Automated forms-processing software and services

by S. Gopisetty
R. Lorie
J. Mao
M. Mohiuddin
A. Sorin
E. Yair

While document-image systems for the management of collections of documents, such as forms, offer significant productivity improvements, the entry of information from documents remains a labor-intensive and costly task for most organizations. In this paper, we describe a software system for the machine reading of forms data from their scanned images. We describe its major components: form recognition and "dropout," intelligent character recognition (ICR), and contextual checking. Finally, we describe applications for which our automated forms reader has been successfully used.

Introduction

Forms processing is an essential operation in business and government organizations. Forms are structured documents that can be filled in, distributed, approved or rejected, stored, retrieved, and handled in other ways. While forms may be paper-based or on-line, the large majority of the forms that are in common use today are paper documents. Because of the many advantages they provide, there is an increasing trend toward image systems in which paper forms are scanned and converted to images and processed like on-line forms. One of the central problems in this process is the cost of capturing information from the scanned images. The technology and software to automate data entry from forms is the topic of this paper.

It has been estimated that approximately $250 billion is spent annually, worldwide, on keying information from paper documents[1]—and this is for keying only 1% of the available documents. Most of this cost is in human labor. When the process of data entry is automated, significant cost savings can be realized; in addition, the percentage of data that is brought on line can also be increased. For example, the U.S. Internal Revenue Service processed 200 million tax returns in 1993, of which 6% were filed electronically [1]. Only about 40% of the data on tax returns was keyed in, and the typical processing time on tax returns was four to six weeks. It is estimated that by the year 2001, 312 million returns will be filed, of which 30% will be filed electronically. As a result of an ambitious tax-system-modernization effort based on imaging, the IRS hopes to capture 100% of the data from tax returns and at the same time reduce the processing time to two to three weeks. Figures also indicate that although the percentage of electronic filings is increasing, paper returns will continue to be a dominant factor for the foreseeable future.

In this paper, we describe an image-based forms-processing system that significantly reduces the human

[1] IBM Charlotte internal study report on data entry costs, 1993.

0018-8646/96/$5.00 © 1996 IBM

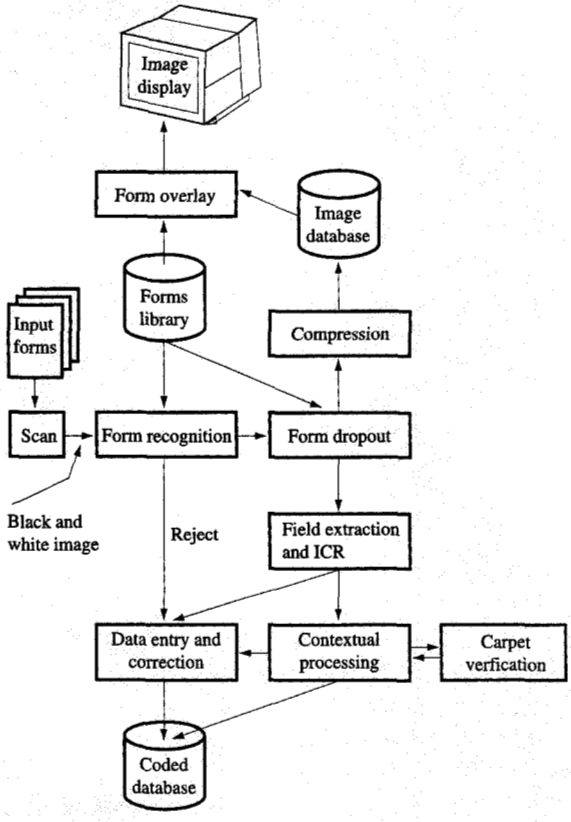

Figure 1

System organization.

labor involved in capturing information from paper forms. After presenting an overview of the system in the next section, we describe its individual components in the succeeding sections. In addition to the components described in this paper, an image-based forms-processing system also requires a workflow manager to route forms that are being processed, a database manager to store and retrieve forms, and a folder manager to organize and present related forms. However, we do not describe these components, because they are outside the main scope of the paper.

System overview

In this section, we present an overview of the image-based forms-processing system. **Figure 1** shows the system organization. A paper form is scanned to produce a black-and-white image. The next task is to identify the form type—for example, in a tax application, to determine whether the image corresponds to Form 1040 or Schedule A, etc. This is done in the "form-recognition" stage. All distinct form types that are used in an enterprise are defined to the system in a "forms-training" phase (not shown in the figure). In the form-recognition stage, the input image is recognized as belonging to one of the previously defined form types. Once the form type is identified, the locations of the fields of interest and the contextual relationships within and between fields are available to the system from the information provided in forms training. If the form type cannot be identified automatically, the data must be keyed in manually. Therefore, reliable form-type recognition is a key requirement in an image-based forms-processing system.

The next task is to extract images of fields that are to be recognized. The input image is carefully registered (i.e., aligned) with its matching blank form (the *template* image), which is stored in the system during the forms-training phase. The template image is then subtracted from the registered input image, leaving only the filled-in data. This is done in the "form-dropout" stage. This process helps in compressing the image significantly, since the dropped-out image has far fewer black pixels than the original filled-in image. The dropped-out image is stored in an image database for display and archival purposes. Notice that an equivalent of the original image can be displayed by overlaying the dropped-out image with the template image.

Images of fields of interest are then extracted from the dropped-out image, and intelligent character recognition (ICR) is applied to them. Contextual information, such as whether a field is numeric-only or alphanumeric, is utilized in ICR whenever such information is available. If no information is available on the case (upper or lower) and type (e.g., numerical) of the field, a two-stage recognition is performed, in which the first stage identifies the case and type, and the second stage establishes individual character identities. Since ICR is not error-free, multiple hypotheses (three for each character, in our implementation) are generated and passed to the contextual-checking stage. Our contextual processor checks the syntactic and semantic correctness of the ICR results, using previously defined constraints. If the contextual processor can resolve the field identity unambiguously, the result for that field is accepted as final; otherwise, the results are passed to the verification stage. For rapid verification, we employ a technique called "carpet" verification, which is explained in the section on verification and correction. Those characters whose ICR identities are verified as correct by the operator are accepted as final. Given these confirmed character identities, the contextual processor tries again to resolve those fields that it could not identify before. The output of the verification and correction process should be close to 100% correct.

Tests on customers' forms show that our system can achieve a data entry rate of about 20000 characters per hour per operator, even with unskilled operators. In contrast, a professional key entry operator can enter about 10000 characters per hour from forms. Therefore, the use of our system provides a significant increase in productivity in the key entry process, while at the same time not requiring the use of professional key entry operators.

Form recognition and dropout

A fundamental problem that must be solved for many form-processing applications is that of efficient image compression. With standard compression algorithms for black-and-white images, such as the CCITT Group 4 MMR algorithm [2], a compression factor between 5 and 20 can be achieved for most types of forms. A typical filled-in form of letter size (8.5 × 11 in.) compressed via the MMR algorithm may require approximately 20-80 KB (kilobytes) of memory. However, there is still a substantial amount of redundancy in these forms, since only the filled-in information is of importance, while the constant information (the printed matter on the blank form) may be stored once and later used for all of the filled-in forms of the same type.

For numerous applications, it is essential to increase the compression ratio quite significantly in order to reduce the required amount of both short- and long-term storage. Fortunately, in many of these applications, the number of different form types is relatively small, while for each form type (and its template) there are an enormous number of filled-in forms that differ only in their filled-in information. This fact gives rise to a more sophisticated compression scheme, which relies on the removal of the common information (the template image) shared by all forms of the same type. After removal of the common data, one is left with an image consisting of the filled-in information only, which requires an order of magnitude less storage than the original image when compressed (via standard G4 MMR compression).

As mentioned above, the procedure that removes the common data from the form and leaves only the filled-in data is called *form dropout (FDO)*. When the FDO algorithm is applied to a filled-in form and the resultant filled-in data are then compressed by a standard MMR algorithm, a typical filled-in letter-sized form may be represented by approximately 2–12 KB, yielding an additional compression factor of nearly 10 with respect to MMR carried out on the original image. A further compression factor of 2 can be achieved by a clever subsampling algorithm (described below) that may follow the FDO. The remainder of this section describes these compression techniques and associated procedures.

A prerequisite for FDO is the knowledge of the form type associated with the given filled-in form. In most applications, multiple (from a few to several hundred) different form types are used; thus, the FDO procedure should typically be preceded by a form-recognition procedure that automatically identifies the type of the input filled-in form from a prestored template library.

One possible alternative to FDO is to print the blank forms with a special color of ink (dropout ink) that is invisible to conventional scanners. In such a case, the template data would be invisible to the scanner, and only the filled-in data would enter the computer. For most systems, however, subsequent image processing is required, such as form recognition, image registration and alignment, de-skew, indexing, and ICR. Using dropout ink makes it almost impossible to carry out many of these tasks. Also, if the application allows usage of photocopies, the technique of using dropout ink for the originals will not apply to the photocopies. In addition, the necessity of form redesign and the use of special ink might be cumbersome and costly to some users. For tasks such as form recognition and image registration, it is best to have "rich" forms with as many "identifiers" as possible (such as lines, boxes, and special marks). Hence, the optimal strategy is to find an appropriate trade-off between adding identifiers to the forms to help with image processing, and eliminating them to help with FDO. Accordingly, a general-purpose, software-based FDO procedure is of significant value for most image applications.

