol substitution. One or several symbols that are expected to occur at this point should not be given any cost. A self-loop or an additional state (introducing a double step transition between consecutive states) stand for a symbol insertion. The training algorithm that is adapted by all methods – the Baum-Welch (forward-backward) algorithm [2,1] – maximizes the probability of the training samples that were introduced to a specific model.

HMMs of this kind were used in all three possible cases of a lexicon size and nature. Most of the systems were designed for limited and large lexicons. Hence, a major part of the HMMs that were used are letter-oriented. Moreover, we found no difference between letter sub-HMMs that serve dynamic limited lexicons and those that serve large lexicons. Therefore, we would like to point out that despite the fact that each method was tested only in a single context (dynamic limited or large lexicon) they can all be used in either one of them if placed in the suitable topology, i.e., a clique or sequential concatenation. The adaptation to small and static lexicons is also very intuitive. In order to construct a single chain of sub-HMMs, one should ignore the semantic meaning of letters and utilize the same structure that letter sub-HMMs were given. The length of the chain should be chosen so as to represent best the training samples.

The first examples of HMM-based segmentation-free methods that we include relate to systems that perform

[1] HMMs that realize segmentation-free methods do not attach any meaning to specific transitions, with respect to character fractions.

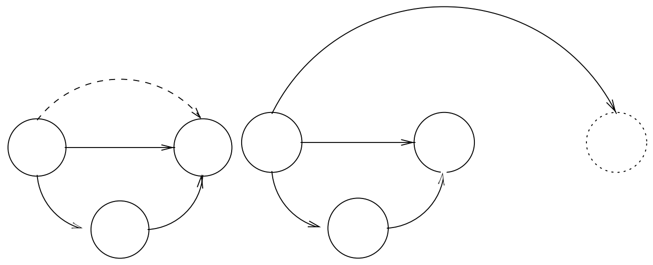

Fig. 5. The two alternatives of an elementary sub-HMM that were used by Gilloux et al. The dashed line represents a null transition

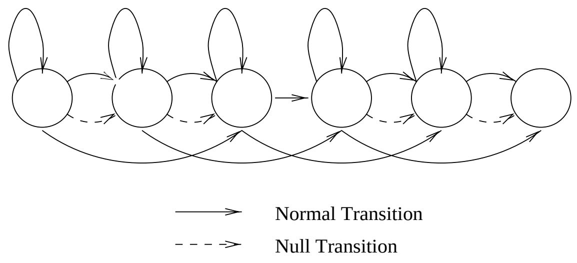

Fig. 6. The letter sub-HMM that was used by Cho et al. Other letter sub-HMMs used in segmentation-free methods are subsets of this model in different lengths

recognition in a small specific lexicon environment. All methods by Guillevic et al., [39], Saon et al., [70], and Gilloux et al., [32–34], represent a word model as a chain of n identical sub-HMMs (see an example in Fig. 5), where n is the most probable length of an observation sequence obtained from the training samples. In Saon et al., each sub-HMM includes a single state and 2 possible transitions: normal left to right and self-loop. Guillevic et al., added the possibility of skipping over the next state. Both methods use low-level features derived directly from the word image. In Guillevic et al., a sliding window was used to split the image into vertical zones, that were then divided into 5 horizontal regions. The distributions of the 8 possible slopes that might be found in a contour, were used to produce a low-level feature vector. In Saon et al., a fixed-size column of the word image is used as a feature vector input to the HMM at each stage. In order to compute the conditional pixel observation probability at each state, the developers considered the image as a non-symmetric half-plane Markov chain. In this case one should take into account only a subset of the image pixels that support a pixel in a given column.

In Gilloux et al., the sub-HMM contains 2 states: initial and additional, and 4 transitions that result in 3 possible paths starting in the initial state:

1. A null transition from the initial state to that of the next sub-HMM [32]. This one was replaced by a normal transition that bypasses the following sub-HMM in the chain in [33,34].
2. A normal transition from the initial state to the one of the next sub-HMM.
3. A transition from the initial state to the additional state and then to the initial state of the following sub-HMM.

The fact that a fixed lexicon rather than a limited but dynamically built one is used, was utilized to enable a more reliable a posteriori probability computation by estimating the a priori probability *p(w)* of each word. The final recognition score of a word model is therefore $p(o|w) \times p(w)$, found by the Viterbi algorithm [85,26]. In other systems by Gilloux et al., [30,31,33, 34] the same elementary sub-HMMs were used to construct letter models. All the above-mentioned methods by Gilloux et al., used a segmentation algorithm to produce observations that were symbols describing each segment. Therefore, they belong to the methods that are based on high-level features of a word representation, but could have been classified as segmentation-based in the first place. However, we have decided to include them in this section because the same HMMs could be used in a segmentation-free method as well. Furthermore, the structure and training technique of these methods is very similar to other segmentation-free methods. More details regarding the letter-oriented system will be given in the segmentation-based section.

In the following is a short description of a few other HMM-based, segmentation-free methods that were published. Most of these methods use low-level features as observations. Sometimes medium-level features are preferred, for example, the first method described next.

In Bunke et al., [5,6], observations are the edges of the skeleton graph of the word image. The number of states in a letter model is the minimum number of edges (= observations) among all training samples. Only self-loops and left-to-right transitions between consequent states are allowed.

In Gillies [29] and Mohamed et al., [59], observations are the columns of the word image, one pixel or constant pixel wide. All letter sub-HMMs contain the same number of states. For example, in [59] each letter model has exactly 12 states. During training, each sample of a letter is re-sampled to a fixed size of 24 columns, associating observations $2i$-1 and $2i$ with state i. The standard re-estimation formulae are used for further training and probability estimation. In addition to the transitions allowed by Gillies and Bunke et al., Mohamed and Gader also allowed skipping of one state, i.e., one may jump from state i to state i+2.

In Caesar et al., [7] all letter sub-HMMs consist of four states. In addition to linear transitions (between following states), self-loops and skipping one state also exist. When applied in a large lexicon environment, the HMM is evaluated by a modified Viterbi algorithm – for the first letters of a word a breadth first algorithm is used which is then dynamically switched to a best-first search.

In Cho et al., [17], non-trivial sub-HMMs are associated with each letter allowing self-loops, null transitions between some of the consequent states, as well as skipping over the next state (see an example in Fig. 6). The length of each sub-HMM, i.e., the number of states, is determined empirically by the training samples. Besides letter sub-HMMs, ligature sub-HMMs were added to model the inter-character strokes. Therefore, letter models need not take care of variations caused by different kinds of ligatures which are left to their specific models to handle. However, ligature models accept null ligatures as legal ones. The resulting word model can be formally described as:

$$Word := Character \cdot \{Ligature' \cdot Character\}^*$$

$$Ligature' := Ligature|\varepsilon$$

When large lexicons are involved, one needs to preserve the clique topology of the global HMM as far as letters together with the word model described above are concerned. Therefore, a circularly interconnected network of letter and ligature sub-HMMs was built. Each letter sub-HMM's last state was connected with each ligature sub-HMM's first state and vice versa. Linguistic probabilities can be combined with the recognition process as the transition probabilities between letters. Illegal words were avoided by pruning prefixes that had no "future". The combination of this technique with the dynamic-programming nature of the Viterbi algorithm may result in a non-optimal solution. There was not any absolute solution to this problem but some suggestions to increase the probability to get close to the optimal solution such as running a Viterbi algorithm backwards or as a combination of forward and backward searches were described.

4 Segmentation-based methods

In segmentation-based methods the recognition process is based on an attempt to find the best complete bipartite match between blocks of primitive segments and a word's letters. These primitive segments are created by some segmentation algorithm, which might be imperfect, and therefore cause over or under-segmentation. We refer to this as a pre-segmentation stage.

