Offline General Handwritten Word Recognition Using an Approximate BEAM Matching Algorithm

John T. Favata, *Member*, *IEEE*

Abstract—A recognition system for general isolated offline handwritten words using an approximate segment-string matching algorithm is described. The fundamental paradigm employed is a character-based segment-then-recognize/match strategy. Additional user supplied contextual information in the form of a lexicon guides a graph search to estimate the most likely word image identity. This system is designed to operate robustly in the presence of document noise, poor handwriting, and lexicon errors, so this basic strategy is significantly extended and enhanced. A preprocessing step is initially applied to the image to remove noise artifacts and normalize the handwriting. An oversegmentation approach is taken to improve the likelihood of capturing the individual characters embedded in the word. The goal is to produce a segmentation point set that contains one subset which is the correct segmentation of the word image. This is accomplished by a segmentation module, employing several independent detection rules based on certain key features, which finds the most likely segmentation points of the word. Next, a sliding window algorithm, using a character recognition algorithm with a very good noncharacter rejection response, is used to find the most likely character boundaries and identities. A directed graph is then constructed that contains many possible interpretations of the word image, many implausible. Contextual information is used at this point and the lexicon is matched to the graph in a breath-first manner, under an appropriate metric. The matching algorithm employs a BEAM search algorithm with several heuristics to compensate for the most likely errors contained in the interpretation graph, including missing segments from segmentation failures, misrecognition of the segments, and lexicon errors. The most likely graph path and associated confidence is computed for each lexicon word to produce a final lexicon ranking. These confidences are very reliable and can be later thresholded to decrease total recognition error. Experiments highlighting the characteristics of this algorithm are given.

Index Terms—Handwriting recognition, OCR, BEAM search, word segmentation, machine reading, pattern recognition.

1 General Introduction

Machine recognition of offline handwritten words [20] presents a problem of transforming a two-dimensional digitized image of a word into a symbolic (textual) representation of that word. Many successful recognition algorithms [2], [3], [7], [8], [10], [11], [12], [20] use some variation of a segment-then-recognize/match approach either implicitly or explicitly. Other competing approaches exist including holistic (or segmentation-free) modeling of word recognition [18]. Our explicit segmentation approach first segments [6] the word image into a series of segments which may represent a full, partial, or spurious character. Next, the segments are arranged into some spatial order, usually sequentially, with a symbol (character) estimation being performed on groups of segments. This step results in an implicit directed graph which represents many possible interpretations of the word. The last step is to prune the paths with the help of a lexicon which supplies the necessary search constraints and/or the language model (or document context). In practice, path pruning becomes difficult when dealing with degraded documents because expected segments can be missing from the image. In addition, spurious segments, misspelling, and other noise complicate the matching process. A general matching strategy will be developed to overcome these degradations and produce reasonably robust recognition. Since it is difficult to discuss our topic of matching without the complete context of a working recognition system, we will provide an overview of one particular word recognition system.

The system and approach that we describe has a number of favorable algorithmic and practical advantages. It is designed to work with most general handwriting styles (mixed combinations of discrete and cursive characters) and places few restrictions on the author. The overall design is modular (see Fig. 1) and each module can be fine-tuned or replaced with improved versions without major impact on the other modules. The ability to substitute better trained character recognition modules is very important and, in practice, done frequently. The graph matching module is designed so that it is relatively easy to incorporate new heuristics which compensate for new types of document noise or handwriting characteristics. Overall, the number of parameters that must be estimated for acceptable system performance is relatively small and fine-tuning is quickly done. An important emergent property of this algorithm is the reliability of the confidences that it produces. The ability to threshold these confidences reduces error rates without

• *The author is with the State University of New York College at Buffalo, CIS, Chase Hall, Room 202, 1300 Elmwood Ave., Buffalo, NY 14222. E-mail: jtfavata@ieee.org, favata@cse.buffalo.edu.*

Manuscript received 26 July 1999; revised 2 Aug. 2000; accepted 4 May 2001. Recommended for acceptance by A. Kundu.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 110314.

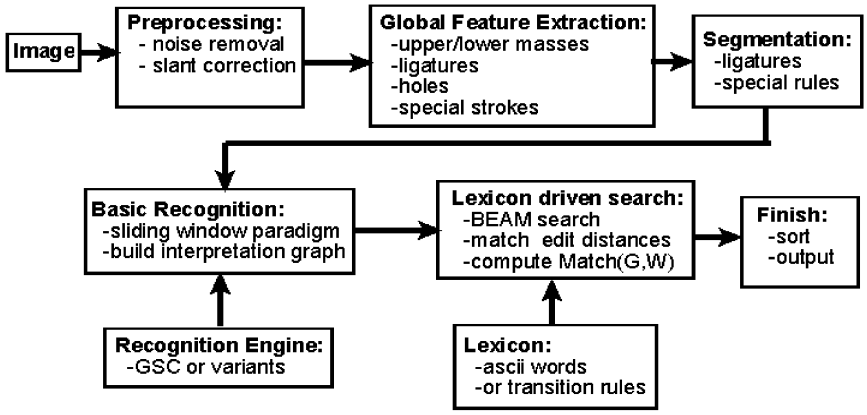

Fig. 1. Overview of system.

unacceptable rejection rates and is useful in many practical word recognition applications. A disadvantage of this algorithm is the fact that it difficult to model its performance under all conditions and the addition of heuristics at different stages of the algorithm further complicates this problem. Each of the modules, in some sense, incorporates metaknowledge about the nature of handwriting, so the final composite model is very heterogeneous. The algorithm is not designed for recognition speed but for accuracy and thresholding capability. However, fast implementations have been produced with a small degradation in recognition performance.

2 ECSWR Algorithm

The foundation of the ECSWR (Explicit Character-Segment Word Recognizer) algorithm is built on the segment-then-recognize/match paradigm as discussed above. Since word recognition can be viewed as a constraint satisfaction problem, the most basic constraints are the plausible character shapes embedded in the strokes of the word. This, by itself, produces a large number of possible word interpretations (symbol identities) because of a certain level of ambiguity in handwritten characters and imperfect optical character recognition (OCR) estimation. These interpretations must then be ranked using the vocabulary constraints or context of the language. Only those interpretations which best approximate the allowable n-gram character transition rules of each vocabulary word are considered as possible word identities. One way of implementing this set of constraints is by maximizing some objective function which estimates the likelihood of a vocabulary word given a sequence of strokes extracted from the word. The success of this approach critically depends on the accuracy of the segmentation algorithm, the recognition behavior of the character recognizer (OCR), and the metrics used in the matching algorithm.

2.1 General Approach to Recognition

The fundamental goal of this system is to segment, isolate, and recognize the characters which make up the word. This task is relatively simple for the case of type 1 (discretely printed) [23] words because the segmentation naturally follows from the white space between characters. For types 2, 3, and 4 (cursive, mixed, and touching), the problem is much more difficult because finding a correct segmentation requires detecting and cutting the appropriate strokes in the word image (see Fig. 2). The ECSWR algorithm takes the strategy of over-segmentation, that is, producing a (possibly large) number of segmentation points in the image. This can produce many possible word identities which must be eliminated during both the character recognition and lexicon matching stage. The goal of the segmentation strategy is to produce a segmentation of the word in which one subset of segments will isolate all characters. Failure to isolate each character doesn't necessary produce a recognition error but increases

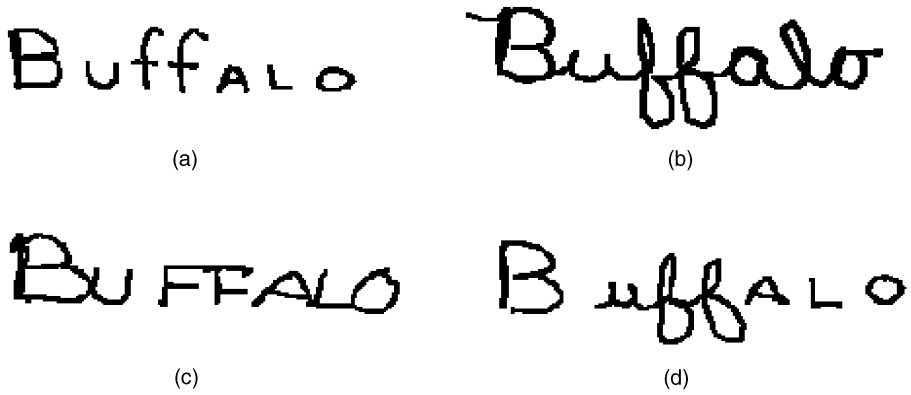

Fig. 2. Four type of handwriting: (a) discrete, (b) cursive, (c) touching discrete, and (d) mixed.