The basic idea of FDO is quite simple. First, in the *forms-training process*, each different blank form is scanned and stored in the system. The fields of interest on which ICR is to be applied are also specified. At the end of the forms-training process, a library of blank forms (called the *template library*) is made available. Then, for each filled-in form to be compressed, the appropriate template form must be recognized, aligned with the filled-in forms (*image registration*), and subtracted from the filled-in form, leaving only the filled-in data, which can then be compressed. The image reconstruction can be carried out by decompressing the "subtracted" form and superimposing the appropriate template data on it.

In **Figure 2**, we show a filled-in version of a tax form (the filled-in data are hypothetical), and in **Figure 3**, the image after the dropout process has been applied. Notice that all of the template data have been removed. Notice also the significant reduction in required storage that we are able to achieve (following compression, from 104.9 KB of storage for the complete image to 10.9 KB after FDO).

- *Form recognition*

For applications with multiple forms, automatic recognition of the type of the filled-in forms is a prerequisite for FDO. This process is based on matching

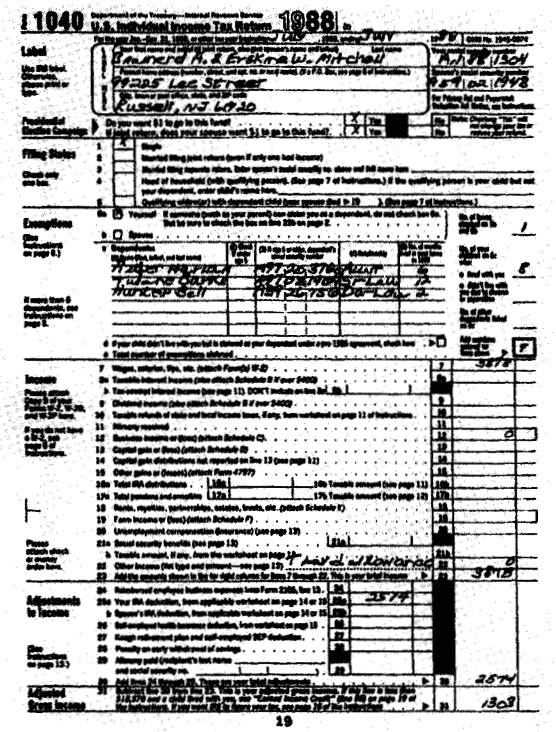

Figure 2

A U.S. tax form with imaginary data filled in. It requires 104.9 kilobytes of storage.

the input filled-in form with a library of prescanned blank forms to find the best match (described later). Another task carried out by the form-recognition module during the matching process is to find the proper transformation between the input filled-in form and its matching template form. This transformation includes translation, rotation, and scaling. Form recognition is also supplemented by a forms-training process, by which new forms are defined to the system.

In most cases, it is sufficient to identify a form by its "signature," which includes the locations and lengths of its horizontal and/or vertical lines. The match between the signature of a filled-in form and a template form from the template library is then carried out by comparing these two sets of signatures by means of an elastic matching procedure [3].

Occasionally, however, this signature is not sufficient to carry out the form recognition, because either the form has too few lines, or different forms in the library have similar signatures. Hence, the elastic matching procedure may match two or more templates with the same input form, leading to ambiguous recognition. In these cases, *user-defined fields* (UDFs) may be specified, in which some other matching function (such as ICR, bar-code reading, or histogram matching) may be used. These UDFs are defined by an operator during the forms-training process, when the system indicates that an input form is too similar to a previously stored form. Then, the form-recognition procedure uses these UDFs to discriminate between similar forms. The type of matching function defined in each UDF depends on the application. For example, if there are two similar tax forms, one for 1994 and the other for 1995, that differ only in the field where the year is specified in some known font, a UDF should be defined for this field, and an ICR function tailored to this font should be used to identify the form by recognizing the year printed in this UDF.

It should be noted that forms of the same type that come from different printing sources differ slightly in their final layout, and forms that have even slightly different images are inadequate for the subsequent FDO. Hence, each different layout and version of the same type of form is defined as a different form type for the automated system. A special feature of the form-recognition software is the ability to detect forms scanned upside down (i.e., rotated by 180°) by rotating the image and then recognizing the form type.

The form-recognition module described here has been successfully used in census, banking, and tax-processing applications, where tens to hundreds of different forms were used.

• *Registration*

The first problem that must be addressed after recognizing the form type is image registration, i.e., the process of aligning an incoming filled-in form with a matching template. The geometric relationship between an instance of a form and its template can be defined by a transformation that is composed of translation, rotation, scaling, and shear. In general, a transformation that can define the image of a form in terms of the template is nonlinear and requires the evaluation of a large number of parameters. The solution to the registration problem was thus divided into two phases:

1. *Coarse registration* Computes and performs a linear transformation containing translation, rotation, and scaling.
2. *Fine registration* Fixes local nonlinear distortions by using a piecewise-linear pattern matching of rectangular blocks in the images.

Coarse registration

The task of coarse registration is to perform global rotation, scaling, and translation, according to five

parameters defining the geometric difference between an incoming form and its template. Since these operations are computationally intensive (especially the rotation of an image), they are performed only if necessary, i.e., if the distortion is larger than some tolerance threshold. The transformation parameters are provided by the form-recognition process.

Fine registration

When coarse registration is complete, we are left with an image that is different from the template by slight rotation, scaling, and localized distortion. The transformation parameters describing the relation of the image to the template are also available. The next step is to perform a fine registration to deal with the localized nonlinear distortions, rotation, and scaling (because of nonlinear distortions in the scanning operation). There are two possible mechanisms to overcome these problems:

- *Block registration* If this mechanism is chosen, the image is partitioned into a number of relatively small, overlapping segments. For each segment, the system computes an independent transformation, which consists of translation only. Since the entire image is composed of a combination of shifted segments, registration of slightly rotated and scaled images is possible. Moreover, since each segment may move independently of its neighbors to some limited extent, the local nonlinear distortions are addressed as well. In practice, the location of each segment is computed independently of its neighbors; therefore, irregularities in segment placing may arise. Such irregularities may produce very annoying results in some cases; for this reason, a heuristic control mechanism is applied to make sure that neighboring segments do not differ in movement by more than a specified number of pixels, thus reducing such irregularities to a minimum. Once the optimal translation parameters are established, each segment is placed at the appropriate location of the output image.
- *Elastic registration* Block registration may cause annoying irregularities in the FDO result, especially when a scanner with an irregular form-feed drive (which is the primary cause of nonlinear distortions) is used. In such cases, it may be better to apply a more complex registration mechanism, which leads to better quality, at the expense of speed of execution. Our dropout software provides an elastic registration mechanism for such cases, which tries to match each pixel in the input image to the respective pixel in the template image, stretching and shrinking the input image as it progresses.

- *Subtraction*

After registration, the exact location of the template data in the form is known. Actually, the output of the registration phase can be described as an image composed of a "noisy" version of the template, and the filled-in information. Once the location of the template data is known, it would seem logical to remove the data for FDO; however, it is imperative to take into account the fact that even for the same form sheet, two separate scans will provide significantly different output. Hence, even after registration, there will be differences between the template data as they appear on the template image and the filled-in form image. Therefore, a simple pixel-by-pixel subtraction would be ineffective, since we would be left with noise concentrated around the template-data locations.

This problem can be resolved by a decision process which in effect classifies each pixel in the form as being part of the template data or the filled-in data on the basis of the nature of the pixel's immediate neighborhood [4]. More specifically, if a pixel in the transformed image is white, the output value for the pixel is also white. When

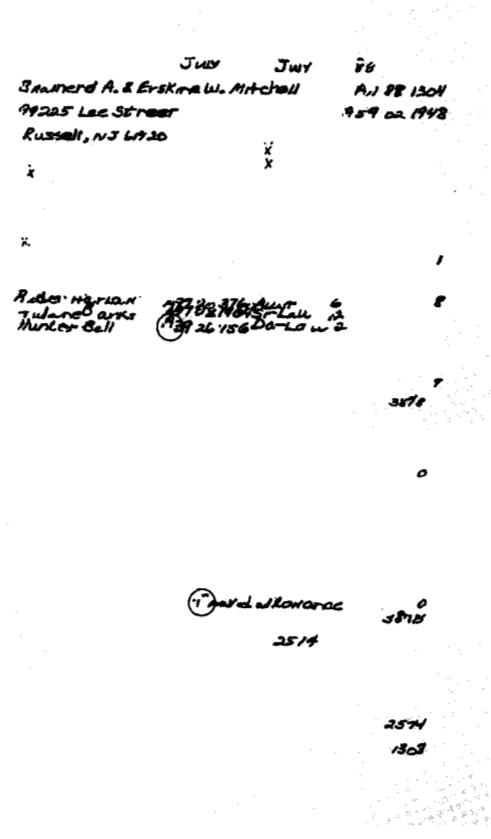

Figure 3

Output image from the form-dropout process for the form of Figure 2. It requires only 10.9 kilobytes of storage. Note that gap-filling corrections are effective in many places but occasionally do not work ideally (e.g., the two circled spots).

both the pixel and the corresponding pixel in the template image are black, the output is white. In the case where a pixel is black and the template pixel in this location is white, the pixel's immediate neighborhood is examined, and the decision made about the output value in accordance with the nature of the pixel's neighborhood.