The matching process of blocks of primitives and letters is subject to various constraints. For example, a complete bipartite match means an isomorphic transformation from the set of primitive segment blocks to the word's letters. Obviously, the spatial order of the primitive segment blocks from left to right should be kept within the matched letters as well. Different blocks of primitive segments should be disjoint and provide a complete cover of all the primitive segments of the word. The size of each block, i.e., the number of primitive segments it contains, must satisfy a pre-segmentation criteria that determines the number of primitive segments a character may be split into and the number of characters that may share a primitive segment. Furthermore, some methods refer to all possible values of primitive segment number that a character may be split into, and initialize *durational statistics*. For each character, the durational statistics determine the probability that it would be segmented into d primitive segments by the current segmentation algorithm, for every legal value of d. These statistics are first computed during training, and then taken into account when a matching score is calculated.

In small static lexicons, there is no point in using a segmentation-based method since words can be recognized more easily in the word level using distinguishing holistic features that are very likely to appear in a small set of words[2]. In this section we refer to the segmentation-based field as if it relates to limited (but dynamic) and large lexicons only.

The major advantage of all segmentation-based methods is their flexibility with respect to the size and nature of the lexicon. This is a direct result of these methods being letter-oriented, i.e., they rely upon letter models. Therefore, it is possible to extend the model-discriminant approach, where the matching is done for a single specific word at a time, to a path-discriminant approach, where the current letter of a single word being matched could be each one of the 26 letters.

The best match can be found by a straightforward dynamic-programming algorithm (see Sect. 4.1), or by other equivalent alternatives. One alternative is to find the shortest path in a special graph in which each node represents a pair of possible blocks of primitive segments and letter interpretations, associated with their matching score as a weight (see Sect. 4.2). Another alternative is utilizing special HMMs in which the pair of segmentation and interpretation is associated with the underlying states of the optimal path (see Sect. 4.3). Both methods use the Viterbi algorithm [85,26] to compute the best path required, hence the similarity to the straightforward dynamic-
programming approach. Usually, one attempts to match one or more characters with a block of primitive segments. However, some of the HMM-based methods implement a different concept of estimating the most likely character that might have generated the feature extracted from the current primitive segment. This implies that the borders of a character are not determined explicitly before the evaluation phase. Moreover, estimating the probability of a feature given a possible character, cannot utilize the same common OCR-like evaluators that participate in methods of the former category. The last category involves methods that rely on their segmentation algorithm, and therefore match each single primitive segment with a complete character. However, as all available segmentation algorithms are imperfect, each of these methods use additional processing to overcome segmentation faults.

The following three sections are dedicated to the abovementioned alternatives respectively. In each section we present both the limited lexicon and the large lexicon versions of each approach. Examples will be discussed in the relevant context. The last section presents some specific methods that handle large lexicons from a different viewpoint. Table 2 summarizes all segmentation-based methods according to the different categories they were

[2] There was only one piece of evidence we ran into during our survey of word recognition systems where a segmentation algorithm was used in a small static lexicon environment. Moreover, we found this single method by Gilloux et al., [32–34], to have a lot in common with segmentation-free methods despite the segmentation algorithm that it was integrated with. Further discussion of this exception was given in the previous section.

Table 2. A summary of all segmentation-based methods according to the different categories they were classified into

Dynamic-programming	Shortest path	HMMs	Specialized methods
[27,28,43–46,49]	[61,24,25]	[30–34,86,87]	[4,78,79,88,60,66]
[47,48,75,58]		[10–13,15,14]	
		[40,41,9,72–74]	

classified into. Table 3 presents all segmentation-based methods according to categories and with respect to the lexicon that was originally used in the experiments reported.

4.1 Dynamic programming

Section C briefly reviews the general concepts of dynamic programming. Numerous publications followed the straightforward dynamic-programming approach. Below we describe a few of them, emphasizing several variations on the above approach. None of the methods mentioned in this subsection allowed for a primitive segment to correspond to more than one character. All systems but two in this section used a dynamic limited lexicon, hence the dynamic programming was executed for each word separately and the word associated with the overall best matching score was selected.

In [27,28], a heuristic function *Legal*(segment(s)) disqualifies some unions of primitive segments from being evaluated if they are too complicated to represent a single character. In addition, a more complicated dynamic-programming algorithm that takes into account the compatibility between two possible consecutive characters was also proposed.

In [43–46] the proposed evaluation function returns the minimum distance between the block of primitive segments given and the training samples of the requested character. Therefore, dynamic programming seeks a global minimum instead of a maximum. When the matching between a character and a sequence of primitive segments is performed only for sequences that are within a permissible window, unnecessary computations are eliminated and the alignment becomes more meaningful. The size of the window depends on the durational statistics of the character that it is being compared with (or the segmentation criteria if the former is not available), and other factors such as the number of characters in the lexicon word and number of primitive segments in the test image. In [44–46] a preliminary test to disqualify impossible lexicon words in advance was used:

$$max\_seg = \sum_{j=1}^{N_c(i)} dur(lex\_entry[j]),$$

$$if\ (num\_seg\_of\_word > max\_seg)\ reject$$

where *lex_entry[j]* represents the j-th character in the current lexicon word. The upper bound of the segmentation criteria, i.e., the maximum number of primitive segments a character may be split into, can be used instead of the durational statistics. Another preliminary test was also proposed to disqualify lexicon words that have more letters than the number of primitive segments given. The potential of character durational statistics to predict inadmissible segmentations was also utilized during the dynamic-programming execution in [45]. Each matching attempt between a character and a block of d primitive segments also took into account the probability of that character to last d primitive segments. The later incorporation of the durational statistics with the evaluation function seems to improve the recognition results.

The same method was used by Kimura et al. [49, 47,48,75]. In [47], a cost of splitting was integrated to the objective function (*value*) to suppress unnatural segmentations due to excessive split and merge. The total splitting cost is defined by

$$Splitting\_cost = \sum_{i=1}^{n-1} \frac{R_i}{R}$$

where n is the number of characters, R_i is the total length of splitting runs which are used to separate i_th and $i+1$_th characters, and R is the summation of all splitting runs generated in the pre-segmentation process. The resulting objective function is combined, here as a penalty, in the word likelihood calculation.

It is our view that it is possible to expand the model-based dynamic-programming technique for working with large lexicons. However, we found only two methods of this kind in the literature. Shridhar et al., [75] and Man et al., [58], did not constrain the k_th character to be a specific one but chose the most likely in the current block of primitive segments. In this case, the evaluation function (*match*) may be replaced by an ordinary OCR module that outputs the most likely character and its confidence given a partition of the word image. The same array and dynamic-programming algorithm can be used in this case as well, if the word length is estimated in advance. The only difference was the semantic meaning of the k_th character that is a variable and not constant. In order to overcome a mis-estimation of the word length, Man et al., executed the algorithm for several other close word lengths. The comparison between two interpretations of different lengths was based on an entropy calculation:

$$p_i = q_i/(q_1 + q_2 + \ldots + q_n)$$

$$H_n(p_1, p_2, \ldots, p_n) = -\Sigma_{i=1}^{n} p_i \ln(p_i)$$

Table 3. A summary of all segmentation-based methods according to categories and with respect to the lexicon that was originally used in the experiments reported

Lexicon type	Dynamic-programming	Shortest path	HMMs	Specialized methods
Small static			[32–34]	
Limited dynamic	[27,28,43–46] [49,47,48,75]	[61,25]	[30,31,33,34] [86,87] [9,72–74]	[60,66]
Large	[58,81]	[61,24]	[13,15,14,40] [41,10–12]	[4,78,79,88]

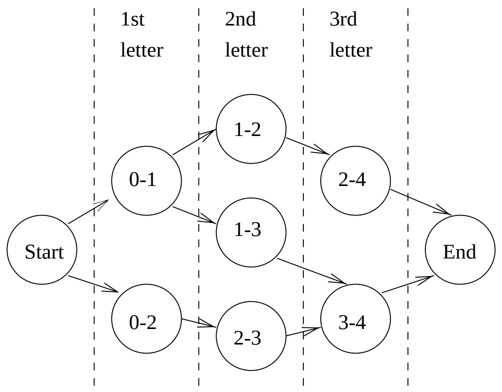

Fig. 7. The corresponding model-discriminant graph for a three-letter word subject to a maximum of two segments per character. The numbers correspond to the indices of respective segmentation points

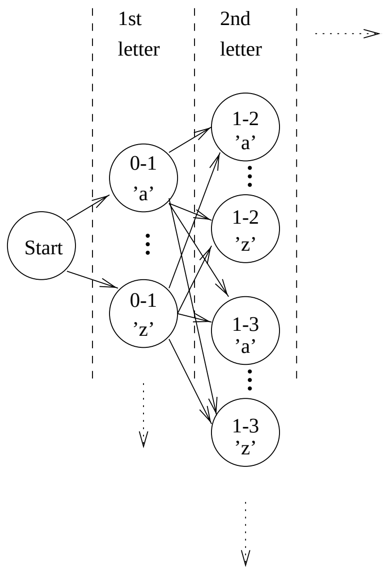

Fig. 8. The corresponding path-discriminant graph for a three-letter word subject to a maximum of two segments per character. The numbers correspond to the indices of respective segmentation points

where q_i represents the degree of membership of each hypothesized character, i.e., its confidence.