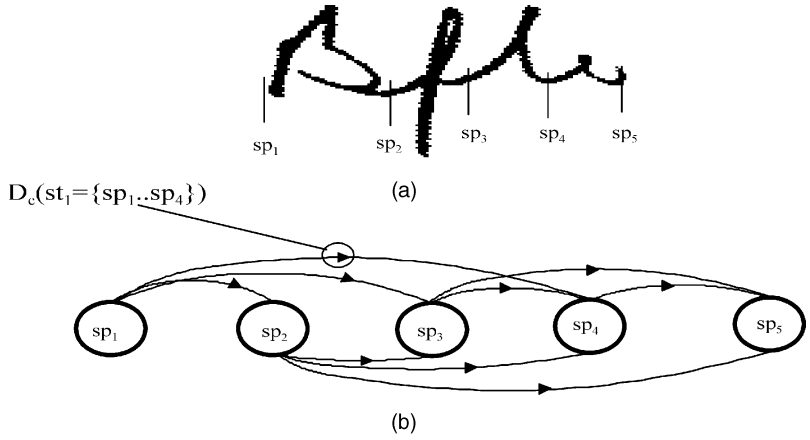

Fig. 3. (a) Word with segmentation points. (b) Directed graph G.

the likelihood of an error. Many segmentation errors will be recovered at the segment-character matching stage.

The OCR algorithm must be of high quality and have a relatively stable behavior, that is, incorrect isolation of the characters will not produce spurious high confidence decisions. We will denote this response as $\mathrm{D_c(st_x)}$ as the *distance* (related to a probability density function) to character class *c*, for some segment $\mathrm{st_x}$. An optimal response is not always easily obtainable with character recognition algorithms. Generally, we must consider all possible word interpretations by searching all valid paths through segments that produce significant $\mathrm{D_c(st_x)}$ responses. We can represent all interpretations of the word by creating (after OCR) an augmented directed graph $\mathrm{G = (V, STE)}$, where V is a set of nodes which represent the segmentation points of the word, and STE is a set of edges which contain the values $\mathrm{D_c(st_x)}$ along with other information. We will label the elements of V as $\{\mathrm{sp_1, sp_2, \ldots, sp_n}\}$, where each $\mathrm{sp_j}$ is a computed segmentation point of the word. The elements of V are naturally ordered left to right with $\mathrm{sp_1}$ being the leftmost segmentation point of the word, and $\mathrm{sp_n}$ being the rightmost segmentation point in the word. For convenience, we define the set $\mathrm{SP} = \{\mathrm{sp_1, sp_2, \ldots, sp_n}\}$, which is related to V, and another set $\mathrm{ST} = \{\mathrm{st_1, st_2, \ldots, st_j}\}$ of valid segments in IW. The argument $\mathrm{st_x}$ to $\mathrm{D_c(.)}$ is a member of the set ST. To clarify, each $\mathrm{st_x}$ element itself consists of some subset of segmentation points $\{\mathrm{sp_i..sp_j}\}$ $(\mathrm{i < j})$ from SP and represents all the strokes (image segments) between segmentation points $\mathrm{sp_i}$ and $\mathrm{sp_j}$. By definition, *single-span* segment $\mathrm{st_x}$ spans exactly two contiguous segmentation points, say $\mathrm{sp_k}$ to $\mathrm{sp_{k+1}}$, while a *multispan* segment spans more than two contiguous segmentation points (see Fig. 3). Throughout this discussion, we distinguish between W, which is a sequence of ASCII characters, and its word length, |W|. Each character of W is denoted as $\mathrm{C_j, j = 0..|W|}$. We also denote IW, which is the two-dimensional pixel representation (image) of the word. Each graph G will contain a set of valid paths $\mathrm{GP} = \{\mathrm{P_1, P_2, \ldots, P_s}\}$, which start at the first segmentation point of the word and stop at the last segmentation point of the word. Each $\mathrm{P_j} = \{\mathrm{st_{\Pi[j]0}, st_{\Pi[j]1}, \ldots, st_{\Pi[j]b}}\}$ contains a sequence of segments and can be a possible interpretation of the word. The set of functions $\Pi[.]$ is a particular valid permutation of ST with $\Pi[.]_\mathrm{k} \in \{1..\mathrm{N}\}$, for some N, which is the maximum number of all valid segment groupings contained in IW. A natural restriction on each $\mathrm{P_j}$ is that the segments must be contiguous, that is, for example, the rightmost segmentation point of $\mathrm{st_{\Pi[j]0}}$ must be the leftmost segmentation point of $\mathrm{st_{\Pi[j]1}}$. Another natural constraint of handwriting is that no multispan segment spans more that five segmentation points, or, equivalently, four single-span segments.

2.2 Average Word Distance Estimation

The ECSWR algorithm can be considered a bottom-up approach which estimates $\mathrm{D_c(ST)}$, for all character classes ($\mathrm{c = 1..26}$, upper and lower cases are currently folded), over all valid segments in ST. As discussed, this results in the construction of graph G. The next step is to estimate the likelihood of each lexicon word. This is done by searching G for a path $\mathrm{P^* \in GP}$, which gives the maximum weighted average confidence for each lexicon word. For example, let a word $\mathrm{W_k}$ in the lexicon L be made up of the sequence of characters, $\mathrm{W_k = \{C_1, C_2, .., C_n\}}$; the task is to find the best path $\mathrm{P^*}$ that maximizes an objective function $\mathrm{Match(G, W_k)}$. This is our estimation of the likelihood that the image IW contains the lexicon word W. We compute this estimate for each word in the lexicon. The final decision for the identity of the word image is that lexicon word which has the largest score over all other words, i.e.,

$$\mathrm{W_{indentity} = \arg \max_{W \in L} AMatch(G,\ W,\ N(G,\ W)_{best})}.$$

See Section 4 for more details.

2.3 Modules

This section will give an outline of the basic recognition modules of the ECMWR algorithm. These modules perform preprocessing, global feature detection, segmentation, OCR, and graph matching. The following algorithms have gone through several generations of evolution and exist in several different forms. The early forms were strictly pixel representation based and used to evaluate the overall strategies of the paradigm. Later forms are chaincode-based for enhanced speed [19]. The latest version uses a combination of chaincode and edge representation for both

speed and ease in developing advanced segmentation and feature extraction algorithms.

2.3.1 Preprocessing

Several general preprocessing steps are done on the composite raw image before word recognition starts. These steps attempt to normalize the image as much as possible. The preprocessing steps are performed once over the whole image before any recognition. It is assumed that the image has been binarized from gray scale.

Slant Correction. This step attempts to correct the general character slant of a word. An estimation is made of the average slant of the vertical strokes in the word and a shear is applied to correct for this slant. No attempt is made to correct for the baseline slant of the word.

Noise Removal. Each isolated (disconnected) component of the word is identified and its mass is computed. If the mass is below a threshold, that component is eliminated from further consideration. This step generally removes much of the background noise (salt and pepper) and other small artifacts such as the i-dot over certain characters.