This implementation does not perform well in two cases, for which the following enhancements are available:

1. A major assumption of the system is that filled-in information is additive in nature; i.e., information is added on a blank form. Therefore, the system detects and filters out only information that has been written upon the form. However, in some instances, data of the blank form may be deleted from the filled-in form by means of a sticker or a label, or by using some sort of white-out material. In those parts where data of the blank form are erased from the filled-in form, there will be no data to drop out. Hence, when the form is reconstructed by superimposing the template on the subtracted form, the erased material shows up again on the reconstructed form, which might be an undesirable result. To overcome this problem, the FDO program contains white-out-detection-and-correction capabilities that locate such areas on the filled-in form. White-out support provides the ability to correctly treat template data that are removed from the input form, and results in a perfect reconstruction. This feature is optional and requires additional processing time.
2. FDO does not perform well in areas with dense nonwhite background textures. If data are written on such areas, it is possible that FDO may fail to reconstruct these data correctly; in extreme cases, the data will be lost. To overcome this problem, form dropout can activate *dense-area processing*, which is able to detect filled-in data located in such areas and filter them correctly. This feature is also optional and requires additional processing time when activated.

   A related problem with FDO is that it introduces gaps in the filled-in data that cross the template data. When the template pixels are subtracted from the image, they are also removed from filled-in data marked on them, resulting in gaps in the filled-in data. For form-reconstruction purposes, this is not a problem, since the template data fill these gaps, resulting in an image with no distinguishable differences; however, it can cause trouble for ICR, which is applied on the data after FDO is performed.

   Therefore, FDO includes *gap-filling* support, which is applied to the dropout result in locations where filled-in data are expected, according to entry-field-definition information. This feature is also optional and requires additional processing time. The gap-filling procedure is implemented by matching black runs (series of black pixels with no intervening white pixels) in the dropout image that are close enough to the template lines. A connect or no-connect decision is made for neighboring black runs, including runs that lie on opposite sides of the template line and those that lie on the same side of the template line. The decision is based on geometrical parameters such as the distance between runs, the slant of the corresponding strokes, and connectivity information in the vicinity of the runs. These considerations are carefully weighted to yield a satisfactory system response.

- *Lossy compression*

The result of the subtraction phase is an image containing only the filled-in information of the original form. For many applications, it suffices to have legible text, which is indistinguishable from the original by the viewer. In these cases, a lossy compression may be applied to further increase the compression ratio by approximately a factor of 2. A trivial approach to this would be to reduce the resolution of the scanning. Indeed, some scanner and facsimile devices do allow this option. However, while this solution might be valid for some images (such as scans of handwritten text with large characters), it would cause unacceptable distortion for other images (such as scans of small, printed characters). Moreover, even for medium-sized text, simple reduction of resolution would cause image quality deterioration that might be quite annoying to the observer.

A lossy compression technique is proposed by which the compression ratio is increased to about twice that of the subtracted form, yet with high image quality. In general, this technique is based on a special subsampling of the subtracted image, with a goal of reducing the resolution by a factor of 2 in both horizontal and vertical directions. The image is partitioned into 2×2 blocks of pixels, and a single pixel is used to represent each such block. For blocks containing zero or one black pixel, the representative pixel is white. For blocks containing three or four black pixels, its value is black. For blocks containing two black pixels, the color of the representative pixel depends on the immediate neighborhood of the block, according to connectivity criteria designed so that narrow lines do not disappear and disjoint lines do not merge [4]. When the subsampled image is compressed, an additional compression ratio of approximately 2 is achieved.

- *Form reconstruction*

The output of FDO is a compressed-image file containing the filled-in data of a form, the name of the template form, and flags related to the FDO process. Upon retrieval of a form from the archive, it is necessary to invoke the reconstruction task, which is responsible for

generating an image of the form that looks exactly like the original scanned paper. Reconstruction uses the information in the compressed-image file to determine what template has to be retrieved; the template data are then superimposed on the decompressed subtracted image (bitwise-OR operation). If the form was processed with the lossy compression option, the reconstruction process also approximately reverses the subsampling operation in order to match the image resolution to that of the template image. The reconstruction is fast, since the subtracted image and its template are already aligned.

Intelligent character recognition

ICR[2] is the process by which bitmaps within fields of a form are converted to character codes (ASCII or EBCDIC). For a review of ICR research, see [5]. Before ICR can be applied to data of a field, we must perform the following preprocessing:

1. Correct skew. Although the global skew of the input form is usually corrected in the FDO process, local skew (due to peculiarities in handwriting or printing) of the field data may still exist.
2. Locate the field data. This is done using the field coordinates defined at the forms-training stage. A certain degree of vertical and horizontal extension of the field may sometimes be necessary to handle writing that goes beyond the field boundaries.
3. Detect the baseline (the imaginary line over which the characters are written or printed). The character image reading order and size normalization are based on the baseline.
4. Detect spaces between words.
5. Identify connected components (i.e., character fragments, individual characters, or touching characters), and merge neighboring connected components that satisfy certain geometrical constraints.

The input to ICR is a bitmap corresponding to a string of characters; this string may contain touching characters, which are usually difficult to segment and hence recognize. Our ICR employs a *recognition-based* segmentation scheme, described below, for separating touching characters. Using a set of heuristics, it first searches for possible split points. Then, it selects the best sequence of splits on the basis of the recognition results on the individual sequences. For each connected component, ICR returns up to three sequences of splits from the graph solver in the recognition-based segmentation algorithm. Each split in these sequences is again associated with

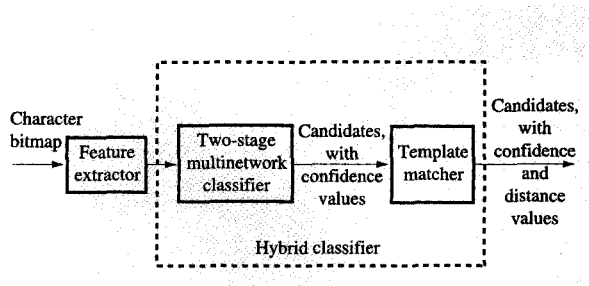

Figure 4

The recognition component for intelligent character recognition.

three character candidates. Moreover, a confidence measure and a distance measure are associated with each character candidate. These alternative splits and character candidates, together with their confidence and distance measures, provide our contextual processor with useful information to resolve the ambiguity and to make corrections.

ICR treats machine-printed and hand-printed text differently. Because the characteristics of machine-printed text and hand-printed text are rather different, ICR employs different classification schemes. In the following presentation, we discuss primarily the ICR system for hand-printed text; however, the variations used for recognizing machine-printed text are pointed out wherever necessary. We first describe the recognition component in ICR and then present the recognition-based segmentation scheme.

• *Recognition of isolated characters*

Figure 4 is a diagram of the recognition component in ICR. It consists of a feature extractor (described below) from the input bitmap and a hybrid classifier, which employs a two-stage multinetwork (TSMN) classifier [a classifier consisting of two stages, each of which contains several neural networks (see **Figure 5**)] and a template matcher (TM) (which matches the input patterns against a subset of stored templates). For machine-printed ICR, the TSMN classifier is replaced by a single feed-forward network with 78 output categories (10 digits, 26 uppercase characters, 26 lowercase characters, and 16 special symbols). This is because machine-printed-character recognition is a relatively easy problem compared to hand-printed-character recognition.

Features

The goal of feature extraction is to extract the most relevant measurements (features) from the character

[2] ICR is also known as optical character recognition (OCR); however, because of the increased sophistication of today's recognition algorithms and the range of input they can handle, such as handwritten data and omnifont machine-print, the industry trend is to use the term ICR.

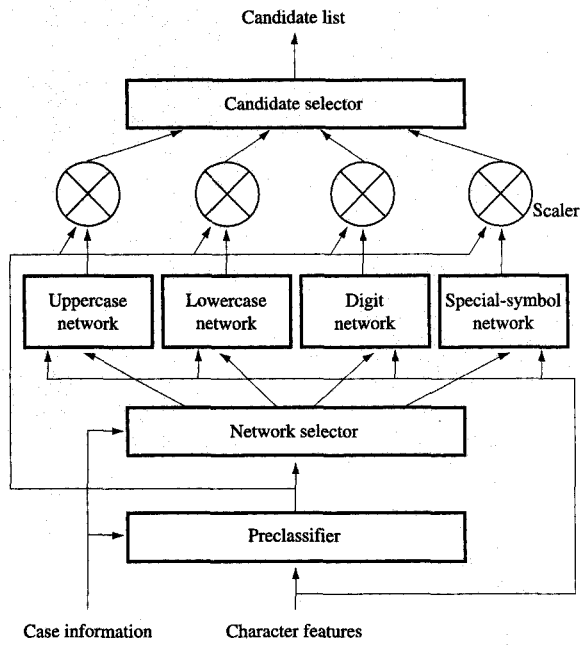

Figure 5

Two-stage multinetwork classifier.

bitmaps, so as to minimize the within-category variability while increasing the between-category variability. Two different types of features [6] are used in our system: contour-direction features and bending-point features.

Prior to feature extraction, character images are normalized to 24 (pixels in height) × 16 (pixels in width) in order to reduce the character-size variation. One can efficiently extract contour-direction features by scanning the normalized 24 × 16 image by a 2 × 2 mask to detect primitive patterns. A primitive pattern is detected if the number of black pixels in the mask is from one to three (neither all white nor all black). Each primitive pattern is classified into one of four types. The 24 × 16 image is sliced in four directions: horizontal, vertical, diagonal (45°), and off-diagonal (135°). Each slice has a width of four pixels; therefore, the vertical direction has four slices, and the others have six slices (22 slices in total). A contour-direction feature is defined as the number of primitive patterns of the same type in each slice. This results in an 88-dimensional (22 slices × 4 types) feature vector.