The authors are currently developing some new algorithms that would allow extending the dynamic-programming methods to large lexicons – see [81].

4.2 Shortest path variation

The simple dynamic-programming algorithm can be mapped to a compatible graph in which finding the shortest path from left to right using the Viterbi algorithm [85,26] is isomorphic to running the general dynamic-programming technique directly on the group of primitive segments. Such an approach was taken by Nohl et al., [61]. In the limited lexicon version, i.e., when computing the matching score of a single word, each interior node in the graph corresponds to a possible image segment *L-R* and a particular character C within the word, numbered *(L-R,C)*, where L and R are preliminary segmentation points that satisfy the segmentation criteria as a couple. Each (directed) arc in the graph corresponds to an allowed sequence of two image segments and their possible interpretation, usually generated by the following rule

$$(L-R,C) \rightarrow (L'-R',C') \quad \text{where } L'=R \quad \text{and } C' \text{ follows } C \text{ in the respective word}$$

Two additional nodes appear in the graph: the "Start" node, which is the origin of arcs terminating on all nodes corresponding to the first character of an interpretation, and the "End" node, which is the termination of arcs from all nodes corresponding to the last character of an interpretation. An example of a three-letter word using a segmentation algorithm that does not split a character into more than two segments is shown in Fig. 7. Every legal segmentation of the word image has a corresponding path in the graph (from "Start" to "End") and vice versa. We associate with each node *(L-R,C)* a score P representing the probability that segment *L-R* should be interpreted as character *C*. Thus, the score for any particular segmentation is the product of the scores $\{P_i\}$ associated with the nodes along the corresponding path. The best segmentation, i.e., optimal path, is computed using the Viterbi algorithm [85,26] to maximize $\prod_i \{P_i\}$.

The limited lexicon oriented method can be adapted for a large lexicon environment, by switching to a path-discriminant method. In this case instead of using a separate graph for each lexicon word, a single graph is used where each path is associated with a possible word interpretation. The method relies on a similar segmentation-recognition graph, in which each legal block of primitive segments that can be interpreted as a character is represented by a state for each one of the possible characters, i.e., usually 26 states. In this case, the term "legal block" means that the block satisfies the segmentation criteria used. The arcs of this graph are between all pairs of states that represent consecutive blocks, i.e., two blocks of primitive segments that share a common segmentation point – one on its left and the other on its right. Each state is associated with a recognition score proportional to the matching score between the current block and the plausible character. The resulting graph can be viewed as the one used in the model-discriminant approach for a specific word where each state was duplicated for each possible character. Some arcs that were eliminated from the model-discriminant graph may be returned. Elimination of arcs may occur when they give rise to paths of length that differ from the constant path length. A constant long path is required only in a graph that represents segmentation-recognition alternatives of a single word. Fig. 8 presents a part of the single word graph shown in Fig. 7 after the transformation from single to multi-word handling. Each path through the graph corresponds to an allowed segmentation alternative and a possible interpretation of the word image. The best scoring path is the one that represents the most likely word hypothesis.

Another graphical modeling of dynamic programming was presented by Favata [24]. In the first stage, all possible letter interpretations are determined by performing basic character recognition for each possible block of primitive segments and calculating a confidence for each of the 26 letters. Each such block contains at most N ($N = 4$) consecutive primitive segments. The system keeps track of the results for each such block including character confidences and left and right segmentation points. In the second stage, Favata considers all possible word interpretations by generating all valid paths through segmentation points. This is done by creating a weighted directed graph $G = (V, A)$, where V is the set of segmentation points and each arc between two segmentation points LP and RP corresponds to a letter interpretation of a block between LP and RP. The weight on the arc is the confidence of the letter. Clearly, there might be up to 26 edges between a pair of nodes differing by their weight and letter labels. An approximate matching using beam search enables the integration of various heuristics with the dynamic-programming algorithm for finding the shortest path.

We find this graphical modeling a perfect solution to dealing with complete cursive words in contrast to short sequences of connected or touching characters. Therefore, this may seem like an upgrade to the method presented by Favata et al., [25], where each plausible segmentation-interpretation alternative was investigated and ranked. However, we believe that combining the graphical modeling from [24] with the additional properties used in [25] (like digram and trigram statistics gathered from the lexicon) would make Favata's results even better. Employing certain spatial rules that reflect the compatibility between letters and their surrounding letters such as relative height would also be very nice to have.

4.3 HMMs

In the context of HMMs we call *segmentation-based algorithms* those algorithms that use significant segmentation, and each transition is associated with an observation symbol that has a semantic meaning of a certain fraction of the respective character. The resulting letter sub-HMMs do not have self-loop transitions. In general the number of possible paths through a letter model is the minimum required in order to preserve all different segmentation/symbol assignments of a letter. Since there is no difference between letter sub-HMMs that serve dynamic limited lexicons and those that serve large lexicons, we did not classify the methods with respect to the lexicon that was used in the experiments reported. However, we included special comments regarding the lexicon used in cases where the method required some lexicon sensitive adaptations.

Gilloux et al., [30, 31, 33, 34] use the same elementary sub-HMMs (see an example in Fig. 5) that were used in their holistic method [32–34] defining them here as letter models. When used as letter models in conjunction with segmentation, the three possible paths from the initial state of the current letter to the corresponding state of the next letter become more meaningful. A path that includes a null transition between two consecutive initial states of the current and following letters represents under-segmentation. A path that includes a normal transition between the same states represents correct segmentation with the common symbol or a different one. A path that includes a transition from the initial state to the additional state in the same letter, and then to the initial state of the following one, represents an over-segmentation scenario.

In order for their method to succeed, the associated segmentation algorithm should not split a letter into more than two primitive segments. Therefore, special letters like *w*, *m*, and *n*, that tend not to satisfy this criterion are represented as a concatenation of more than one sub-HMM. Training was done according to the forward-backward (Baum-Welch) algorithm [2, 1], the same algorithm that was used to train all HMM-based segmentation-free methods.

An upgrade to the previous system is presented in El Yacoubi et al. [86, 87]. Each transition (besides a null one) turns into a double transition with a state between the two. This supplement was used in order to model the segmentation point (sp) type, as well as inter-letter spaces.

Several HMM methods for segmentation-based word recognition systems were developed by the CEDAR research group at SUNY [13, 15, 14, 40, 41, 10–12]. Their initial work developed robust letter sub-HMMs in order to handle both over and under-segmentation cases [13, 15, 14]. According to the segmentation algorithm each letter may be decomposed into three parts at the most:

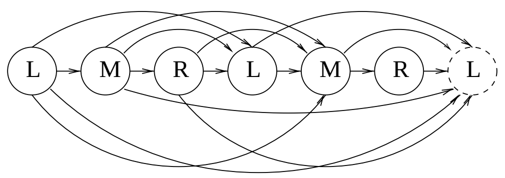

Fig. 9. Combining two-letter sub-HMMs for handling both over and under-segmentation (Chen et al.)

left, middle and right. On the other hand, two letters may be joined together to appear in a single primitive segment. The resulting HMM is built of three state letter sub-HMMs, representing all possible locations of a segmentation point. In addition, transitions between almost every pair of states, whether they belong to the same letter sub-HMM or to different letter models, are allowed (see an example in Fig. 9). Each transition represents a unique interpretation of a single primitive segment, as a sequence of consecutive letter parts. The probability associated with each transition (a_{ij}) is a multiplication of all the following probabilities:

– The probability of the segmentation point represented by state i to occur.
– The probability that all the consecutive letter parts included in the current primitive segment were not separated.
– The probability of the two relevant letters to appear in a raw, when working in case of inter-letter transitions in an HMM having a clique topology for a large lexicon environment.