Smoothing and Stroke Thickness Normalization. The raw image is smoothed to reduce edge noise and small stroke gaps are filled. This step also tries to make sure that all strokes are at least several pixels thick. This ensures that the chaincode generation step will be successful.

2.3.2 Feature Generation

After preprocessing, a number of features from the word are extracted. These features are critical for the segmentation and help in the detection of certain characters. They are computed from the chaincode description of the upper and lower contours of the word. The features currently used are: *across*, *ascending*, *descending*, and *tee* strokes. In addition, several other features computed: *upper masses*, *lower masses*, and *holes*. The basic feature extraction strategy is to traverse the lower or upper contour chaincode descriptors and apply a set of rules involving contour direction and proximity to other contours. For example, to detect across strokes, the algorithm follows the lower contour looking for runs (intervals) of chaincode that are relatively horizontal. When such runs are identified, the algorithm tries to find matching runs on the upper contour. If a pair of lower and upper runs are found that are relatively horizontal and within approximately one average stroke width thickness (estimated earlier) of each other, an across stroke is detected. The other features are detected similarly using rules that are hardcoded into the feature extraction module. In addition to these features which are used for word segmentation, another set of GSC features (see Section 3) will be extracted for the recognition of the segment(s) using OCR.

2.3.3 Word Segmentation

The segmentation algorithm is built using a number of separate modules which generate segmentation points based on the (above listed) features found in the word image. All of the modules work on the image and the results of each algorithm are stored in a table. Next, the points are coalesced and redundant points are removed. The resulting points are the segmentation point set (SP) for the word. The basic segmentation strategy is to look for relatively horizontal strokes between sharp vertical strokes. We call these horizontal strokes the *ligatures* of the word and they usually connect one character to the next. Some characters, such as lower-case cursive *m* and *w*, are made up of several ligatures and we will have to account for this in our estimation of $D_c(\mathrm{st})$.

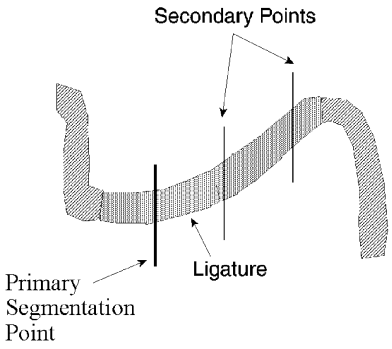

Fig. 4. Ligature with primary and secondary segmentation points.

It should be noted that, in general, segmentation algorithms require many heuristics for proper operation and the designer needs some insight into handwriting styles. Usually, some performance criteria (design goal) is set, such as, there must exist, at least, one set of segments that isolate each character. Other criteria, like the maximum number of segments that a character can span, are also used to measure the segmentation performance. There is a trade-off in the granularity (number of segments produced per character) of the segmenter (segmentation algorithm) and the ability of the OCR to reject partial characters.

Ligatures. Ligatures are detected from the horizontal strokes and one to three segmentation points are placed depending on the length of the ligature. This multiple segmentation strategy enhances character recognition. These carefully chosen segmentation points allow the system some flexibility (redundancy) for minor mismatching with the normalized training characters (see Fig. 4).

After the ligature detection, algorithms are applied which search for the most common cases of touching characters (that is, missing expected ligatures). A series of rules are applied to IW looking for the juxtaposition of certain features. If the conditions of a rule are satisfied, a segmentation point is generated splitting the two characters. We treat white space gaps between two strokes as a special case of a ligature.

Double Hole Segmentation. This heuristic is used to segment a special case of touching characters which very frequently occurs in handwriting. This case is the double *o*, which occurs when words with two sequential *o* characters are written in touching fashion (such as in w**oo**d). Such double *o*s are carefully segmented for recognition (see Fig. 5).

Left/Right Hole Segmentation. This heuristic is used to segment another case of character malformation in which two characters are written but there is no detectable ligature between the characters. This phenomena was observed in a sufficient number sample words to warrant adding this heuristic. A segmentation point is placed at the left of every valid hole in the word.

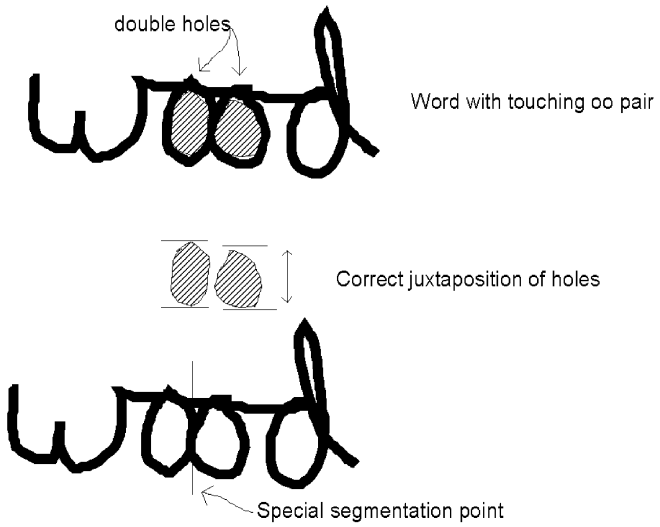

Fig. 5. Double hole segmentation rule.

Touching Tee Stroke Segmentation. This heuristic is used to segment another frequent case character malformation in which a horizontal *t* cross stroke touches an adjacent character. In particular, it is common for most authors to cross a word with two sequential *t* characters with one stroke (such as li***tt***le). It is necessary to detect the occurrence of this condition and split the double character (see Fig. 6).

Right Lower Mass Segmentation. This segmentation rule is used to segment words that contain descender to ascender character pairs (such as in the word ri***gh***t). In general, there is a rapid transition from the descender character to the ascender character (a nearly vertical stroke). This rapid transition does not fit the form of a ligature and must be detected and segmented explicitly.

The segmentation points from all of the segmentation algorithms are concatenated together and sorted according to spatial position. The next step is to remove redundant points from the set. The strategy considers ligature points as having the highest priority and removes all other segmentation points that are within a specified radius of the ligature point. The exact value of this radius is a small multiple (1.5 to 4.0) of the word's average line thickness. The result of this strategy is that all ligature points plus those special points that are sufficiently far away from the ligature points are retained. This is the final segmentation set that is used during the building of G (see Fig. 7).

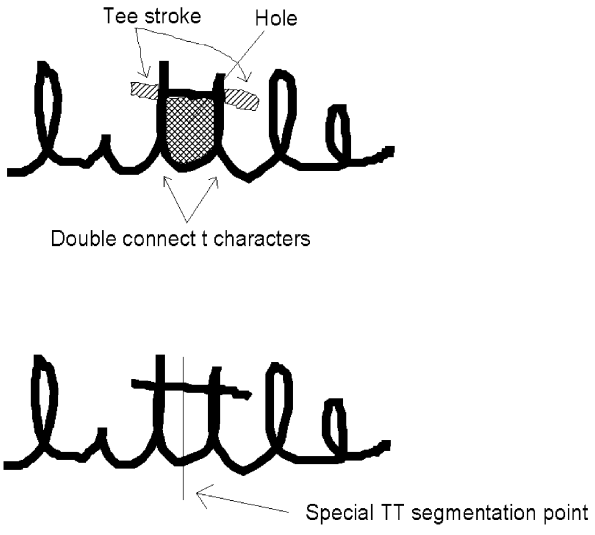

Fig. 6. TT segmentation rule.

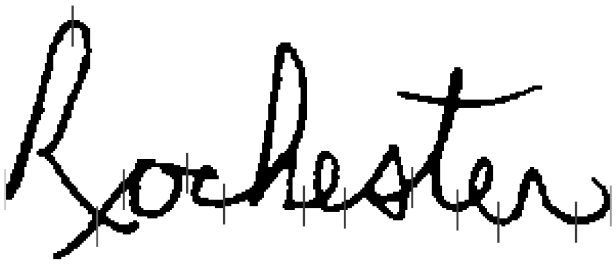

Fig. 7. Word with final segmentation points.