Bending-point features include some topological characteristics of a character, such as high-curvature points, terminal points, and fork points, which are detected in the character image by tracing the contours of strokes. A special geometrical mapping from bending points and their attributes (e.g., acuteness, position, orientation, and convexity or concavity) to a fixed-length (96-element) feature vector has been designed [6]. The normalized image is evenly divided into 12 regions (6 × 6 or 6 × 5 pixels). The bending points in a normalized image are coded according to the regions in which they occur and according to their curvature orientations, which are divided into eight cases (four orientations, each of which is either convex or concave). The value of acuteness of a bending point is used as the magnitude for the corresponding component in the feature vector.

For hand-printed-character recognition, all of the 88 contour-direction and 96 bending-point features are used, while for machine-printed-character recognition, only the 88 contour-direction features are used. However, the normalized character height, width, and position (distance between the center of the character and the baseline) are used as three additional features for recognizing machine-printed characters. These latter features provide useful information for distinguishing some lowercase and uppercase characters that are otherwise difficult to differentiate (e.g., c/C, k/K, o/O). We do not utilize these three features for hand-printed characters, because they are not reliable for distinguishing such characters.

Two-stage multinetwork classifier

The concept of a TSMN classifier is based on the following observations. For some fields on a form, the target character set to be recognized can be determined *a priori* (e.g., for zip code and social security number fields, the target character set is limited to the 10 digits); therefore, a specialized network can be designed for this task. It is well known that the smaller the character set for which a network is trained, the better the performance the network can achieve for the specific task (in accuracy, speed, and confidence measure, which is used for a rejection operation and postprocessing). Unfortunately, for many other fields (e.g., street address and account number), it is not so easy to constrain the target character set *a priori*. In such cases, two methods are often used. One is to invoke all of the necessary specialized networks (networks that are trained to recognize subsets of the characters) and select the winning character(s) among all of the characters chosen by the invoked networks. Obviously, this is not a reliable method, because all of the specialized networks are trained separately, without any competition among them. It has been observed that feed-forward networks with sigmoid functions [7] often generate high output values for characters from categories that are not present in the training data. This undesirable property causes this method to behave very poorly in recognition accuracy and confidence measure. The method also suffers from low recognition speed, since multiple

networks must be invoked. An alternative method is to design a single large feed-forward network with a sufficient number of "output units" to cover all of the possible target categories (each output unit represents a target character category). This method works reasonably well in the situations where no case information is available, but not as well as the specialized networks in known-case situations. Moreover, a large single network makes the training process difficult.

Thus, neither method achieves a satisfactory balance among recognition accuracy, confidence, speed, and flexibility. To solve this problem, we propose the TSMN classifier shown in Figure 5. It consists of a bank of specialized networks, each of which is designed to recognize a subset of the entire character set. A preclassifier and a network selector are employed for selectively invoking the necessary specialized networks. The network selector makes decisions based on both the case information obtained from the field definition and the outputs of the preclassifier. Compared with a system that uses either a single network or one-stage multiple networks, the TSMN system offers advantages in recognition accuracy, confidence measure, speed, and flexibility [8].

Partition of target character set A natural partition of the target character set (output categories) into uppercase, lowercase, digit, and special-symbol subsets is used. This partition is appropriate for many applications, such as forms processing. The special-symbol subset contains 16 punctuation symbols that are sufficient for our application (! # $ % * ( ) + < > " : ; / = ?). The comma and period are handled separately in our system.

Preclassifier The preclassifier has four output units (each of which corresponds to one of the four character subsets), 40 hidden units, and 184 input nodes. The standard sigmoid function is used.

Specialized networks The uppercase network and lowercase network have the same architecture, with 50 hidden units and 26 output units. The digit network has 40 hidden units and 10 output units. The special-symbol network has 40 hidden units and 16 output units. These network architectures were selected after numerous trials. The standard sigmoid function is used in all of the specialized networks.

Network selector Since we are interested in only a few most likely candidates for the classification purpose, it is not necessary to invoke the specialized networks for which the output values from the preclassifier are low. This does not affect the recognition accuracy, because categories from these specialized networks are unlikely to be among the top candidates; however, this selection operation significantly reduces the computational requirement.

The decision logic for network selection utilizes both the prior-case information (when available) and the output values of the preclassifier. The decision logic for network selection uses the following set of rules:

1. If the prior-case information can uniquely determine which specialized network to invoke, the preclassifier is bypassed.
2. The ith network is selected by the preclassifier if and only if the ith output value of the preclassifier (normalized by the maximum output) is greater than a prespecified threshold which controls the number of specialized networks that are invoked.
3. The specialized networks that are selected by the preclassifier and identified on the basis of prior-case information are invoked. If no specialized network is selected, the input character is rejected. (This might occur, e.g., if a number were written in an alphabetic field.)

Scaling the specialized networks The outputs of the specialized network are multiplied by the corresponding output values of the preclassifier.

Candidate selector The three categories corresponding to the three largest scaled output values are selected as candidate characters. The scaled output values are associated with the candidates as confidence values.

Template-matching (TM) classifier
The template-matching (TM) or nearest-neighbor classifier [6, 9] is well known and is commonly used in statistical pattern recognition. It is a nonparametric classifier that makes no assumption about the underlying pattern distributions. The TM decision rule compares an input pattern with a collection of stored patterns (templates) and assigns the input pattern to the category associated with the nearest neighbor among the stored patterns.

A severe drawback of the TM classifier is that it requires a large amount of computation and storage when a large number of training patterns are involved. Because of this computational burden, the TM classifier is not very popular when real-time requirements must be met. In the domain of character recognition, a large number of training patterns are often available; therefore, applying the basic TM classifier is not attractive. In [10], a technique was developed for grouping similar training patterns from the same category (such as "a") into clusters, based on a similarity measure that is the city-block distance (between a pattern and a cluster center in the feature space) divided by the radius (standard

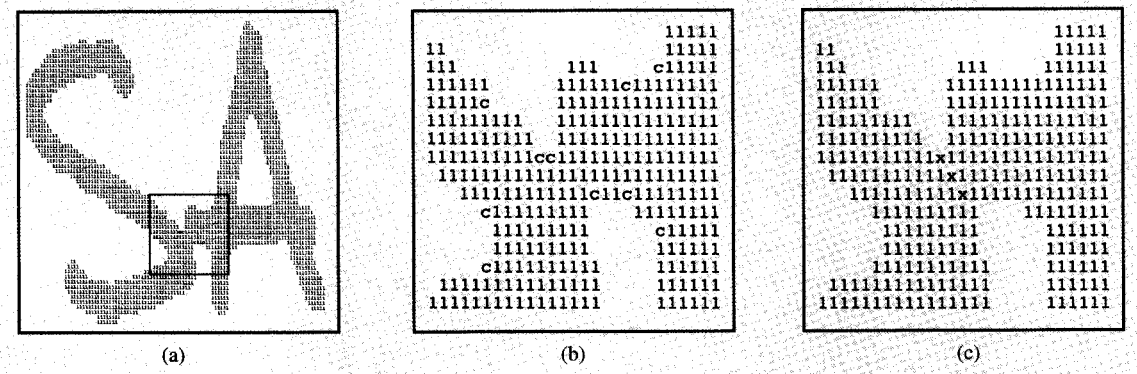

Figure 6

Splitting touching hand-printed characters: (a) Letters "S" and "A," with black pixels marked as "1"s and candidate split points marked as "c"s; (b) enlargement of area marked in (a); (c) enlargement of marked area, with split points marked with "x"s.

deviation with respect to the cluster center) of the cluster. More than one cluster may be created for each category.

We first apply a *k*-means type of clustering algorithm [11], with the city-block distance metric, to all patterns within a category, independent of other categories (for example, to all training patterns of "a," independent of patterns of other lowercase letters). The initial clusters are randomly generated. If the distance between a pattern and its nearest cluster center is within a predetermined threshold, the input pattern is added to that cluster. If not, a new cluster is created. Clusters with variance (with respect to the cluster center) larger than a specified threshold are split into smaller clusters until the variance constraint is satisfied. The mean vector of a cluster is chosen as the prototype for that cluster. At the end of this process, we have one or more clusters for each category and a library of prototypes, one for each cluster. We then eliminate clusters that have too few patterns. The radii of clusters, which are initialized to the same value, are fine-tuned as follows. We first classify all of the training patterns against the prototype library, using the nearest-prototype criterion. If all of the patterns in a cluster are correctly classified, no change is made to the radius of the cluster. Otherwise, the radius is shrunk by a small amount for the next iteration. As the radii are tuned, the classification results improve. This process is continued until there is no more improvement to the classification accuracy.

The clustering algorithm produces a set of prototypes. The TM classifier assigns an input pattern to the category associated with the best matching prototype. The best matching distance is also obtained and associated with the classification.

Hybrid classifier

It has been found by many researchers that combinations of different feature sets (a set of measurements that represent certain characteristics of a character) and different classifiers can compensate for each other's weakness, thus improving the classification performance [12, 13]. The hybrid classifier that we have developed is based on the following observations: 1) In general, the probability that the correct answer is among the top three candidates (coverage rate) is higher (by 2% to 8%) than the probability that the first choice is correct; 2) the TSMN classifier outperforms the TM classifier by 2% to 4% [14]; and 3) it is inefficient for the TM classifier to search for the best matches among all of the categories. Since the coverage rate of the top three candidates is very high (98%) for the TSMN classifier, focusing on the top candidates proposed by the TSMN classifier is more efficient. These observations led us to pursue a method of reordering the top three choices produced by the TSMN classifier, using the TM classifier when the TSMN classifier has low confidence.