Segmentation statistics are calculated during the training phase after performing the segmentation algorithm on the training samples. Linguistic statistics that relate to letters and pairs of letters occurrence frequency are derived from the lexicon (if necessary). A similar interpolation is used to calculate initial state probability – π_i. Using this method, the initial state probability is not trivial as in all previously mentioned HMMs even if a model-discriminant is used, let alone an HMM of a path-discriminant kind. Each one of the three states of a letter, that may turn out to be the first letter, is a candidate to become the initial state of the optimal path. Hence, when this method is used for a limited lexicon, i.e., when using a model-discriminant HMM, there are six alternatives as only the first two letters may be the starting ones. However, in a large lexicon scenario, all 26×3 states have a potential for being initial states.

Unfortunately, although the approach seems to be very promising theoretically and experimentally, training such a system is rather complicated and laborious, especially when a large lexicon is used, as in the experiments that were reported.

To overcome this complication, a new system was developed based on a variable duration HMM (VDHMM) [10–12]. Each letter sub-HMM is cut to the minimum of one state only, thus reducing the number of transitions enormously. No interpolations are necessary to compute all the probabilities involved (π_i, γ_i, a_{ij}), since they depend only on the lexicon. A new and robust segmentation algorithm, which can split (almost) all the touching letters, but on the other hand may decompose one letter into four primitive segments, was used. In previously mentioned HMM-based word recognition systems, a transition between states covered only one primitive segment. In this case, states are isomorphic to complete letters, and each transition is associated with a variable size block that contains between one to four consecutive primitive segments. In other words, states are given a new dimension of duration. Mathematically speaking, each state is associated with an extra probabilistic parameter that expresses the probability of the letter it represents to be split into d primitive segments:

$$D = \{P(d|q_i)\}; \text{ where } P(d|q_i) = Pr\{duration(q_i) = d\}$$

Furthermore, the induction step of the Viterbi algorithm is updated as follows:

$$\delta_t(j) = \max_{1 \le i \le N} \max_{1 \le d \le D} \delta_{t-d}(i) a_{ij} P(d|q_j) b_j(O^t_{t-d+1})^d$$

A different approach of handling this kind of HMM given the same segmentation algorithm can be found in [40,41]. An adaptive length Viterbi algorithm was developed to handle transitions of variable duration. At each step of the Viterbi recursion, for each state, we choose the block of primitive segments that has the maximum character probability among all possible blocks at this stage. The algorithm should overcome the fact that at any step, different paths may have covered a different number of primitive segments. So, some paths may terminate earlier than others.

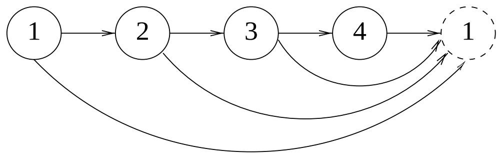

Fig. 10. An alternative to a variable duration letter sub-HMM

An interesting simulation (alternative) to a variable duration HMM (VDHMM) using conventional HMM basics was proposed by Chen et al., [9] in order to create a letter model (see an example in Fig. 10). Each letter sub-HMM contains 4 states. Transitions are allowed between a state and either the following state or the first state of the next letter. The resulting sub-HMM contains 4 different paths from the first state of the current letter to the first state of the next one, having 1, 2, 3, or 4 transitions respectively. Using the same letter duration statistics as in [10–12], transition probabilities are assigned in such a way that the probability that a path contains d transitions is the same as the probability that a letter contains d primitive segments.

A very similar letter sub-HMM was used by Senior et al., [72–74]. However, in this system the input to the HMM was a sequence of probability vectors representing the probability of the current frame taken from the word image to be (a part of) each letter. Recurrent neural networks were used in order to estimate a letter probability. There was only one difference in the letter sub-HMM topology, allowing a self-loop transition in the last state. The self-loop enables letter paths longer than 4 transitions with probabilities decreasing exponentially. Using this kind of letter models showed significant improvement in comparison with simple models of one state only (and a self-loop), or models that force different minimum duration on different letters. The additional stage in the recognition process, i.e., transforming a feature vector into a letter probability vector preceding the HMM phase, is actually a change in concept. The resulting method becomes very similar to a segmentation-based one although it does not perform any meaningful segmentation. In [74], a combination of model-discriminant HMMs and a path-discriminant HMM was suggested. In this case, only some of the lexicon words are given a unique model. In the path-discriminant model, each inter-letter transition was associated with the probability of the relevant pair of letters to appear in the lexicon. The final decision should take into account both the highest scored word model and the word interpretation resulting from the path-discriminant stand-alone HMM.

4.4 Specialized methods

In this section we review a unique segmentation-based method that was designed for a large lexicon environment. The dynamic programming that was used in all previously mentioned methods to find an optimal segmentation-recognition pair, was abandoned for the benefit of a heuristic search through the domain of all possible interpretations. The heuristic search used is a variation of the famous A^* algorithm. We finish this section with a brief review of three methods that use pure segmentation followed by a separate recognition module.

A widely known non-HMM approach was presented by Bozinovic and Srihari [4] (see early versions in [78] and [79]). This seems to be the first system to deal with a large lexicon. We will refer to the top two levels which are relevant here, the word and letter levels respectively. In the word level an ordered list of hypotheses is maintained. Each hypothesis relates to a prefix of a lexicon word and a sequence of consecutive primitive segments starting at the leftmost primitive segment of the word image. However, different hypotheses may relate to segment sequences of different lengths. The score of a hypothesis is the average matching score between the respective characters and the union of primitive segments they claim to represent. At each step the current best hypothesis is being expanded, and the list is resorted given the new hypotheses that replaced the original ones. The resulting new list is sent for word formation, or lexicon lookup, where inadmissible hypotheses are pruned, and the one with the highest rating is expanded and so on. An expansion is the process of generating all possible single characters that might have proceed the given hypothesis. Following this is the establishment of new hypotheses as the concatenation of the original one with each new character. The character generation task is the role of the letter level. Given the rightmost SP (segmentation point) that appears in the given hypothesis, we give possible character interpretations to the symbols that appear in the next primitive segment or in the union of the next two or three primitive segments. The primitive segment union alternatives are due to the fact that a character may spread over three primitive segments at the most.

In contrast to all methods mentioned in this section, the following three do not integrate pre-segmentation within the recognition phase. In this case, one segmentation is executed prior to the recognition module. Methods of this kind can use either one of the following techniques to overcome segmentation mistakes. When the segmentation algorithm does not produce many alternatives, one can execute the recognition module for each possible segmentation, choosing the best interpretation over all segmentation alternatives. This method was successfully tested by Yanikoglu et al., in [88]. Another option independent of the first one is to use intensive post-processing such as string-to-string correction. A string-to-string correction technique is the letter-level version of the popular minimum edit-distance algorithm that is discussed in detail in the segmentation-free section and in Sect. B in the appendix. However, when used in this scenario, special operations such as substituting two letters for one and vice versa are considered in order to handle over and under-segmentation respectively. One should note that post-processing of this type limits the size of the lexicon since it requires each possible word to be compared with the input. Examples of methods that have utilized this post-processing can be found in [60] and [66]. These methods are very rare because one cannot rely on the segmentation algorithms that exist today. Moreover, overcoming segmentation mistakes in the post-processing is more limited than within the recognition algorithm. As a result, we chose not to give these special methods further discussion in this paper.