2.3.4 Building Graph G

After the valid segmentation points are determined, the next step is to reorder the segments into allowable configurations. A simple left-to-right ordering of the segments can produce incorrect sequences of segments with certain handwriting styles. The reordering algorithm analyzes the spatial relationships of the segments and produces a sequence of segments that are most likely to be compatible, that is, belong to the same character. After reordering, we build the graph G by estimating the $\mathrm{D_c}(.)$ measure for valid segment groups. This process is accomplished by performing a Basic Recognition Cycle (BRC) at each reordered valid segmentation point in the image. The basic recognition cycle of the system starts at a left point (LP) in the word. All segmentation points between the LP and the next N $(\mathrm{N} = 5)$ ligatures, including all special segmentation points, become the right points (RPs) (see Fig. 8). The stroke between the LP and each RP is physically cut, extracted by tracing the contours of the object between these points, and passed to the OCR for estimation of $\mathrm{D_c}(\mathrm{st_n}), \mathrm{st_n} = \{\mathrm{LP..RP}\}$. The system keeps track of the results and stores them in a data structure which represents the nodes V and edges STE of G.

3 The GSC OCR Algorithm

There are two main assumptions that are made about the behavior of the OCR [1], [5]. The first is that the response must have a peak when a group of segments (essentially a window) isolates a character and then this response must fall off rapidly as the window extends to the right (oversegments). The second assumption is that the response profile peaks with the true identity of the character in the current window (recognition accuracy). These two behaviors are difficult to achieve for real character recognizers for several reasons: 1) the feature space may not have cleanly defined class boundaries, that is, the classifier may not generalize smoothly to unseen exemplars and 2) the inherent ambiguity of certain (mostly cursive) characters. Usually, a certain amount of compromise is necessary in the design of the OCR. In general, it must be designed to vigorously "reject" objects that are not "seen" in the training

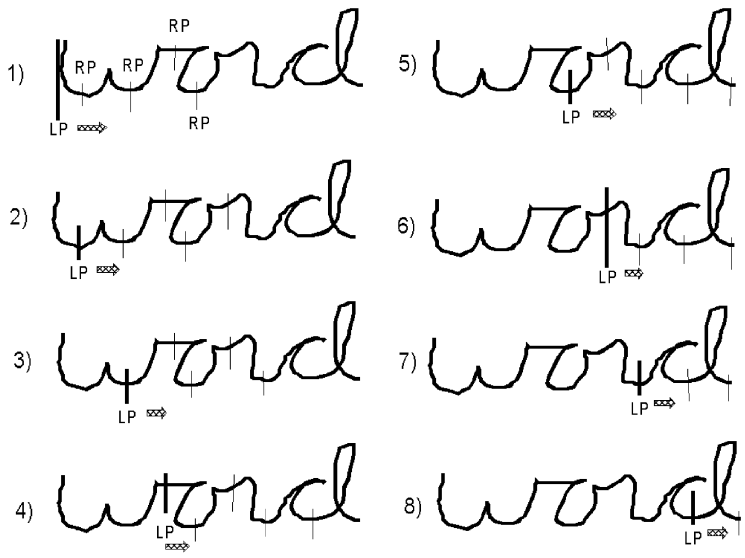

Fig. 8. Basic forward scan of word.

set. This can be done explicitly by adding a *rejection* class to the OCR. That is, explicitly train the recognizer with incorrectly isolated characters gathered from some representative database of words. Sometimes other auxiliary spatial contextual information is needed to modify the OCR response function. This auxiliary information can be incorporated in the form of modulation or scaling parameters which force the response to meet the above criteria and can include local or global characteristics of the word. In general, we will not perfectly meet the above criteria and the overall recognition accuracy will be penalized. In addition, some "fine tuning" may be necessary for optimal performance.

The character recognition system used in this work is based on three general feature categories. The features were chosen because they are somewhat orthogonal and are at different scales to each other. Collectively, these features are known as the Gradient, Structural, Concavity (GSC) feature set [13], [14]. The feature space is of dimension 512 and each feature is binary (0,1). An overview of the GSC features is as follows:

Gradient Features. These features are extracted by computing the gradient of the image using 3x3 pixel Sobel-like operators. A 192-bit feature vector is extracted which reflects 12 discrete ranges of the gradient subsampled on a 4x4 grid of the image. These features essentially capture local edge curvature information and are stored in a gradient feature map.

Structural Features. These features are computed using the gradient feature map and represent larger scale localized stroke information in the image. The structural features include short strokes of different angles and strokes that form right angles in various directions. A 192-bit vector is produced which reflects these local features.

Concavity Features. These features reflect the presence of holes and concavities in the image. The concavities are those which are basically pointing left, right, up, and down. A 128-bit feature is generated which codifies this information.

The main classification scheme used in this work is a weighted k-nearest neighbor (W-k-nn) classifier which computes a distance measure from the unknown character to a training set of labeled exemplars [16], [17], [21]. The exemplars (also called prototypes) are produced by automatically segmenting a large number of training words and then manually labeling correctly isolated characters. After feature extraction, during recognition, the k-nearest prototypes are combined to cast a weighted vote for the identity of the unknown object. A potential drawback of W-k-nn classifiers is that they can be generally slower than other classifier paradigms. Clustering (partitioning of the feature space) can be used compensate for some of the k-nn slowness by quickly reducing the number of character exemplars that need to be searched. The actual classifier used in this work improves upon W-k-nn by partitioning the feature space into two regions: *r-critical* and *r-noncritical*. R-critical regions contain localized class probability density functions (PDF) which significantly overlap. By carefully changing the W-k-nn metric in these regions, we reduce the probability of misclassification. Generally, tests among different classifier paradigms have indicated that W-k-nn is indeed very well-behaved for our character search mechanism because of a good roll-off response when given an under/over segmented character. This can be seen in the following way: A good match for a W-k-nn classifier requires two criteria: distance to a known labeled prototype and nearness to a number of similarly labeled prototypes. Nearness to a number of identically labeled class prototypes (a peak region in the PDF) tends to generate a confident labeling. Nearness to a number of dissimilarly labeled class prototypes can produce a correct identification, but at a lower confidence. The more distant an unclassified object is from the prototypes, the lower the decision confidence. Other classifier schemes can have nonsmooth or erratic behavior in certain regions of their feature space and may produce spurious high confidence decisions (depending how they are implemented and trained). Needless to say, the overall classifier response is dependent on the underlying

separability of the feature space, which depends on the features chosen. The addition of a specifically trained reject class for improved performance has not yet been implemented and will be added in the future.

The input to the GSC classifier is an extracted sequence of strokes. These strokes are passed to the GSC feature extraction algorithms which generate a 512-bit feature vector. After recognition of this feature vector by the W-k-nn GSC classifier, additional information about the context of the stroke segment (extracted during the segmentation phase) is provided to scale (fine tune) the response of the classifier, as discussed earlier. The number of ligature strokes and the number of holes spanned by the segments are also used to modulate the response of the GSC classifier. A lookup table of scale factors as a function of the number of ligatures and holes spanned is provided for each character class (a-z). In a sense, we are actually computing a distance function $\mathrm{DF_i}(\mathrm{ST}, \mathrm{H}, \mathrm{L})$, where H is the number of holes embedded in ST and L is the number of ligatures and the other special segmentation points spanned by ST. This table is tuned using a priori knowledge and optimized with a training set of words. The GSC features along with the W-k-NN classifier and scaling table generally satisfy the two recognition criteria outlined; however, misestimations do result in less than perfect word recognition especially with large lexicons. For the rest of this paper, we denote: $\mathrm{conf}(\mathrm{st}, \mathrm{C})$ as the confidence produced from the GSC recognizer for character C from segment st. This confidence function, which is computed approximately as $1.0\text{-}\mathrm{Df_j}()$, spans the range $0.0 <= \mathrm{conf}() <= 1.0$, with 1.0 being a very high confidence that indicates very low probability of OCR error.