The hybrid classifier works as follows. First, the TSMN classifier identifies the top three candidates. Next, the TM is invoked to match the input pattern with only those templates in the three categories selected by the TSMN classifier. The best template-matching distance in each category is determined. If the difference between the confidence values of the top two candidates calculated by

the TSMN classifier is less than a prespecified threshold, they are reordered according to the best template-matching distances calculated by the TM. The same rule is then applied to the second and third choices. Thus, template-matching distances are used to reorder choices only if the TSMN is not "sure" about its decision. Note that the TM is invoked in any case, in order to obtain the template-matching distance for each of the three candidates, which is used as a length in the graph solver in the segmentation algorithm. This hybrid classifier has little extra computational overhead with respect to the TSMN classifier, since only the top three categories are matched for the best template distances.

• *Recognition-based segmentation*

Segmentation of touching characters has long been known as a critical problem in ICR. A number of approaches have been proposed in the literature. An excellent survey of strategies in character segmentation was provided in [15]. Our ICR system employs a recognition-based segmentation scheme [16]. This scheme consists of the following phases: 1) Identifying potential split points; 2) constructing a graph; and 3) finding the shortest path in the graph from the leftmost node to the rightmost node. We now discuss these three phases in detail.

Identifying potential split points

Touching characters in machine-printed text occur because of thick printing, insufficient scanner resolution, or poor binarization. For hand-printed text, contact between adjacent characters may be due to the style of writing as well as to crowding of symbols. In some cases, a stroke belonging to one character may traverse a stroke belonging to another. While relatively simple means can be devised to separate touching machine-printed characters, we have developed one method for separating touching characters that works for both machine-printed and hand-printed text. We now describe this method for identifying potential split points.

Typically, a "clue" is available to detect the point of penetration. The clue may be described as a concavity at the point of contact. This suggests that when merged characters are suspected, the segmenter should seek such concavities in the most likely parts of the image. Our algorithm, using a contour-following process to provide edge-direction data for estimating local curvature, examines the middle section of the pattern for such occurrences.

Figure 6(a) shows touching hand-printed letters "S" and "A." The "1"s represent black pixels. The "c"s mark points located at portions of the outer contour that are locally concave and have nonvertical direction. The marked section of Figure 6(a) enlarged in **Figure 6(b)** contains all of the "c"s (candidate split points). These points are subjected to additional tests in order to choose likely split points. The points are searched in order of an estimate of local curvature, and a split point is defined if it is within a specified distance from a point of concavity on the opposite side of the same contour. The original pattern is broken along a line between these two points. If a resultant subpattern would be too "small," we ignore the particular split points. In **Figure 6(c)**, the proper separation between "S" and "A" is indicated by "x"s, according to this method. Other points of concavity are filled in with "1"s.

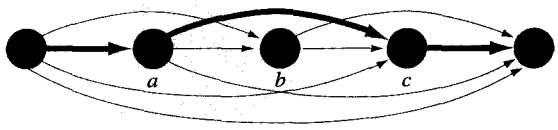

Figure 7

Segmentation graph for an image with three potential split points (*a*, *b*, and *c*), showing all possible edges (subimages). The heavy path represents the segmentation consisting of the following three subimages: (1) from the left side of the image to split point *a*, (2) from *a* to *c*, and (3) from *c* to the right side of the image.

Constructing a segmentation graph

The potential split points are ordered according to their horizontal locations. Each neighboring pair of split points defines an elementary segment of the image. Neighboring triples define larger segments, and so on. A directed graph (segmentation graph) is constructed, as shown in **Figure 7**. The interior nodes of the graph represent the split points, while the first and last nodes represent the left and right sides of the image, respectively. All of the edges in the graph are directed from left to right. Each edge represents the subimage (segment) defined by the two split points that the edge connects. A segmentation of the image can be described by a path from the leftmost node to the rightmost node, with the nodes on the path being the final split points. The number of characters is the number of interior nodes on the path plus one.

If we assign a length to each edge in the graph, the length of a path is taken to be the sum of the lengths of the edges along the path. The optimal path can then be defined as the shortest path. An edge corresponding to a segment wider than some specified amount can be assigned an infinite length, which allows the edge to be

Table 1 Results of ICR and contextual processor on the tax form of Figure 2.

Field name	*ICR results*	*Contextual processor results*
YEAR_BEGINNING	JULY	JULY
YEAR_ENDING	JUIY	JULY
YEAR	88	88
TAX_PAYER_NAME	BRwnerd A. & Erskine W. mrtcheN	BERNERD A. ERSKINE W. MITCHEM
SS_NUMBER	A11 88 1304	A11 88 1304
STREET_ADDRESS	99225 uC StreeT	99225 LEE STREET
SPOUSE_SS_NUM	As9 a2 19y8	A59 22 1948
CITY	RusseU	RUSSELL
STATE	NJ	NJ
ZIP_CODE	61920	61920
EXEMPTIONS	1	1
DEPENDENT_1	BadLoi mgrLan	Barlow Marlan
DEPENDENT_1_SSN	1918203764	A18 20 3764
...	...	...

removed from the graph (provided that continuity is maintained from the leftmost node to the rightmost node).

Since our classifier provides a confidence and a distance measure for the hypothesis of identity of the input segment, these values can be used for assigning lengths to the edges in the graph. Let c_{ij} $(0 \leq c_{ij} < 100)$ and d_{ij} $(0 \leq d_{ij} < 256)$, where the values 100 and 256 are arbitrary maxima, be the confidence and distance values returned by the classifier for the subimage between split points i and j. We assign a length of $(100 - c_{ij})d_{ij}$ to the edge that connects nodes i and j. Note that the larger the confidence values and the smaller the distance value, the shorter this length will be. A perfect character subimage will have zero distance from the template matcher and confidence value 99 (the maximum confidence value) from the TSMN classifier, resulting in a zero length for the corresponding edge.

Finding the shortest path

A dynamic programming technique is used for finding the shortest path from the leftmost to the rightmost node. The algorithm is modified to generate the three shortest paths, so that our contextual processor can take advantage of these alternative paths.

- *Recognition performance of the ICR component*

The performance of ICR varies with the quality of the image data and writing style. For isolated, machine-printed, multifont characters with moderate noise, the recognition rates of ICR on numeric, uppercase, lowercase, and special symbols exceed 99%. For constrained, hand-printed characters—e.g., the NIST (National Institute of Standards and Technology) Special Databases 3 and 7 [17]—ICR achieves a recognition accuracy of 98% for numerics, 96% for uppercase, and 95% for lowercase (databases 3 and 7 were mixed, and then redivided into a training set and a test set). In May 1992, NIST held a conference in which 44 different hand-print OCR systems were considered. This conference tested the ability of these systems to recognize presegmented, hand-printed characters. The performance of our ICR system was among the top three [17]. The overall conclusion from the NIST study was that the state of the art of machine recognition of discrete hand-printed characters is as good as or better than human readers; however, for nonsegmented cursive fields (with some hand-printed fields), the recognition rate at the character level without contextual processing drops appreciably. In **Table 1**, we show the results from our ICR system when it is applied to the form shown in Figure 2. The table gives the strings corresponding to the highest confidences generated by ICR. ICR actually returns the top three alternative sequences of segments and the associated character hypotheses and confidences for each connected component. Notice that at this stage, where no contextual information has been exploited, a number of errors are made by ICR. Many of the errors are corrected by our contextual processor, making use of higher-level constraints as well as multiple character hypotheses; these results can also be seen in Table 1.

Exploitation of context

Basic ICR software, such as that described above, recognizes digits and letters with reasonable accuracy; still, the recognition of hand-printed words or phrases remains a challenge. In the best case (isolated, unbroken, well-formed digits), the individual recognition rate may be around 98%; however, for a 10-digit telephone number, such performance yields only an 82% correctness for the entire field (assuming independence among characters). For more difficult cases, such as unconstrained, mixed-case characters, the individual character recognition rate is closer to 70%. This would mean that practically every word would contain an error! Obviously, something

more is needed to make the technology viable. First, the exploitation of *a priori* contextual knowledge can increase the recognition accuracy. Second, the recognition system must include a user-friendly manual verification and correction subsystem to complement the automatic process.

In this section, we describe our technique to exploit context, particularly for recognizing data from forms. In a following section, we briefly discuss some ongoing work on verification and correction. What makes automated form-data entry feasible is the fact that tight constraints exist for most fields on a form. For example, the sequences (city, state, zip) for a location or (mm, dd, yy) for a date have strong constraints associated with them, both syntactic (zip has five or nine digits, mm one or two, etc.) and semantic (there exist dictionaries of acceptable combinations). The input of free-format text, on the other hand, is a different problem, requiring sophisticated natural-language processing facilities that are beyond the scope of the system addressed in this paper.

In many of today's systems, the exploitation of context is left to the application program. This represents quite a burden for the application developer, since it may involve sophisticated fuzzy-dictionary look-ups, tree searching, backtracking, etc. Also, the code can and should be reused from application to application. Thus, the right approach is to remove exploitation of context *from* the application, and move it *to* the recognition system.

First, we describe a system architecture for a general-purpose, highly customizable *context analyzer* that allows the application program to specify constraints that the recognized information must satisfy, both syntactic and semantic. Some of these constraints may be quite general, but most are application-dependent. To make the specification of these constraints easier, our system provides a language called *Document Specification Language (DSL)* [18]. A "program" written in DSL specifies the syntax of the information to be recognized, together with routines used to ensure semantic validity. DSL is specialized, concise, and nonprocedural. Once the specification is written in DSL, it can be compiled into an internal representation (the *context structure*).

Figure 8 shows the general architecture of the context analyzer. The process is driven by an application program, which first invokes the character recognition program, the results of which are placed in a buffer. Then, it invokes the context analyzer, which uses the context structure and the character results and produces the best possible results for the field. These are also stored in the buffer. When the process is complete, the application program retrieves the final values from the buffer.