5 Perception-oriented methods

In this section we present another approach to word recognition. Although it is not as popular as the two former approaches, we find this approach significant as it seems to resemble a good working model, namely the human reading scheme. However, there are only a few methods that prefer this approach, perhaps because of implementation issues. Unlike all different methods that were discussed earlier, a perception-oriented method does not work sequentially. Sequential recognition methods attempt to match an ordered list of observations/primitive segments with a word pattern (either specific or variable) from left to right. When using perception-oriented methods, efforts are made, in a bottom-up manner, to identify letters anywhere in the observation sequence derived from a word image. Next, a decision procedure is run, resulting in an interpretation which consists of the best non-overlapping set of letters, and the most likely interpretation of possible gaps that were left between them. In a similar way, a human reader doesn't search a word image from left to right in an attempt to match consecutive segments with possible characters. Usually a human reader seeks for the most reliable characters he/she can recognize. In the next stage he/she tries to match the remaining gaps with candidates from the lexicon that agree with the preliminary characters that were already recognized ([83]). As a result of these similarities, this approach is often called human-like reading. Sometimes, when the final output of a method of this type is only the letters that were recognized with high confidence, it is classified as a key letter extraction method.

A perception-oriented method is independent of previous operations. Therefore, the input to the recognition module can be either one of feature vectors associated with primitive segments, or observations derived directly from a word image. In other words, segmentation is not obligatory. Moreover, perception-oriented methods do not require certain level of word representation and can perform a match in any level of representation. Except for one system, all the experiments using methods of this kind were carried out using small lexicons only. However, we believe that these methods (as proven by the exceptional case) are capable of handling large lexicons as well.

In Edelman et al., [22], recognition is based on the alignment of letter prototypes, given an image of a cursive word. We perceive this method to be one of the most successful word recognition methods in general and in perception-oriented in particular. The main stages of the recognition process are as follows:

– *Anchor point extraction.* This is performed by tracing the vertical and horizontal extrema of the contour and the line endings (including T-junctions).
– *Stroke detection.* Strokes are recognized by prototype alignment, using affine transformations computed from anchor-point correspondences.
– *Letter hypothesization.* Potential instances of each of the 26 letters are detected. Every instance at this stage has a score that reflects its closeness to the part of the contour with which it is aligned. This score takes into account the distance between the relevant part of the contour and the respective prototype. The amount of distortion undergone by the stroke's prototype and intrusion to forbidden zones defined by each prototype reduce the score.
– *Instance filtering.* The previous stage frequently results in the detection of several overlapping instances of the same letter. The set of all detected instances of a letter is filtered to discard those that are clearly superfluous, based on the concept of *domination.* A character instance is said to dominate another one if it has a higher score and the overlap between the two is sufficiently large.
– *Interpretation.* At this stage, a best-first search is used to assemble the interpretation string out of the set of all detected letter instances. The algorithm reminds the one used by Bozinovic et al., [4,78,79] – both relate to the famous A^* algorithm. At each step, one expands the most promising node among all those situated along the current expansion frontier. The expansion is performed by a function that returns all one letter continuations to the right of the current string. Each one of the new expanded strings is checked for validity. For example the distance between the additional letter and the rightmost letter of the original string should not exceed a threshold. The entire cycle is repeated a preset number of iterations, after which all strings that reached the right end of the word image are sorted.
– *Application of lexical knowledge.* The fast spell-checking function available in the Symbolics Lisp environment is used in order to synthesize the lexical neighborhood and statistical Englishness of a string.

In the method presented by Simon et al., [77,76], "anchors" i.e., features that are reliable, are found in the symbolic (feature) description chain of the word. The symbolic description chain is produced, like in some segmentation-based methods, by translating each feature vector that might be found in a segment into a unique symbol. Thus, one or more contiguous fields representing one or more segments, determine the probability of the existence of one or more characters at a certain location. Then, the words from the lexicon that may agree with the partial recognition are chosen as possible candidates. In the next step, a dynamic matching is done between the fields of the symbolic description chain that have not yet been interpreted and the remaining characters of the candidate word. A very similar process is described by Cheriet et al., in [16]. In this case the letter evaluation stage results in a special string of the form:

$$(A^*(a|b)^*)^+$$

where A denotes a set of possible labels of the input key letter, a and b denote the two types of segmentation points that may be produced by the segmentation algorithm that was used. A can be a set of labels, say $A = \{k_1, \ldots, k_m\}$, where m is the number of possible interpretations of the input key letter.

In Cote et al., [19,20], a cyclic process is used to choose and rate possible candidates. The bottom-up route, i.e., filtering words according to observed features that are compatible with letters at certain locations, is very similar to the one described in the previous method. Nevertheless, there are two significant modifications. First, the interaction between the different levels – features, letters, and words – is based on a perception model, using excitatory activation of a neural type. A discovered feature fires and activates all letters that may relate to it. In a similar way letters activate words. A letter (word) reacts to the sum of all of its excited neighboring features (letters). The second difference with respect to the first method, concerns the representation of a letter position. In this case a possible letter position is fuzzy and not accurate, thus enabling one to deal with the ambiguity of letter location. The top-down process is used to validate or reject active words, according to the existence or absence, respectively, of features that are required to create the missing letters. After a few cycles, an ordered list of possible lexicon words is obtained.

In Leroux et al., [50], a pre-segmentation stage produces segments that refer to possible letters. Then a pre-recognition process filters out segments that either contain nothing but noise or seem like very noisy letters. Segments that look like a portion of a letter are aggregated. The first phase of recognition consists of finding whether a presumed letter is a member of a class of letters as typified by primitive feature groups ligatures. The second phase aims at "breaking up" the classes using geometric knowledge. This process yields either a letter or a set of letters. The third phase consists of validating the membership class and assigning a confidence value to the letter. In the final word recognition, if possible, only characters with a correct confidence value, an evaluation of the word length, or any other structuring element are used. Otherwise, we compute edit-distance between a lattice of candidate letters and each of the lexicon words.

Some classic segmentation-free methods that go through some changes often become perception-oriented. In general, most holistic word recognition methods can be transformed to follow the perception principles. Given a sequence of features derived from a word image, it is possible to search for possible letter appearances using letter templates. A letter template consists of a grammar that heuristically determines the production rules that describe valid mappings of feature sequences into letters. The search for identifiable letters results in a temporary string. This string contains the recognized letters, some special characters that represent a group of letters that cannot be distinguished, and gaps that did not match any letter template. Then a string matching against all the lexicon words is performed. The smaller the lexicon is, the fewer letters need to be identified. In the following are a few examples of holistic methods in their perception-oriented form.

First, we refer to the holistic method of Parisse [65] mentioned in the segmentation-free section (Sect. 3). Parisse used a minimum edit-distance method that utilizes word profiles for comparison. In this case, given a profile of an unknown word image, all possible profiles of known two-letter n-grams (a combination of any two letters), are identified and scored. Finally, the highest scored n-grams are retained. The direct comparison between complete profiles was replaced with finding the best correlation between the input image and a lexicon word, with respect to the locations of the obtained n-grams.

Govindaraju et al., [37] used an ordered list of strokes extracted from the skeleton of a given word image. The strokes were ordered according to the presumed order they were produced. A special mechanism that attempts to discover the missing temporal information was used to conclude the stroke order. Each letter prototype referred to a possible stroke sequence that is likely to appear whenever the letter is part of the cursive word.

6 Discussion

In this paper we have reviewed the field of offline cursive word recognition. We have mainly dealt with the various methods that were proposed to realize the core of recognition in a word recognition system. These methods were discussed in view of the two most important properties of such a system: the size and nature of the lexicon involved, and whether or not a segmentation stage is present. We have found it useful to classify the field into three categories: segmentation-free, segmentation-based, and perception-oriented.

Segmentation-free methods compare a sequence of observations derived from a word image with similar references of words in the lexicon. A very popular technique to find the most likely interpretation is the minimum edit-distance algorithm, that is implemented by classic dynamic-programming tools as well as unique HMMs. Due to the well-known Baum-Welch algorithm, the HMMs are more easily trained and adapted than simple dynamic-programming methods based on minimum edit-distance.

Segmentation-based methods look for the best match between consecutive sequences of primitive segments and letters of a possible word. One can use either dynamic programming, graphical models, or HMMs to find the optimal solution. These methods should be used carefully as they strongly depend on the segmentation algorithm.