4 Graph Search

If the output of the recognition stage was completely reliable, the decision of the word recognition system would be generated by simply tracing through the paths which had the highest recognition confidences. The basic graph matching algorithm would follow each path in the graph G and match characters to segments. The best paths would have a one-to-one match between the segments and characters, and the poorer paths would have unmatched (extra) segments or character (see Fig. 9). However, because of the general ambiguity of handwriting, system segmentation/OCR errors, and other failures, it is necessary to approximate a best cost path (see Fig. 10). An optimal assignment, under the metric chosen, of segments to characters must be computed, taking into account beforehand that some characters and segments may not be matched. The problem can have high computation complexity because G, in general, has many paths and there are multiple ways of assigning W to any particular path.

There are several algorithmic extremes to consider in our graph matching problem: exhaustive search, dynamic programming, and beam search.

Exhaustive Search. This brute force algorithm finds the best match over all paths in the graph. While this approach is guaranteed to find an optimum path, it is computationally expensive.

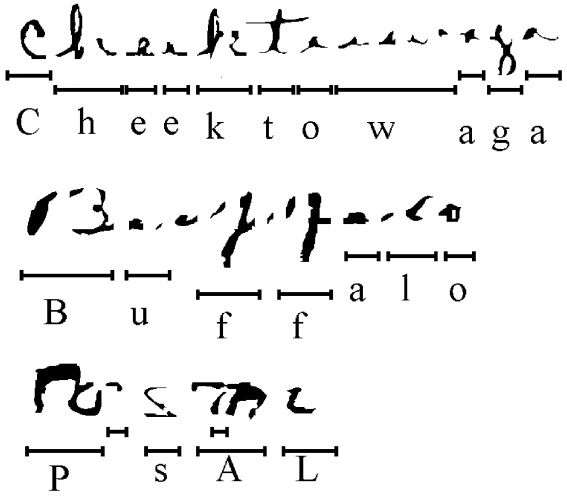

Fig. 9. Approximate match of characters to segments.

Dynamic Programming. This approach assumes that the global optimal can be computed by making an optimal choice for each partial match of a substring of a lexicon word W and a subpath in the graph. Generally, this is a very fast matching technique, if the local optimality assumption is met. In this problem, a sequence of optimal partial matches may not always give the global optimum. Also, inclusion of heuristics into the matching process can be difficult depending on the exact formulation and implementation of the recognition paradigm.

Beam Search. This approach is a compromise between exhaustive search and dynamic programming. The idea is to carry the L best partial matches forward by using a queue structure to hold each partial match [24]. The algorithm takes each of the current L matches and expands them to find the next incremental match between a character and segment. Each new partial match is evaluated by the match evaluation function and the next L best paths are retained. This process continues until each path reaches the end of the graph.

4.1 Approximate Matching with BEAM Search

There are a number of problems which can occur in the handwriting recognition process that can produce recognition failures. These can be roughly categorized as:

1. *Image Quality*. This includes digitization errors (such as undersampling), thresholding (poor grayscale to binary conversion), background artifacts, texture, ruler lines, and severely rotated text.
2. *Lexicon Errors*. Missing lexicon entries or the wrong form of the words (like missing plural forms) and missing alternate spellings of common words (such as color versus colour).
3. *Writing Errors*. Human misspelling, very poor handwriting technique, "unique" handwriting styles, odd embellishments, and the use of broad tipped writing instruments.
4. *System Errors*. Failure of segmentation (incomplete rules), OCR (incomplete training data), and misestimation of system parameters.

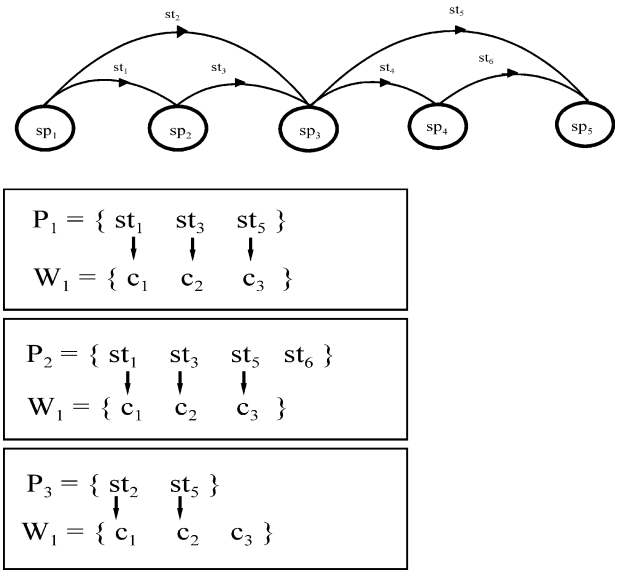

Fig. 10. Several possible matches of segments in G to characters in W.

These problems are generally interrelated, in that, they can occur in any combination with any degree of severity. Also, any of the first three will produce some form of the fourth. Consider the problem of matching an ASCII word from the lexicon with N characters with M segments. Recall that we defined C_k as the kth character of the word and st_j as the jth segment which may span more than two segmentation points. Normally, the image of a character can be split into multiple segments, for example, the character *w* can be split into two, three, or four single-span segments. The goal is to find an optimal placement of the character string with the segments that maximizes the match. Starting with the first character, C_1, we match it to the first segments that share the leftmost (LP) segmentation point, sp_1, of IW. These segments consist of a single-span segment and several multispan segments. A matching algorithm should first choose the best match of the current character with one of the segments and then continue with the next character and next unmatched set of segments that share a common right segmentation point (RP), with the now matched segment. Of course, the matching process is not this simple due to the potential errors outlined above.

Consider a partial match of C_1 to C_j with a path that contains $sp_1(LP)$ to $sp_k(RP)$, the following situations are likely to occur next:

1. *Normal Match*. C_{j+1} matches the next segments associated with $[sp_{k+1}..sp_{k+1+i}]$.
2. *Spurious Segments*. The next segments associated with $[sp_{k+1}..sp_{k+1+i}]$ are noise and there is no match to C_{j+1}.
3. *Missing Segments*. No segments correspond to C_{j+1}.
4. *OCR failure*. Proper segments present but OCR did not correctly identify underlying character.

An efficient way to search a graph, taking into account these anomalies, is a modified BEAM search algorithm, carrying the best L matches forward. Three heuristics are defined to compensate for the most significant types of errors (see Fig. 11).

Heuristic 1. Continue the path even if there is no OCR response for this character (Fig. 11a), that is, don't terminate a path if a character match cannot be made. When expanding the current path, the basic idea for this rule is if there isn't a match between the next segment and next character, continue expanding this path for a while (but with a lower score) before deciding to terminate it. This heuristic allows for recovery from some OCR errors and segment extraction errors and still finds the best match using the surrounding character and segment context to compensate.

Heuristic 2. Always assume a single segment is noise and don't match any character to it (Fig. 11b). The motivation for this rule comes from errors in the

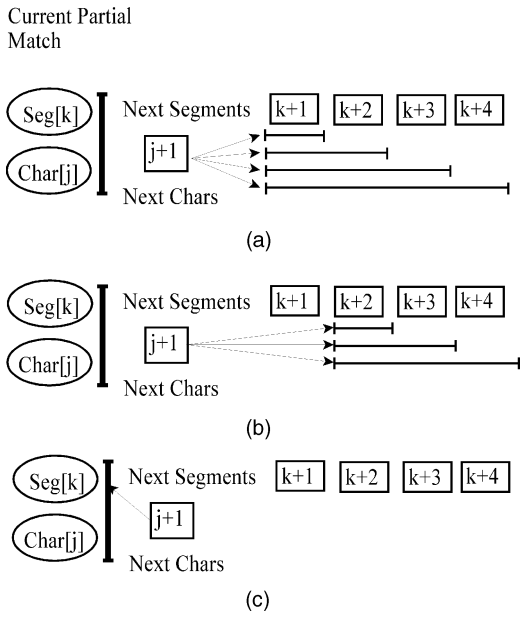

Fig. 11. Operation of heuristics: (a) continue path if C_{j+1} cannot be matched, (b) assume sp_{k+1} is noise, and (c) assume C_{j+1} cannot be matched.

segmentation algorithm. It is very difficult to segment characters from cursive script. Usually there will be some short segments that contain small strokes which actually belong to an adjacent character. This occasional over-segmentation of a word is difficult to control and many times the OCR system will confidently recognize the segment as the cursive letter *i* or *e*. Continuing a path with this decision sometimes causes the competing correct path to terminate early and results in a recognition error.