At execution time, the context analyzer behaves like a parser: The recognition program produces tokens, which the context analyzer tries to match with the context

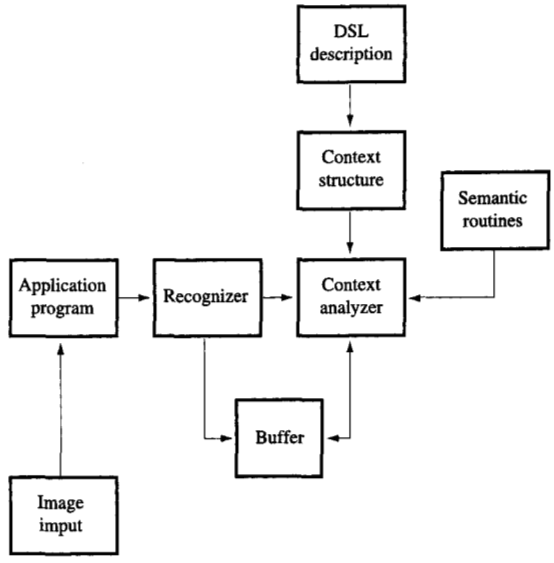

Figure 8

System architecture for contextual processor.

structure while making sure that semantic constraints are not violated. Since there may be several choices for a character, the parsing generally involves backtracking.

- *Document Specification Language (DSL)*

A DSL specification (also called DSL file) provides, for each field of a form, the field name and the field type. In addition, it includes the definitions of these types. Essentially, the definition of a type includes the constraints that apply to a field of that type. A type can be *elementary* or *composite*. For example,

ELEM_TYPE zip . . .

COMP_TYPE phone_nbr . . .
FIELD zip_code, zip
FIELD home_phone, phone_nbr
FIELD work_phone, phone_nbr

Before showing how types are defined, we need to introduce *alphabets*:

- *Basic alphabets*: numeric, uppercase, lowercase, special symbols, and a few others.
- *Defined alphabets*: An alphabet can also be defined in DSL by the inclusion in the DSL file of a record containing the alphabet name and the alphabet

definition (union of previously defined alphabets and/or specified characters).

Here are two examples:

```
ALPHABET digits ["0123456789"]
ALPHABET hexa [digits, "ABCDEF"]
```

Elementary field types There is only one way to write a five-digit zip code such as 95120 or the digits of a telephone area code such as 408. But there are several ways to write a telephone number; for example, (408) 927-3999 and 408 9273999 may both be valid. The strings 95120, 408, 927, and 3999 are instances of certain *elementary field types*, while the phone number is an example of a *composite field type.*

An elementary type is defined in an ELEM_TYPE record in the DSL file. Such a record contains the type name, the keyword PHRASE or WORD (indicating whether or not spaces are acceptable), an alphabet name (to specify the alphabet to which characters in this field must belong), a LENGTH condition (such as LENGTH = 5, or 6 < = LENGTH < 9), and (optionally) the name of a dictionary or routine (this is discussed below).

The following are examples of records defining elementary field types:

```
ELEM_TYPE zip    WORD, digits, LENGTH = 5
ELEM_TYPE area   WORD, digits, LENGTH = 3
ELEM_TYPE prefix WORD, digits, LENGTH = 3
ELEM_TYPE ext    WORD, digits, LENGTH = 4
```

Composite types Each composite type is defined by a record in the DSL file with the name of the type, a list of pairs describing the elements involved in the definition of this composite type (each pair comprises the element name and its type), a list of acceptable representation(s), as illustrated below, and the representation to be used for the result string.

Each representation describes one valid way of combining the elements. It is expressed as a sequence of element names and/or string constants. For example, the definition of a phone number as a composite field type with three elements might look like this ("_" represents a space):

```
TYPE phone a(area), p(prefix), e(ext)
  REP      "( "a")_"p"-" e      e.g., (408) 927-3999
  REP      a"_"p e              e.g., 408 9273999
  OUTPUT "("a")_"p"-" e         e.g., (408) 927-3999
```

The representations (we show only two) are ordered by decreasing likelihood of occurrence. The output representation is the one used for the output string, independently of how the information was initially written. This ensures that the information is always converted to a specified format, ready to be stored in a database or otherwise processed.

Routines and dictionaries
DSL may specify, for an elementary field or a composite field, a routine or dictionary that can be used to improve recognition. In general, routines and dictionaries are application-dependent and are supplied by the user.

Routines Any routine must conform to a simple calling interface. Essentially, arguments specify the type of the elements involved in the constraint. For example,

```
ROUTINE check_area (area).
```

A routine returns a true or false value but may also update the values of some elements in the buffer. If the routine is associated with an elementary type, the corresponding ELEM_TYPE statement contains the name of the routine, as in the following:

```
ELEM_TYPE area WORD, digits, LENGTH = 3,
  ROUTINE (check_area).
```

The routine check_area may test, for example, whether the second digit of the area code is 1 or 0, and return true if the constraint is satisfied, or false otherwise.

If the routine is associated with a composite field type, a clause is added to the definition of the type. For example, the addition of the following clause in the definition of type *phone* could force the routine *check_prefix* to check the validity of the prefix *p* for a particular area code *a*:

```
CHECK check_prefix (a, p).
```

Dictionaries Dictionaries must be defined in DSL by means of a DICTIONARY statement. For example,

```
DICTIONARY fname IN "my_fname"
```

defines a dictionary of first names. The ELEM_TYPE statement for first names would refer to that dictionary in the following way:

```
ELEM_TYPE first_name WORD, . . . , DICT (fname).
```

As in the case of routines, dictionaries can also be applied to composite field types. In fact, in the general form, the dictionary and routine mechanisms of our contextual processor allow constraint checking to involve several elements, even if these elements come from different fields. For example, suppose there exists a dictionary

called "map," containing the valid pairs ⟨zip, area⟩; the CHECK statement

CHECK map (zip_code, home_phone.area)

then specifies that a check be made on the pair (zip_code, home_phone.area), where zip_code is the name of the zip field and home_phone.area is the fully qualified name of the element area in the field home_phone.

• *The context analyzer*

As mentioned earlier, the analyzer must interpret the tokens returned by the recognition program according to the constraints imposed by the DSL "program." The overall task may be couched in terms of an optimization problem. Interpreting the complete document consists of deciding upon the appropriate characters among the choices returned by the recognition program, and/or the appropriate alternatives in the DSL specification. Any character choice comes with a certain confidence; therefore, a solution itself has a confidence obtained by combining the elementary confidences of the choices. Since the recognizer may produce several choices for segmenting a connected component into characters and several alternatives for each character, the potential for combinatorial explosion is high. Fortunately, the explosion can be controlled quite efficiently by use of the following techniques:

1. If a field (or an element in a field) is not subject to a DSL constraint, the combination based on all of the most likely choices made by the recognizer is the best answer. Suppose we want to recognize a string of digits for which there is no constraint. The recognizer may return the following choices when trying to recognize "95320":

   0(8) 1(4) 5(3) 3(5) 2(3) 0(6,5)

   9(4) 5(3) 3(5) 2(3) 0(6,5)

   (Each line represents a different segmentation. Each group of symbols in a line gives the most likely choice made by the recognizer, followed, in parentheses, by other choices.) This illustrates what might happen if the 9 were segmented into a loop (recognized as 0, with 8 as second choice) and a vertical stroke (recognized as 1, with 4 as second choice). The alternative 9(4) corresponds to another segmentation and is also returned, but with a lesser confidence. Since there is no constraint, "015320" is the most likely answer.
2. Suppose we want to recognize a country name such as "CANADA," with the semantic condition that it belong to a dictionary D. The recognizer may return segmentation and recognition choices such as

   C(G) R(A) M(N) A(R) O(P) A(B,R)

   W(M) M(N) A(R) O(P) A(B,R)

   A brute-force approach that would systematically backtrack to try all choices and look up each obtained word in D would have to do it $(4 + 2) \times 2 \times 2 \times 2 \times 3 = 144$ times. It would not even find the correct answer, since "CANADA" is not in the set of 144 hypotheses. Instead, we can use an efficient method that performs a "fuzzy" search on the dictionary, given an approximate string—in particular, the one obtained by taking all of the most probable choices. In the example, the argument to the fuzzy search would be "CRMAOA." Fuzzy-search algorithms select the dictionary entry at the minimal distance from the search argument. Our contextual processor uses the method proposed in [19], extended for handling alternate choices. The original method, as well as the extension, uses dynamic programming.
3. Suppose we attempt to recognize a telephone number such as "9291720." We can express in DSL the syntax of a valid U.S. phone number as a string of seven digits. If there is no semantic constraint on the phone number, only the segmentation choices need be considered to ensure that the number of characters is 7. Suppose the following results from the recognition program:

   9(8) 2(3) 0(1) 1(4) 1(7) 7(4) 2(3) 6(0,5)

   9(8) 2(3) 9(4) 1(7) 2(3) 2(3) 6(0,5)

   During the backtracking process, the first hypothesis will be the string "92011726," which violates the seven-digit syntax. The next hypothesis will be "9291726," which satisfies the syntactic constraint. Note that we did not have to look at all choices for individual characters. Actually, the last digit choice is 0, but, since there is no semantic constraint, there is no reason for the contextual processor to consider 0 rather than the more probable 6. If there were a list of valid telephone numbers, the contextual processor would examine the different possible combinations of likely characters and would probably find the right answer.