The last category, the perception-oriented approach, relates to methods that perform a human-like reading technique. In this case anchor features found all-over the word are used to boot-strap a few candidates for a final evaluation phase. This category was found to be unique and independent of segmentation issues and therefore deserves a special class by its own.

It is our view that the field is not mature enough to enable a comparison of methods. Consequently, we have decided not to describe experimental results achieved by the various systems. One can justify the fact that comparison is inadmissible at the this stage by the following reasons. First, many methods used proprietary databases or tested their systems on small lexicons despite their potential to handle larger ones. Second, performance of a system depends on many factors besides the recognition algorithm used. A complete word recognition system is more complicated and relies on several modules such as preprocessing, post-processing, segmentation, character recognition, etc. In addition, the chosen feature set also contributes in a major way to the final performance. Finally, some methods we reviewed were not investigated and tested up to their limit. These methods went through preliminary experiments only.

In what follows, we briefly discuss some of our more important observations regarding the offline cursive word recognition field. We also include some observations that can be used in order to select or design a system for various needs. Several parameters should be considered when one is about to evaluate a method. Among them are: the lexicon capabilities (with respect to size and nature), recognition rate, average processing time per word, complexity of programming and training, amount of training data required, portability, etc. Usually the most important constraint determined by the application level is the lexicon involved. However, in real-time applications, the average processing time per word is the most important issue. Obviously, the recognition rate would be a trade-off between all the above mentioned factors.

One of the most important factors that determine the recognition rate achieved by a system is the feature set involved. It is our view that most of the problems in word recognition, especially in a large lexicon environment, emerge from a wrong feature selection. At the moment it seems that there is no feature set that can be considered optimal. While high-level features are limited, i.e., they cannot distinguish among similar words, low-level features are very sensitive to noise in addition to their tendency to be writer dependent.

Another significant issue that one should consider is training. One should train a system with real data, i.e., holistic models should be trained with samples of the relevant word. Letter-based models should be trained with samples of the respective letters extracted from images of cursive words. This is very important because the ligatures that are often integrated with letters of a cursive word may influence its interpretation. In many cases, an extended training phase, i.e., more samples of cursive words/letters, would improve performance. Unfortunately, there has not been a breakthrough yet, after which recognition improvement could only be asymptotic without gaining a major progress.

Although we did not treat other modules besides the core recognition, we have observed that the state of the art word recognition system is a combination of modules of different developers and not a single system mentioned. Fortunately, such a combination is feasible since most modules are independent of the preceding or following ones. Moreover, it is very popular nowadays to integrate several recognition algorithms in a single system. When several methods are used in parallel, the final decision can be made by a weighted calculation over their answers. In other cases, the methods are merged in hierarchy, resulting in a situation where one method performs as a lexicon reducer, letting the more powerful method(s) concentrate on the most likely words. Both combinations if used with independent methods can improve the recognition rate. A hierarchical combination may also save computation time.

It seems that the problem of word recognition in the case of small and static lexicons is essentially solved. The improvements that are still needed can be achieved through optimization of already existing methods. However, this is definitely not the case for limited and large lexicons. We would like to point out that we may be close to the limits of stand-alone word recognition, and efforts should be made to improve post-processing techniques that will take advantage of syntax, context, and other external parameters.

References

1. L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. Inequalities 3:1–8 (1972)
2. L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann. Math. Stat. 41(1):164–171 (1970)
3. R. E. Bellman. Dynamic Programming. Princeton University Press, 1957
4. R. M. Bozinovic and S. N. Srihari. Off-line cursive script word recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 11(1):68–83 (1989)
5. H. Bunke, M. Roth, and E. G. Schukat-Talamazzini. Off-line recognition of cursive script produced by a cooperative writer. In Proceedings Int. Conf. on Pattern Recognition, pages 383–386, 1994
6. H. Bunke, M. Roth, and E. G. Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden Markov models. Pattern Recognition 28(9):1399–1413 (1995)
7. T. Caesar, J. Gloger, A. Kaltenmeier, and E. Mandler. Recognition of handwritten word images by statistical methods. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 409–416, 1993
8. R. G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(7):690–706 (1996)
9. M. Y. Chen and A. Kundu. An alternative to variable duration HMM in handwritten word recognition. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 82–91, 1993
10. M. Y. Chen, A. Kundu, and S. N. Srihari. Handwritten word recognition using continuous density variable duration hidden Markov model. In Proceedings IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, pages 105–108(V), 1993
11. M. Y. Chen, A. Kundu, and S. N. Srihari. Variable duration hidden Markov model and morphological segmentation for handwritten word recognition. In Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, pages 600–601, 1993
12. M. Y. Chen, A. Kundu, and S. N. Srihari. Variable duration hidden Markov model and morphological segmentation for handwritten word recognition. IEEE Transactions on Image Processing 4(12):1675–1688 (1995)
13. M. Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition (hwr) using a single contextual hidden Markov model. In Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, pages 669–672, 1992
14. M. Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition using a hidden Markov model type stochastic network. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(5):481–496 (1994)
15. M. Y. Chen, A. Kundu, J. Zhou, and S. N. Srihari. Offline handwritten word recognition using hidden Markov model. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 563–577, 1992
16. M. Cheriet and C. Y. Suen. Extraction of key letters for cursive script recognition. Pattern Recognition Letters 14(12):1009–1017 (1993)
17. W. Cho, S.-W. Lee, and J. H. Kim. Modeling and recognition of cursive words with hidden Markov models. Pattern Recognition 28(12):1941–1953 (1995)
18. L. Cooper and M. W. Cooper. Introduction to Dynamic Programming. Pergamon Press, 1981
19. M. Cote, E. Lecolinet, M. Cheriet, and C. Y. Suen. Building a perception based model for reading cursive script.