Heuristic 3. Assume no segments exist for this character, that is, forcibly skip the match of the current character and start matching the next character (Fig. 11c). This heuristic compensates for missing segments. Sometimes a word is not correctly segmented because of algorithm failure or characters that are touching in some unusual way. By continuing a path for a while assuming no segment is available for the current character, we compensate for this kind of segmentation failure. If this decision is wrong (there actually is a segment for the next character), the path will usually terminate quickly from competing paths.

Application of these heuristics involve estimating a penalty in the final overall matching score. In essence, our algorithm, in a breadth-first manner, follows all promising paths in G, matching characters to segments. Only the best matches are retained at any given point and the other paths are pruned. The cost function computed is an average quality measure of the match between the characters and the image segments. This function combines the overall OCR results with possible inexact matches between some of the characters and segments. Consider the partial match of all characters up to C_j with all segment paths of length k, we can define the cost of the match as:

$$\mathrm{Match(j,\,k)} = \frac{\sum \text{characters matched} \sum \text{character confidences} \sum \text{segment confidences}}{\mathrm{N}},$$

where the first term in the numerator is a string edit measure, the second term weighs the average character recognition confidence, and the third term accounts for the reliability of the segmentation points. N is a normalizing factor. The third term is not used in the present work.

The evaluation algorithm proceeds as follows: Let us define structure NODE, which records the current result of matching the contiguous segments of a path with a substring of W, as follows:

$$\mathrm{NODE} = [\mathrm{STM},\ \mathrm{SCM},\ \triangle\ (\mathrm{STM},\ \mathrm{SCM})],$$

where

$\mathrm{STM} = \{\mathrm{st}_{\Pi 1}, \mathrm{st}_{\Pi 2}, \ldots, \mathrm{st}_{\Pi \mathrm{j}}\}$, a set of contiguous segments in current partial path.

$\mathrm{SCM} = \{\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}\}$, a substring of $\mathrm{W}(\mathrm{k} < |\mathrm{W}|)$ in the current partial path.

$\triangle = \{\mathrm{d}_1(), \mathrm{d}_2(), \ldots, \mathrm{d}_\mathrm{L}()\}$ is a set of functions which record the assignment of a segment to a character, that is, $\mathrm{d}(\mathrm{st}_\mathrm{a}, \mathrm{C}_\mathrm{b}) \rightarrow 1$, if st_a is assigned to C_b, 0 otherwise.

At any given point in the algorithm, there will be a set of nodes $\mathrm{SURVIVE} = \{\mathrm{N}_{\Sigma 1}, \mathrm{N}_{\Sigma 2}, \ldots, \mathrm{N}_{\Sigma \mathrm{L}}\}$, which are the current best matches of the segments to the characters. To evaluate the score of a node, we use the following function:

$$\mathrm{AMatch(G,W,Node)} = \left(\frac{\left(\sum_{\mathrm{st}\in\mathrm{STM}} \sum_{\mathrm{j}=1,\mathrm{Cj}\in\mathrm{SCM}} \mathrm{d(st,C_j)} \right) \left(\sum_{\mathrm{st}\in\mathrm{STM}} \sum_{\mathrm{j}=1,\mathrm{Cj}\in\mathrm{SCM}} \mathrm{d(st,C_j)conf(st,C_j)} \right)}{\mathrm{Max}(|\mathrm{SCM}|,|\mathrm{STM}|)^2} \right),$$

where $|\,.\,|$ denotes the string length or path length (in segments) of the argument.

If a node of SURVIVE is still a partial path, it is expanded, using the heuristics defined above, and pruned using the value of the AMatch function to find another surviving set of best partial paths. In the end, a set of complete path matches will exist in the SURVIVE set. The node with the best match of W and G is now:

$$\mathrm{Node(G,W)_{best}} = \underset{N\in SURVIVE}{\arg\max}\ \mathrm{AMatch(G,W,N)}.$$

It is also possible to place the matching algorithm in a context free mode which does not require an explicit lexicon. Experiments have indicated that some form of context significantly improves recognition performance. Language n-grams, which can be estimated from some general lexicon source can be used in the matching algorithm instead of the explicit lexicon.

In choosing a match score function, several issues must be examined. For the following, we assume that the OCR algorithm is perfectly behaved, that is, meets the two OCR criteria discussed in Section 3. Consider the following cases:

1. **Perfect Segmentation.** We assume that a path through G exists which isolates all of the embedded characters of IW. Consider the lexicon word W which is the correct interpretation of IW. On average, $\mathrm{AMatch(G,W,N_j)} > \mathrm{AMatch(G,W_1,N_k)}$, for another competing word $\mathrm{W_1}$ and a node $\mathrm{N_k}$ (a different path). However, failures can occur in very special cases, which are combinations of certain lexicon words and handwriting styles. In these cases, additional surrounding textual context must be used to disambiguate the cases. For the rest of this discussion, we denote
$$\mathrm{Amatch_{opt}} = \mathrm{AMatch(G,W,N_j)}$$
as the best match of W to G.
2. **Missing Segments**. We assume that the segmentation algorithm has failed and a character $\mathrm{C_j}$ has not been properly segmented from IW. This means that there is an atomic segment $\mathrm{st_x}$ which contains two characters and no other contiguous path of segments splits these characters. The stroke which contains $\mathrm{C_j}$ is now merged with $\mathrm{C_{j-1}}$ or $\mathrm{C_{j+1}}$, its left or right neighbor in IW. Several problems now occur; $\mathrm{conf(st_x, C_j)}$ is now reduced and the overall value of the match is reduced to:
$$\mathrm{AMatch_{new}} = \left[\frac{|\mathrm{SCM}-1|}{|\mathrm{SCM}|}\right]\left[\frac{\sum \mathrm{conf_{sub}}}{\sum \mathrm{conf_{opt}}}\right]\mathrm{AMatch_{opt}},$$
where $\sum \mathrm{conf_{sub}}$ and $\sum \mathrm{conf_{opt}}$ are the sum of the confidences of all of the characters in G with and wthout, respectively, missing segments.

TABLE 1
Recognition Results Using all Heuristics, 500 City Word Images, Lexicon Size = 1,000

BEAM Breadth Width, L	Correct word image identity placement (percent) within the Top n (n=1,2,5,10,30) highest ranked lexicon words (see text)				
	Top 1	**Top 2**	**Top 5**	**Top 10**	**Top 30**
2	77.4%	81.4%	86.0%	89.0%	90.6%
4	81.8%	86.6%	89.4%	92.0%	94.8%
8	82.0%	86.8%	90.8%	92.8%	95.4%
16	82.0%	86.2%	91.2%	93.8%	95.8%

3. **Extra Segment**. In this case, we assume that our segmentation algorithm has introduced extra segments into the optimal path. The net effect is that the extra segments will not be matched to any characters (or matched to the *null* character). This reduces the optimal match to:

$$\mathrm{AMatch}_{\mathrm{new}} = \left[\frac{|\mathrm{SCM}|}{|\mathrm{STM}|}\right]^2 \mathrm{AMatch}_{\mathrm{opt}},$$

where $|\mathrm{STM}| > |\mathrm{SCM}|$.