Our contextual analyzer implements a context analyzer based on DSL and exploits the ideas presented above. Most of the system is operational, and we are gathering statistics to evaluate the power of the approach. Table 1 shows the results of recognition before and after contextual checking was applied to the tax form of Figure 2. Notice the many recognition errors that have been corrected by exploiting both syntactic and semantic constraints. In particular: JUIY was changed into JULY by use of a dictionary on months, BRwnerd and Mrtchen

were changed to BERNERD and MITCHEM, which is incorrect but gives a reasonable confidence (as do BERNARD and MITCHELL). The ⟨city, state, zip⟩ triple is corrected; once the zip code is known, a dictionary of local street names can be used to transform uC into Lee (the fuzzy search routine knows about alternative choices).

- *Performance evaluation*

The results above are purely qualitative. To give a better understanding of the power of our contextual processor, we provide in this section some quantitative results for two applications. The first application deals with forms of poor quality, the second one with forms of very good quality.

Application 1: Credit card application forms

The goal of this system is to recognize data from credit card application forms that are filled in by hand. A form comprises a variety of fields in small, adjacent boxes; inside a box, the writing is free-style, often careless, sometimes interfering with printed text and lines. As a result, the character-recognition rates are relatively low. Still, the use of our contextual processor exhibits the following significant improvements (evaluated on a sample of 100 forms):

- For U.S. telephone numbers, syntactic checking alone increases the recognition rate of the field from 28% to 44%.
- For last names, a fuzzy search of a dictionary of last names (10000 entries) increases the field recognition level from 9% to 27%. (Note that the dictionary does not provide 100% coverage.)
- For ⟨city, state, zip⟩, a fuzzy search of a complete dictionary (about 45000 entries) increases the field recognition rates from 8% to 60% for city, 18% to 64% for state, and 28% to 50% for zip.
- For college names, a fuzzy search of a small, complete dictionary of triples (⟨name, campus, zip⟩) increases the field recognition rate from 4% to 90%. This tremendous jump is understandable: Without context, the probability of recognizing all letters correctly is very small for a long string (about 40 letters); with our contextual processor, only a small number of letters have to be recognized in order to identify the correct value.

Application 2: Shipping form

The form identifies the sender by company name, address, postal code, city, and state. The receiver is identified by company name, customer number, and postal code. A full database contains all of this information for all companies involved. The form is carefully filled in by hand, generally with attention paid to the individual letter boxes. There are three lines for the address; all elements of the address are not always used, and the grouping of the elements in three lines is not always done consistently. This means that not only must the words be recognized, but the elements must be labeled by their semantic role: number, street, city, etc. However, the redundancy and the availability of a complete database facilitate the task greatly.

A brief summary of the results (evaluated on a sample of 65 forms) follows:

- The identification of the receiver or sender is correct in more than 90% of the cases.
- The recognition of yes/no marks is, as expected, excellent (99.8%).
- The recognition rate of individual handwritten integers is over 95% (many of the errors were due to the ligatures in 00 or 000).

These numbers are quite promising. Also, the results of both applications underscore the importance of the form design, which is something that should be kept in mind for all applications.

However, the most striking result of our experiment has been the ease with which our contextual processor can be adapted to new applications. The use of DSL and the system behind it drastically reduces the development time of new applications and makes readily available an excellent way of exploiting contextual knowledge.

- *Verification and correction system*

Whatever the quality of the recognizer and the ability of the context analyzer to improve the results, the percentage of fields correctly recognized is still short of what is needed for a data entry application. For the whole process, the client wants 100% correctness, or something close to that limit.

It is clear that human intervention is needed. We are currently adding to the system described above an interactive phase for verification and correction. For each field, the user will be able to choose a sequence of operations invoked in a specified order. If the first operation is sufficient to assign a value to a character or a field, and do it with enough confidence, the sequence is interrupted. Otherwise, the second operation is executed. If it does not yield a sure result, the next operation is executed, etc.

Suppose we need to recognize, with very high confidence, a numerical field in which the last character is a check code computed from the other digits (e.g., the sum of all digits, modulo 7). A meaningful sequence of operations may be as follows: an automatic step in which a result (that satisfies the check-code constraint) is accepted if the confidence is higher than a threshold specified by the user, an interactive session in which

characters are shown and certified or simply entered by the operator, a new invocation of our contextual processor that uses the newly acquired information to determine the values of more fields, and finally another interactive session in which the operator sees the fields that have not yet been resolved and enters their values. Typically, the first contextual-processor execution may yield a result in 80% of the cases; for the 20% remaining, the second execution may yield a result in 80% of the cases not resolved in the first execution (i.e., 16% of all cases). Only 4% of the cases have to be shown to the operator for manual acceptance or entry.

For each operation, the system must facilitate operator interaction as much as possible. Some special techniques can be used for that purpose. In particular, the verification of isolated characters relies on a technique called "carpeting." It consists in showing an array of 100 (or some appropriate number) character images that have been recognized as "1," for example. If the operator sees an image that is not a "1," the operator can click on the image, thereby invalidating the recognition result. The same is done for all "2"s, "3"s, etc. This method is efficient only for characters that have been recognized with high confidence, i.e., an error rate of only 2–3%. If this is not the case, it may be more efficient to show the complete field result.

Applications/services of image-based forms processing

The components described in this paper have been implemented in a software package that runs under both AIX® and OS/2® operating systems. In this section we describe briefly several of the more significant projects for customers in which our software has been successfully deployed. To protect the anonymity of our customers, we do not divulge their names.

- *Automated processing of state census forms*

The first application involved the national census of a European country. There were different language versions of the same form (since several official languages were used). Each form consisted of four pages, and every page had a preprinted ID number of nine numerals, all printed in the same font. This ID number contained information about the form type and the language used. The field containing the ID number was processed by our machine-print ICR software. The ICR result was used to identify the language of the form and the page number, and also to carry out the form recognition. The "signature" portion of the form recognition was not used here, since forms printed in different languages had the same signatures of lines. Hence, the ICR of the ID field (specified as a UDF of the form) was used for form recognition. Once the form language was known, the corresponding form template was used for form dropout and subsequent field extraction and ICR.

About 32 million pages have been processed by this application.

- *Tax return imaging system*

This application handles tax-return forms by scanning the forms, recognizing them, and compressing them for subsequent archiving and ICR. This application is characterized by many forms that are very similar to one another. For example, the same type of form is created by different printers; each variation of the form is a different template for the purpose of form recognition. Another example is tax forms that are almost identical except for the year number, which generally appears at the top of the form. Also, many taxpayers submit their returns on photocopies, which makes form recognition difficult. Nevertheless, the form-recognition algorithm is able to cope with most of the cases, generally by applying its second phase—i.e., by utilizing UDFs with appropriate matching. The filled-in data are largely in unconstrained hand-printing. The use of recognition-based segmentation is helpful in separating touching characters. In one of the projects, the number of people working on data entry was reduced by 75%. This was partly because of the automated data entry and partly because of the productivity improvement from an image system employing automated routing and distribution of documents.

- *Postal-address readers*

Reading postal addresses, sorting mail, and delivering mail are labor-intensive tasks for post offices throughout the world. Automation is increasingly employed in this task by scanning the envelopes, locating the destination-address block, reading the address using ICR, interpreting the address, and imprinting a bar code (corresponding to nine- or eleven-digit zip codes in the U.S.), which eliminates the need for any manual operation until the point of delivery. The five-digit zip code can be obtained from the zip code field in the address. If the remaining four or six digits are not present, they have to be inferred from the street address field; this is why the entire address has to be read and interpreted. Postal applications are characterized by extremely high throughput rates—for example, 90000 addresses per hour for a system. Those addresses for which the machine does not have enough confidence must be verified and corrected by human operators. In the particular project with which we were associated, our ICR is used for reading hand-printed or machine-printed addresses.

- *Sorting bank-check images*

This system is intended for automatically adding check images into a database according to account number. The

system was installed at the central branch of a large bank. Every check was scanned and processed by the FDO module. All of the checks had the same template, and all of the fields were printed on every check with the same font. After FDO, the fields containing the account ID, the check number, and the telephone number of the account owner were processed by ICR. There was also a CMC-7 code line (CMC-7 is a font that can be read by humans as well as by magnetic and optical means) printed at the bottom of each check, which contained the account ID, the check number, and the amount to pay. The CMC-7 line was optically read by a special ICR program.

Usually a check passes through many hands on its way from the account owner to the scanning station, and the signature often falls on the position occupied by the CMC-7 code line. Therefore, the image data to be recognized by ICR were of poor quality, and logic was used to match all of the results produced by the ICR for the different fields of the same check in order to increase recognition accuracy. Check digits were also used to detect and sometimes to correct possible recognition errors.

- *Indexing and archiving bank forms*

Banks maintain customer files, including various kinds of forms such as signature samples, mortgage applications, and account-opening applications. To facilitate forms handling, banks are looking into image-archiving solutions. Our solution includes systems that scan the forms, index them via form recognition, and compress them via FDO. This application generally involves a large number of form types. Our systems can process existing forms and add new forms interactively.

- *Imaging payment receipts in an insurance company*

The car-insurance department of a large company manages a database of clients. A new client or a client who intends to renew an insurance contract receives a statement from the company in order to pay the premium to a bank. The statement consists of two similar pages, insurer's receipt and client's receipt, and contains information printed out from the database: client's ID and details of car, insurance term, policy ID, and the amount of the payment. The receipts are printed in hundreds of the company's agencies with different printers and fonts. Dot-matrix printing is used in many cases.

All paid insurer's receipts, stamped by banks, are received by the company central office. Then the corresponding records of the client database are marked as "paid," and the receipts are stored in the office site. The receipts processing was formerly carried out manually.