In Proceedings Int. Conf. on Document Analysis and Recognition, pages 898–901, 1995
20. M. Cote, E. Lecolinet, M. Cheriet, and C. Y. Suen. Automatic reading of cursive scripts using human knowledge. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 107–111, 1997
21. W. P. de Waard. An optimised minimal edit distance for handwritten word recognition. Pattern Recognition Letters 16(10):1091–1096 (1995)
22. S. Edelman, T. Flash, and S. Ullman. Reading cursive handwriting by alignment of letter prototypes. International Journal of Computer Vision 5(3):303–331 (1990)
23. A. Eliaz and D. Geiger. Word-level recognition of small sets of handwritten words. Pattern Recognition Letters 16(10):999–1009 (1995)
24. J. T. Favata. General word recognition using approximate segment-string matching. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 92–96, 1997
25. J. T. Favata and S. N. Srihari. Recognition of handwritten words using a hypothesis generation and reduction methodology. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 237–251, 1992
26. G. D. Forney. The Viterbi algorithm. IEEE Proceedings 61(3):268–278 (1973)
27. P. Gader, M. Mohamed, and J.-H. Chiang. Segmentation-based handwritten word recognition. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 215–225, 1992
28. P. D. Gader, M. Mohamed, and J.-H. Chiang. Handwritten word recognition with character and inter-character neural networks. IEEE Transactions on System, Man and Cybernetics - Part B: Cybernetics 27(1):158–164 (1997)
29. A. M. Gillies. Cursive word recognition using hidden Markov models. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 557–562, 1992
30. M. Gilloux. Writer adaptation for handwritten word recognition using hidden Markov models. In Proceedings Int. Conf. on Pattern Recognition, pages 135–139, 1994
31. M. Gilloux, J.-M. Bertille, and M. Leroux. Recognition of handwritten words in a limited dynamic vocabulary. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 417–422, 1993
32. M. Gilloux and M. Leroux. Recognition of cursive script amounts on postal cheques. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 545–556, 1992
33. M. Gilloux, M. Leroux, and J.-M. Bertille. Strategies for handwritten words recognition using hidden Markov models. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 299–304, 1993
34. M. Gilloux, M. Leroux, and J.-M. Bertille. Strategies for cursive script recognition using hidden Markov models. Machine Vision and Applications 8(4):197–205 (1995)
35. N. D. Gorsky. Experiments with handwriting recognition using holographic representation of line images. Pattern Recognition Letters 15(9):853–859 (1994)
36. V. Govindaraju and R. K. Krishnamurthy. Holistic handwritten word recognition using temporal features derived from off-line images. Pattern Recognition Letters 17(5):537–540 (1996)
37. V. Govindaraju, D. Wang, and S. N. Srihari. Using temporal information in off-line word recognition. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 529–543, 1992
38. D. Guillevic and C. Y. Suen. Cursive script recognition applied to the processing of bank cheques. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 11–14, 1995
39. D. Guillevic and C. Y. Suen. HMM word recognition engine. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 544–547, 1997
40. Y. He, M. Y. Chen, and A. Kundu. Handwritten word recognition using HMM with adaptive length Viterbi algorithm. In Proceedings IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, pages 153–156(III), 1992
41. Y. He, M. Y. Chen, and A. Kundu. Off-line handwritten word recognition using HMM with adaptive length Viterbi algorithm. In Proceedings Int. Conf. on Pattern Recognition, pages 460–462, 1994
42. S. Impedovo, L. Ottaviano, and S. Occhinegro. Optical character recognition - a survey. International Journal of Pattern Recognition and Artificial Intelligence 5(1-2):1–24 (1991)
43. G. Kim and V. Govindaraju. Handwritten word recognition for real-time applications. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 24–27, 1995
44. G. Kim and V. Govindaraju. Recognition of handwritten phrases as applied to street name images. In Proceedings IEEE Conf. on Computer Vision and Pattern Recognition, pages 459–464, 1996
45. G. Kim and V. Govindaraju. A lexicon driven approach to handwritten word recognition for real-time applications. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(4):366–379 (1997)
46. G. Kim and V. Govindaraju. Handwritten phrase recognition as applied to street name images. Pattern Recognition 31(1):41–51 (1998)
47. F. Kimura, M. Shridhar, and Z. Chen. Improvements of a lexicon directed algorithm for recognition of unconstrained handwritten words. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 18–22, 1993
48. F. Kimura, M. Shridhar, and N. Narasimhamurthi. Lexicon directed segmentation-recognition procedure for unconstrained handwritten words. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 122–131, 1993
49. F. Kimura, S. Tsuruoka, M. Shridhar, and Z. Chen. Context directed handwritten word recognition for postal service applications. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 199–213, 1992
50. M. Leroux, J. C. Salome, and J. Badard. Recognition of cursive script words in a small lexicon. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 774–782, 1991
51. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In Sov. Phys. Dokl. 10, pages 707–710, 1966
52. Y. Lu and M. Shridhar. Character segmentation in handwritten words - an overview. Pattern Recognition 29(1):77–96 (1996)
53. S. Madhvanath and V. Govindaraju. Using holistic features in handwritten word recognition. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 183–198, 1992

54. S. Madhvanath and V. Govindaraju. Holistic lexicon reduction. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 71–81, 1993
55. S. Madhvanath and V. Govindaraju. Holistic lexicon reduction for handwritten word recognition. In Proceedings of the SPIE - Document Recognition III, pages 224–234, 1996
56. S. Madhvanath, E. Kleinberg, V. Govindaraju, and S. N. Srihari. The HOVER system for rapid holistic verification of off-line handwritten phrases. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 855–859, 1997
57. S. Madhvanath and V. Krpasundar. Pruning large lexicons using generalized word shape descriptors. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 552–555, 1997
58. G. M. T. Man and J. C. H. Poon. Cursive script segmentation and recognition by dynamic programming. In Proceedings of the SPIE - Character Recognition Technologies, pages 184–194, 1993
59. M. Mohamed and P. Gader. Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(5):548–554 (1996)
60. J.-V. Moreau, B. Plessis, O. Bougeois, and J.-L. Plagnaud. A postal check reading system. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 758–766, 1991
61. C. R. Nohl, C. J. C. Burges, and J. I. Ben. Character-based handwritten address word recognition with lexicon. In Proceedings of U.S. Postal Service 5th Advanced Technology Conference, pages 167–182, 1992
62. T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the Levenshtein metric. IEEE Transactions on Computers c-25(2):172–178 (1976)
63. C. Olivier, T. Paquet, M. Avila, and Y. Lecourtier. Recognition of handwritten words using stochastic models. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 19–23, 1995
64. T. Paquet and Y. Lecourtier. Handwriting recognition: Application on bank cheques. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 749–757, 1991
65. C. Parisse. Global word shape processing in off-line recognition of handwriting. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(4):460–464 (1996)
66. B. Plessis, A. Sicsu, L. Heutte, E. Menu, E. Lecolinet, O. Debon, and J.-V. Moreau. A multi-classifier combination strategy for the recognition of handwritten cursive words. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 642–645, 1993
67. L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. IEEE Proceedings 77(2):257–286 (1989)
68. L. R. Rabiner and B.-H. Juang. An introduction to hidden Markov models. IEEE ASSP Magazine 3(1):4–16 (1986)
69. L. R. Rabiner and B.-H. Juang. Fundamentals of Speech Recognition (Chapter 6). Prentice Hall, 1993
70. G. Saon and A. Belaid. Off-line handwritten word recognition using HMM-MRF approach. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 118–122, 1997
71. K. M. Sayre. Machine recognition of handwritten words: A project report. Pattern Recognition 5(3):213–228 (1973)
72. A. W. Senior. Off-line handwriting recognition: A review and experiments. Technical report, Cambridge University, England, 1992
73. A. W. Senior and F. Fallside. An off-line cursive script recognition system using recurrent error propagation networks. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 132–141, 1993
74. A. W. Senior and A. J. Robinson. An off-line cursive handwriting recognition system. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(3):309–321 (1998)
75. M. Shridhar, G. Houle, and F. Kimura. Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 861–865, 1997
76. J. C. Simon. Off-line cursive word recognition. IEEE Proceedings 80(7):1150–1161 (1992)
77. J. C. Simon and O. Baret. Regularities and singularities in line pictures. International Journal of Pattern Recognition and Artificial Intelligence 5(1-2):57–77 (1991)
78. S. N. Srihari and R. M. Bozinovic. A multi-level perception approach to reading cursive script. Artificial Intelligence 33(2):217–255 (1987)
79. S. N. Srihari and R. M. Bozinovic. A multi-level perception approach to reading cursive script. In Proceedings Int. Joint Conf. on Artificial Intelligence, pages 844–847, 1987
80. S. N. Srihari, V. Govindaraju, and R. K. Srihari. Handwritten text recognition. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 265–274, 1994
81. T. Steinherz, E. Rivlin, and N. Intrator. Extending small and limited word recognition methods for large lexicons. Technical report, Tel-Aviv University, Israel, 1999. In preparation
82. C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition - a survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(8):787–808 (1990)
83. I. Taylor and M. M. Taylor. The Psychology of Reading. New York Academic, 1983
84. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey. Pattern Recognition 29(4):641–662 (1996)
85. A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Transactions on Information Theory IT-13(2):260–269 (1967)
86. A. El Yacoubi, J.-M. Bertille, and M. Gilloux. Towards a more effective handwritten word recognition system. In Proceedings Int. Workshop on Frontiers in Handwriting Recognition, pages 378–385, 1994
87. A. El Yacoubi, J.-M. Bertille, and M. Gilloux. Conjoined location and recognition of street names within a postal address delivery line. In Proceedings Int. Conf. on Document Analysis and Recognition, pages 1024–1027, 1995
88. B. A Yanikoglu and P. A. Sandon. Recognizing off-line cursive handwriting. In Proceedings IEEE Conf. on Com-

puter Vision and Pattern Recognition, pages 397–403, 1994

Appendix

A Hidden Markov Models (HMMs)

This section briefly reviews HMMs and the aspects relevant to the word recognition methods that are mentioned in Sects. 3,4. The interested reader should refer to Rabiner et al., [69,68,67] for a complete explanation. Given a set of states $q_1, \ldots, q_S$ and a codebook of observations $v_1, \ldots, v_O$, a first-order HMM can be described by the parameter $\lambda = (\Pi, A, \Gamma, B)$, where

the initial state probability $\Pi = \{\pi_i\}$,
where $\pi_i = Pr(q_i \ at \ t = 1)$
the state transition probability $A = \{a_{ij}\}$,
where $a_{ij} = Pr(q_j \ at \ t+1 | q_i \ at \ t)$
the last state probability $\Gamma = \{\gamma_i\}$,
where $\gamma_i = Pr(q_i \ at \ t = T)$
the observation probability $B = \{b_j(k)\}$,
where $b_j(k) = Pr(v_k \ at \ t | q_j \ at \ t)$

There are two kinds of HMMs: discrete or continuous density. In a discrete HMM, observations are associated with a symbol probability of the nearest symbol in the codebook, while in a continuous density HMM, the observation probability is calculated as a mixture of Gaussians that refer to the codebook symbols. We chose not to mention which kind of HMM (discrete or continuous density) was used in any of the methods since it has no effect on the nature of the method that was used and can be considered as arbitrary.