In general, the more characters in the word, the less of a chance that a missing/merged or extra segment will cause a recognition error. This is reasonable behavior since longer words have additional character context to permit recovery from failures. Very short words are very sensitive to errors and recovery generally requires an extra postprocessing module which uses the context of adjacent words to pin the correct identity.

5 Experiments

The purpose of these experiments is to test the ability of the ECSWR algorithm to recognize unlabeled word images. The typical input to the algorithm is a word image extracted from some document along with a word lexicon that is generated from the document's context. The algorithm ranks each lexicon word according to the likelihood that it is the true identity of the unlabeled input word image. The output of the algorithm is the lexicon ranked (sorted) by confidences. Usually, only the highest ranked subset (top 30 words) of the lexicon is output and these words are typically passed to higher level document reading algorithms [9] (say, for sentence level recognition). The lexicon word with the highest numerical confidence value (Top 1 choice) can be taken as the decision by ECSWR of word's identity. However, in practical document reading algorithms, the top n $(n <= 30)$ highest ranked words can be combined with additional document context (or with other word recognition algorithms) and a reordering of the lexicon can be produced. An implementation of ECSWR was tested on a database of handwritten city words extracted from a sample of live mail (name, address, city, state, and ZIP fields on postal envelops), digitized at 300 ppi and binarized (using adaptive thresholding) from gray scale. This data is publically available on the Cedar CDROM 1 database in the *Bs* city name directories. Lexicons of 1,000 words (city names) were randomly prepared. A random selection of 500 words (images of handwritten city names) from the database were used in these experiments. The word type and frequency reflect the underlying statistics of the sampled mail stream. The GSC classifier was trained with 39,000 segmented isolated characters (discrete upper/lower and cursive upper/lower) which were selected from several databases. The first experiment used all three heuristics with various *breadth-width* sizes. The breadth-width is defined as the L-best number of nodes which are expanded during the BEAM search algorithm. Table 1 shows the results of this recognition experiment. As expected, as the breadth-width increased, so did the accuracy. Speed of recognition was not a concern in these experiments, but a breadth-width of eight seems to produce the best balance of accuracy and speed. Table 2 shows the same experiment with only Heuristic 1 used. Compared to Table 1 (all heuristics on), these two rules (Heuristic 2 and Heuristic 3) do improve the recognition accuracy. Table 3 demonstrates the effect of Heuristic 1. The significantly poorer recognition result

TABLE 2
Recognition Results with Heuristic #1 only, 500 City Word Images, Lexicon Size = 1,000

BEAM BreadthWidth, L	Correct word image identity placement (percent) within the Top n (n=1,2,5,10,30) highest ranked lexicon words (see text)				
	Top 1	**Top 2**	**Top 5**	**Top 10**	**Top 30**
2	77.8%	81.8%	85.4%	87.8%	89.6%
4	79.8%	84.0%	87.2%	89.2%	91.4%
8	80.6%	84.4%	87.0%	89.6%	92.0%
16	80.6%	85.2%	87.6%	90.0%	92.6%

TABLE 3
Recognition Results with no Heuristics, 500 City Word Images, Lexicon Size = 1,000

BEAM Breadth Width, L	Correct word image identity placement (percent) within the Top n (n=1,2,5,10,30) highest ranked lexicon words (see text)				
	Top 1	**Top 2**	**Top 5**	**Top 10**	**Top 30**
1	52.2%	55.0%	58.6%	59.2%	60.2%
2	59.4%	61.2%	62.0%	62.2%	62.8%
16	62.8%	65.0%	66.0%	66.4%	66.8%

(compared to Table 2) indicates that OCR misrecognition is a serious system problem and not surprising when dealing with cursive character recognition. A breath-width of one shown in Table 3 is essentially the dynamic programming solution to the graph matching problem and the recognition results are significantly poorer compared to the BEAM search algorithm. An important characteristic of handwritten word recognition algorithms is the ability to discriminate between complete word image objects and partial image fragments. A related desirable characteristic is the ability to reject a lexicon (by assigning a very low Top 1 choice confidence) if the lexicon does not contain the correct image label. These characteristics are important to higher-level document reading algorithms which try to find word boundaries in a contiguous sequence of word objects (such as reading phases or complete sentences). Table 4 shows the results of the full system in a 1,000 word discrimination test. This table demonstrates the ability of ECSWR to reject lexicons that do not contain the correct identity of the test word image. Two test sets were generated, one contained the truth of the test image (in the lexicon) and the other did not. The task was to see if thresholding the output confidences could reject the decision (ranked lexicon) of the system when the word's label was not in the lexicon. Table 4 is interpreted in the following way: Column A shows the experimental results of a word discrimination test using 2,376 word images and their associated 1,000 word lexicons that *did not* contain the true identity of the image, Column B is the same test but the associated lexicons contained the true identity of the word. Generally, as the threshold increased, the system rejected the test word images without a valid lexicon identity at a faster rate than with test word images that had a valid lexicon identity. Because of this general discrimination ability, the ECSWR system has been used in general document reading algorithms which use word recognition to discriminate between words and nonwords [4], [15]. An experiment was designed to gauge the ability of ECSWR to discriminate between complete and partial word images. Generally, given a partial word image (incorrectly segmented from a document image), we expect that the average confidences of the ranked lexicon will be lower than with correctly segmented (complete) words. This experiment, which compares the confidence distribution between complete and partial word images, was computed using two image test sets extracted from handwritten text pages [15] used in a sentence reading system. The first word set contained images which were correctly segmented from the page. The second word set contained partial words which are images missing several characters (usually at the beginning or end of the word) or a word image with additional spill (extraneous characters) from the adjacent words in the sentence. The lexicon for this test was approximately 1,600 words and contained the 1,000 most frequently used words in the English language. Fig. 12 shows the frequency histograms of the Top 1 choice confidences from these two image sets plotted simultaneously. The confidences produced from the partial word images are much lower than the confidences produced from the full word image set. In general, we conclude that this recognizer exhibits a fair level of discrimination ability and can be used in document reading algorithms which explictly search for word boundaries guided by an OCR system.

6 Summary

Several distinct algorithmic approaches have been described in the literature for offline machine handwritten

TABLE 4
The 1,000 Word Lexicon Discrimination Test with/without Lexicon Entries (See Text)

Top 1 Choice Accept Threshold	**Column A** Lexicons without word image identity (2376 images)	**Column B** Lexicons with word image identity (2376 images)	
	Words Accepted (above threshold)	Words Accepted	Error Rate
.18	50.5%	80.0%	15.2%
.24	13.1%	56.0%	6.3%
.28	3.5%	40.0%	2.5%
.32	0.7%	26.8%	0.7%

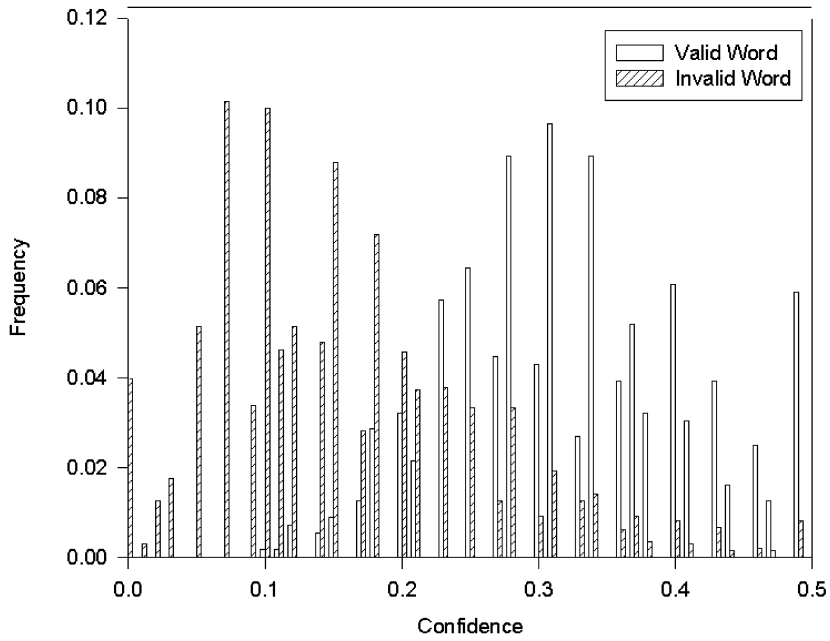

Fig. 12. Distribution of Top 1 choice word confidences between correctly (valid) and incorrectly (invalid) segmented word images in a sentence reading experiment (see text).