Our solution is as follows. All of the receipts are scanned, and all of the images pass form recognition and FDO. (At present, there are two types of receipt form.) Then all of the fields containing the insurance information described above are read by ICR. The ICR uses an *omnifont* recognition database with a very wide repertoire of fonts.

Next, the corresponding records are automatically found in the client database. The search essentially uses the redundancy of the information obtained in the multiple fields. The system shows very high recognition performance. Typically, the system fails to find the corresponding record only if the receipt was printed with strong misalignment relative to the form template.

- *Imaging credit card slips and giro slips in bank branches*

This application is to automate the data-entry system for the information contained on credit card and giro (payment voucher) slips. The program runs in hundreds of branches of two banks. The documents are scanned by small desktop scanners, resulting in the production of gray-scale images with 16 shades and a resolution of 100 pixels per inch in both directions.

Credit card slips The role of ICR is to read impressed credit card numbers and preprinted slip numbers. The slips are those filled out by hand. The card and slip numbers are copied from the card to the slip via carbon paper. The main problems are the presence of dark copy-ink spots and extremely variable contrast levels.

Giro slips These payment vouchers include text lines containing about 50 digits and special symbols, printed in a special font called OCR-B. The lines are composed of parts printed by different sources, leading to misalignment of different parts of the line. Another factor making recognition difficult is the low level of contrast on the input documents.

The ICR subsystem can work only on black-and-white input images. Since a considerable fraction of the documents processed by this system cannot be recognized, even by human readers, when scanned in black-and-white mode, binarization of the images (converting gray-scale images to black-and-white ones by setting a pixel white when its gray-scale value exceeds a given threshold, and black otherwise) is performed by the software, with feedback from ICR. If the recognition confidence for a character is sufficiently high (and if the number of black pixels in the area of the character exceeds a fixed threshold), the character is accepted; otherwise, the binarization threshold is modified, and recognition is attempted again. We call this technique "ICR-driven binarization."

The system has been tested on a benchmark with 1000 images from authentic giro and credit card slips. No contextual correction was made, in order to test the net ICR performance. A document was considered to be erroneous if at least one character was recognized

erroneously. Each credit card slip contained 20 characters on average; each giro slip contained 50 characters on average. The testing has shown the following:

- Credit card slips: error rate = 0.7%, reject rate = 3.6%.
- Giro slips: error rate = 0.4%, reject rate = 0.9%.

The giro slip data contain less noise than the credit card data, which is reflected in the testing results.

The system has been in use by the bank in a production mode for about two years.

Conclusion

In this paper, we have described a system for the machine reading of forms data from scanned images. Given the state of the art of recognition technology, it is impossible to completely eliminate operators from the data entry process. However, our field tests show that we are able to provide a twofold improvement in productivity in the data entry process, even with unskilled operators, compared to key entry by professional operators. Various modules of the system described in this paper have been used in customer projects worldwide. The verification and correction subsystem, which is a crucial component of the system, is still in development. We expect to see further improvements in productivity when this subsystem attains maturity. We expect our system, after further development and tuning, to go a long way in reducing data entry costs for many applications.

References

1. *Computerworld*, April 11, 1994, p. 26.
2. R. Hunter and A. H. Robinson, "International Digital Facsimile Coding Standards," *Proc. IEEE* **68,** No. 7, 854–867 (1980).
3. D. J. Burr, "Elastic Matching of Line Drawings," *IEEE Trans. Pattern Anal. & Machine Intell.* **3,** No. 6, 708–713 (1981).
4. S. Ayoun, I. Berger, D. Chevion, I. Gilat, A. Heilper, O. Kagan, Y. Medan, S. Sorin, E. Walach, and E. Yair, "Automatic Data Acquisition in 1991 Swiss Population Census," *Proceedings of the 27th Convention of the IPA (Information Processing Association) of Israel*, November 1992, pp. 143–153.
5. S. Mori, C. Y. Suen, and K. Yamamoto, "Historical Review of OCR Research and Development," *Proc. IEEE* **80,** No. 7, 1029–1058 (1992).
6. J. Mao, K. M. Mohiuddin, and T. Fujisaki, "A Two-Stage Multi-Network OCR System with a Soft Pre-Classifier and a Network Selector," *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, 1995, pp. 78–81.
7. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing, Volume 1: Foundations*, MIT Press, Cambridge, MA, 1986.
8. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
9. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Second Edition, Academic Press, Inc., New York, 1990.
10. H. Takahashi and K. M. Mohiuddin, "A Clustering Method and Radius Tuning by End Users," *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, 1995, pp. 698–701.
11. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.
12. L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition," *IEEE Trans. Syst., Man, & Cybernet.* **22,** No. 3, 418–435 (1992).
13. H. Takahashi, "A Neural Net OCR Using Geometrical and Zonal-Pattern Features," *Proceedings of the First International Conference on Document Analysis and Recognition*, 1991, pp. 821–828.
14. K. M. Mohiuddin and J. Mao, "A Comparative Study of Different Classifiers for Handprinted Character Recognition," *Pattern Recognition in Practice IV*, North-Holland, Amsterdam, 1994, pp. 437–448.
15. R. G. Casey and E. Lecolinet, "Strategies in Character Segmentation: A Survey," *Proceedings of the Third International Conference on Document Analysis and Recognition*, Montreal, Canada, 1995, pp. 1028–1033.
16. R. G. Casey, "Segmentation of Touching Characters in Postal Addresses," *Proceedings of the 5th U.S. Postal Service Technology Conference*, Washington, DC, 1992, pp. 743–754.
17. *The First Census Optical Character Recognition System Conference*, R. A. Wilkinson and J. Geist et al., Eds., NISTIR 4912, U.S. Department of Commerce, NIST, Gaithersburg, MD, 1992.
18. Raymond Lorie, "A System for Exploiting Syntactic and Semantic Knowledge," *Proceedings of the Workshop on Document Analysis Systems*, Kaiserslautern, Germany, 1994, pp. 277–294.
19. R. A. Wagner and M. J. Fisher, "The String-to-String Correction Problem," *J. ACM* **21,** No. 1, 169–178 (1974).

Received February 10, 1995; accepted for publication August 29, 1995

Sandeep Gopisetty *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (sandeep@almaden.ibm.com).* Mr. Gopisetty received the M.S. degree in computer science from Santa Clara University in 1991. He is currently a research staff programmer in the Almaden Computer Science Department. His research interests are in character segmentation and recognition, and object technology and databases.

Raymond Lorie *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (lorie@almaden.ibm.com).* Mr. Lorie is a research staff member at the IBM Almaden Research Center. He graduated as Ingenieur Civil Electricien-Mecanicien from the University of Brussels, Belgium, in 1959, and joined IBM in 1960. He has been working on relational database systems since the early 1970s. Mr. Lorie was a major contributor to the architecture and implementation of System R, the early research prototype that was the precursor of the IBM relational products. In 1980, he managed a project on the application of relational technology to engineering and other non-business areas.

From 1985 to 1989, he managed a parallel-database-machine project. Since 1990, Mr. Lorie has been involved in several facets of document recognition. His main interests are in the exploitation of contextual knowledge, and verification and correction subsystems. His work on System R was recognized by an IBM Corporate Award; he also shared the 1988 ACM System Award with six colleagues from IBM and U.C. Berkeley. He is a member of the ACM.

Jianchang Mao *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120.* Dr. Mao received the B.S. degree in physics in 1983 and the M.S. degree in electrical engineering in 1986 from the East China Normal University, Shanghai, P.R. China. He completed the Ph.D. degree in the Department of Computer Science at Michigan State University in August 1994. Since January 1994, he has been with the IBM Almaden Research Center. His research interests include pattern recognition, neural networks, document image analysis, image processing, computer vision, and parallel computing. Dr. Mao received the Honorable Mention Award from the Pattern Recognition Society in 1993. He is a member of the IEEE.

Moidin Mohiuddin *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (moidin@almaden.ibm.com).* Dr. Mohiuddin is the manager of the Document Image Analysis and Recognition group at the IBM Almaden Research Center. He received the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1977 and 1982, respectively. He joined IBM Research in 1982. Dr. Mohiuddin's research interests include document image analysis, pattern recognition, data compression, and computer architecture. He is an associate editor of the *IEEE Transactions on Pattern Analysis and Machine Intelligence*; he was also on the editorial board of the IEEE *Computer* magazine from 1984 to 1989.

Alexander Sorin *IBM Research Division, Haifa Research Laboratory, Matam, Haifa 31905, Israel (asorin@vnet.ibm.com).* Mr. Sorin received the M.Sc. degree in 1979 from the Department of Applied Mathematics at the Moscow Institute of Oil and Gas. Until 1987, he was a staff member of the Central Geophysical Expedition, Moscow, where he conducted research in digital signal processing and its application to seismology. Since joining the IBM Haifa Research Laboratory in 1988, Mr. Sorin has worked in the area of document imaging and optical character recognition.

Eyal Yair *IBM Research Division, Haifa Research Laboratory, Matam, Haifa 31905, Israel (yair@vnet.ibm.com).* Dr. Yair received the B.Sc. degree in 1982 and the Ph.D. degree in 1987, both from the Department of Electrical Engineering at the Technion–Israel Institute of Technology. In 1983 he joined the IBM Haifa Research Laboratory, where he was involved in various projects in speech processing, image processing, and OCR. From 1987 to 1989, Dr. Yair was a visitor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara, where he was involved in neural network and pattern classification research. Currently, he is engaged in document image processing and handwritten-character recognition.