HMMs gained their popularity in classification problems in general and cursive word recognition in particular because they are most suitable to model the variance that usually appears in symbolic description chains of objects. They can be easily designed to handle a combination of a symbol deletion, substitution with another or an extra symbol insertion that may change the predicted description entirely.

HMMs are often used in word recognition systems as word models. When small or limited lexicons are used, the model-discriminant method is preferred. In this case, each model is associated with a separate HMM that attempts to estimate the likelihood of the observation sequence to represent the particular word. Moreover, in an HMM of this kind, (using either one of segmentation-free or segmentation-based approaches) each word model is a concatenation of the relevant letter sub-HMMs as illustrated in Fig. 2. Both initial and last state probabilities are degenerated because a path must start on the leftmost state and end on the rightmost one. The a posteriori probability of a model is the highest probability of a path through it, found by the Viterbi algorithm [85, 26]. Because of the nature of handwriting that dictates a left to right movement, there are no backwards transitions in a sub-HMM representing a letter. It is very important to distinguish between word model creation phase that can be letter-oriented, and recognition phase that can utilize such a model using a segmentation-free, i.e., complete word oriented method. On the other hand, an HMM-based method for large lexicons uses one HMM called path-discriminant HMM. This single model is used for the whole lexicon, and different paths, i.e., state sequences are used to distinguish one word from the others. An example of a single HMM that can handle large lexicons is presented in Fig. 3. In general, an HMM of this kind can be described as a graph with a clique topology, where each node represents a letter sub-HMM and the arcs represent transitions between them. Using 26 nodes, one for each letter, assures that each legal word is isomorphic to a path. However, each path is a family of paths because each node along the path is actually a sub-HMM itself. The output of the recognition process is the word associated with the path of maximum probability over all possible paths with respect to the observation sequence, i.e., a path $I^* = i_1^* i_2^* \ldots i_T^*$, that satisfies:

$$Pr(O, I^* | \lambda) = \max_I [Pr(O, I | \lambda)] = \max_{i_1, i_2 \ldots i_T} [\pi_{i_1} b_{i_1}(o_1) \times \prod_{t=2}^{T} a_{i_{t-1} i_t} b_{i_t}(o_t) \times \gamma_{i_T}]$$

Usually the Viterbi algorithm [85,26] is used to find this optimal path.

B Minimum edit-distance

Given a sequence of observations derived from an input word image and a database of similar descriptions of all possible words, one usually attempts to find a stored description that has the minimum edit-distance with respect to the input. Usually, Levenshtein's metric concept ([51]) is used. The minimum edit-distance between two chains of symbols denoted by $o_1 \ldots o_m$ and $r_1 \ldots r_n$ is the cheapest cost one needs to pay in order to transform one chain to the other using the operations of symbol deletion, substitution, or insertion of an extra one. This problem has a well-known dynamic-programming solution:

$$d(o_1 \ldots o_m, r_1 \ldots r_n) = \min \begin{cases} d(o_1 \ldots o_{m-1}, r_1 \ldots r_{n-1}) + sub(o_m, r_n) \\ d(o_1 \ldots o_{m-1}, r_1 \ldots r_n) + ins(o_m) \\ d(o_1 \ldots o_m, r_1 \ldots r_{n-1}) + del(r_n) \end{cases}$$

where $del(r_n), sub(o_m, r_n), ins(o_m)$ are the cost parameters for the three above-mentioned operations respectively. A cost parameter may be constant or a function of the symbol.

C Dynamic programming

The essence of most segmentation-based recognition methods is a dynamic-programming technique for calculating the best matching score between a word $C_1 C_2 \ldots$

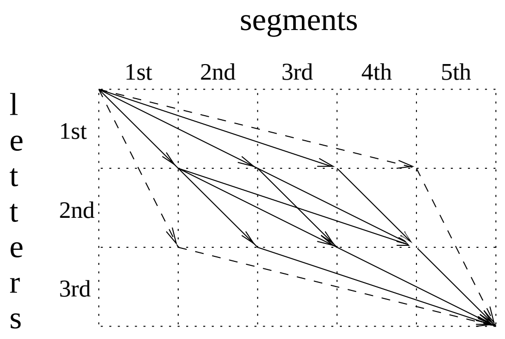

Fig. 11. A graphical illustration of the dynamic-programming algorithm used for finding the best match between a three-letter word and a sequence of five primitive segments. The dashed lines represent paths in which a single segment contains two characters, and therefore will be considered only if the segmentation criteria tolerates such alternatives

C_{N_c} and an image represented by a sequence of primitive segments $S_1S_2\dots S_{N_s}$:

$$\begin{aligned}&value(N_c,N_s)\\&=\max_k\{value(N_c-1,k)+match(S_{k+1,N_s},C_{N_c})\\&\quad\mid N_c-1\le k<N_s\}\end{aligned}$$

where match (segment(s),character) is an evaluation function that estimates the correlation between a union of primitive segments and the respective character, and $S_{l,m}$ is the sequence of primitive segments from the lth to the mth. The assumption that each primitive segment contains at most one character can be relaxed. If for example, we allow for a primitive segment to correspond to two characters, we need to update the maximization above by checking additionally the following expression:

$$value(N_c-2,N_s-1)+match(S_{N_s,N_s},C_{N_c-1}C_{N_c})$$

This requires additional training of the evaluation function so it can handle a pair of characters in a single primitive segment.

A graphical illustration of the dynamic-programming approach is shown in Fig. 11.

In order to find the best matching score of a single word, an array $A(N_c,N_s)$ is formed, where N_c and N_s are the number of characters and primitive segments respectively. The node $A(i,j)$ holds the value of the best match between the first i characters in the string and the first j primitive segments. The induction step is computed according to the above-mentioned formula. After the dynamic-programming phase is finished, the optimal segmentation that is associated with the highest score can be restored by back tracing.

Tal Steinherz received his B.Sc degree in mathematics, physics and computer science from the Hebrew University in Jerusalem. Currently he is a Ph.D. student at the Computer Science Department in Tel-Aviv University. His research interests are offline cursive word processing and recognition, document analysis and writer model.

Ehud Rivlin received the B.Sc and M.Sc degrees in computer science and the M.B.A degree from the Hebrew University in Jerusalem, and the Ph.D from the University of Maryland. Currently he is an Associate Professor in the Computer Science Department at the Technion, Israel Institute of Technology. His current research interests are in machine vision and robot navigation.

Nathan Intrator received his Ph.D. from Brown University. He currently holds a position at the Computer Science Department in Tel-Aviv University and at the Institute for Brain and Neural Systems at Brown. Dr. Intrator has contributed to the mathematical analysis of the BCM theory (Bienenstock Cooper and Munro, 1982) and applied his methods to various real-world applications including speech and 3D object recognition. His areas of research include neural computation, high-dimensional statistics, methods for controlling bias and variance of predictors, combining experts and model discovery via neural network. He has recently developed training methodologies for feed-forward networks working in many parameter models. Dr. Intrator also studies various data analysis methods which are based on the exploratory projection pursuit statistical framework. These methods improve model interpretation and network prediction.