word recognition. Most involve some form of explicit word segmentation, estimation of embedded partial or full characters (symbols) and, finally, an optimal graph search for the most likely interpretation of the word image. Practical recognition algorithms used in real document reading applications must be robust to many different handwriting styles and document noise sources. In our formulation of the ECSWR algorithm, we wanted a recognition system that could accept a wide variety word images (from many different authors) with high recognition accuracy. In addition, we wanted good output confidence behavior so that the algorithm could be used in higher-level document reading algorithms. During the development of ECSWR, we noticed that our first attempts at algorithms which did straightforward segmentation, recognition, and optimal graph path searching produced a large number of errors. An investigation of these errors concluded that OCR misestimation, segmentation errors, image noise, and general writer ambiguity produced most of the recognition problems. We immediately concluded that document context in the form of a lexicon is needed to guide the graph matching process and is a very important ingredient for highly accurate algorithms. For segment-lexicon graph matching efficiency and flexibility, a BEAM graph matching algorithm was developed with the appropriate optimal partial path cost measure. Three important heuristics were developed for inclusion into the BEAM algorithm to compensate for some of the most common recognition failures. Experiments have shown that these heuristics significantly improve the recognition performance over a wide variety of handwriting styles. Test sets of handwritten words obtained from the postal and other general document domains have indicated that the algorithm is reasonably accurate and robust. Additional experiments testing the feasibility of ECSWR as a word boundary discriminator in sentence level recognition have been promising.

Future work includes improving all components of the system. A more sophisticated segmentation algorithm which uses a configuration analysis strategy to reorder the segment presentation to the OCR system is being planned. The GSC OCR algorithm is being improved using better features and newer classifier paradigms including multiple classifier combinations. Additional error analysis indicates that several new heuristics should be added to the BEAM match algorithm. We acknowledge that the incorporation of many heuristics may seem to produce an ad hoc algorithm, but feel that this type of approach is necessary for high performance algorithms.

REFERENCES

[1] A. Amin, "Off-Line Character Recognition: A Survey," *Proc. Fourth Int'l Conf. Documents Analysis and Recognition (ICDAR '97),* pp. 596-599, Aug. 1997.

[2] R. Bozinovic and S. Srihari, "Off-Line Cursive Script Word Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 11, no. 1, Jan. 1989.

[3] M.-Y. Chen, A. Kundu, and J. Zhou, "Off-Line Handwritten Word Recognition Using a Hidden Markov Model Type Stochastic Network," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 16, no. 5, pp. 481-496, May 1994.

[4] E. Cohen, "Interpreting Handwritten Text in a Constrained Domain," PhD dissertation, Dept. of Computer Science, State Univ. of New York, at Buffalo, Technical Report 92-06, Feb. 1992.

[5] R. Casey and E. Lecolinet, "A Survey of Methods and Strategies in Character Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, pp. 690-706, 1996.

[6] C.E. Dunn and P.S.P. Wang, "Character Segmenting Techniques for Handwritten Text—A Survey," *Proc. 11th Int'l Conf. Pattern Recognition,* vol. 2, pp. 577-580, 1992.

[7] P.D. Gader, M. Mohammed, and J.H. Chiang, "Handwritten Word Recognition with Character and Inter Character Neural Networks," *IEEE Trans. System, Man, and Cybernetics,* pp. 158-164, vol. 27, no. 1, 1997.

[8] M. Gilloux, M. Leroux, and J.M. Bertille, "Strategies for Cursive Script Recognition Using Hidden Markov Models," *J. Machine Vision and Applications,* vol. 8, pp. 197-205, 1995.

[9] G. Kim, V. Govindaraju, S. Srihari, "Extension of a Handwritten Word Recognition Method to Street Name Images," *Proc. Fifth Int'l Workshop Frontiers in Handwriting Recognition,* pp. 221-226, Sept. 1996.
[10] G. Kim and V. Govindaraju, "A Lexicon Driven Approach to Handwritten Word Recognition for Real Time Applications," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 4, pp. 366-379, Apr. 1997.
[11] F. Kimura, M. Shridhar, and N. Narasimhamurthi, "Lexicon Directed Segmentation—Recognition Procedure for Unconstrained Handwritten Words," *Proc. Third Int'l Workshop Frontiers in Handwriting Recognition,* pp. 122-130, 1993.
[12] J. Favata and S. Srihari, "Offline Recognition of Handwritten Cursive Words," *Proc. SPIE/IS&T Symp. Electronic Imaging Technology,* Feb. 1992.
[13] J. Favata and G. Srikantan, "A Multiple Feature/Resolution Approach to Handprinted Digit/Character Recognition," *Int'l J. Systems and Technology,* vol. 7, pp. 304-311, Dec. 1996.
[14] J. Favata, S. Srikantan, and S. Srihari, "Handprinted Character/Digit Recognition Using a Multiple Feature/Resolution Philosophy," *Proc. Fourth Int'l Workshop Frontiers in Handwriting Recognition,* pp. 57-66, Dec. 1994.
[15] J. Favata, S. Srihari, and V. Govindaraju, "Off-Line Handwritten Sentence Recognition," *Proc. Fifth Int'l Workshop Frontiers in Handwriting Recognition,* pp. 171-176, Sept. 1996.
[16] Y. Hammamoto et. al, "A Bootstrap Technique for Nearest Neighbor Classifier Design," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 1, Jan. 1997.
[17] T. Hastie and R. Tibshirani, "Discriminant Adaptive Nearest Neighbor Classifier," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 6, June 1996.
[18] S. Madhvanth and V. Govindaraju, "The Role of Holistic Paradigms in Handwritten Word Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 23, no. 2, Feb. 2001.
[19] S. Madhvanth, G. Kim, and V. Govindaraju, "Chaincode Processing for Handwritten Word Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 9, Sept. 1999.
[20] M. Mohammed and P. Gader, "Handwritten Word Recognition Using Segmentation-Free Hidden Markov Modeling and Segmentation-Based Dynamic Programming Techniques," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 5, pp. 548-554, May 1996.
[21] S.A. Nene and S.K. Nayar, "A Simple Algorithm for Nearest Neighbor Search in High Dimensions," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 9, pp. 989-1003, Sept. 1997.
[22] R. Plamondon and S.N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, pp. 63-84, Jan. 2000.
[23] C.C. Tappert, "An Adaptive System for Handwriting Recognition," *Graphonomics: Contemporary Research in Handwriting,* H.S.R. Kao, ed., Elsevier Science, 1986.
[24] P.H. Winston, *Artificial Intelligence,* second ed. Reading, Mass.: Addison-Wesley, 1994.

John T. Favata received the PhD degree in computer science from the State University of New York at Buffalo in 1992. He was a research scientist from 1992 to 1997 with the Center for Document Analysis and Recognition (CEDAR). He is currently a faculty member at the State University of New York College at Buffalo. His main research interests include handwritten sentence, word, character, and digit recognition. Other interests include general pattern recognition problems, feature extraction methodologies, classifier paradigms and machine learning, data mining techniques and knowledge representation, medical image processing, and recognition metrics. He is a member of the IEEE and the IEEE Computer Society.