1993

Syntactic Segmentation and Labeling of Digitized Pages from Technical Journals

Mukkai Krishnamoorthy
Rensselaer Polytechnic Institute

George Nagy
Rensselaer Polytechnic Institute, nagy@ecse.rpi.edu

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Mahesh Viswanathan
IBM Pennant Systems, Boulder, CO

Syntactic Segmentation and Labeling of Digitized Pages from Technical Journals

Mukkai Krishnamoorthy, George Nagy, *Senior Member, IEEE,* Sharad Seth, *Senior Member, IEEE,* and Mahesh Viswanathan, *Member, IEEE*

***Abstract*—Alternating horizontal and vertical projection profiles are extracted from nested sub-blocks of scanned page images of technical documents. The thresholded profile strings are parsed using the compiler utilities Lex and Yacc. The significant document components are demarcated and identified by the recursive application of block grammars. Backtracking for error recovery and branch and bound for maximum-area labeling are implemented with Unix Shell programs. Results of the segmentation and labeling process are stored in a labeled X-Y tree. It is shown that families of technical documents that share the same layout conventions can be readily analyzed. More than 20 types of document entities can be identified in sample pages from the *IBM Journal of Research and Development* and IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. Potential applications include preprocessors for optical character recognition, document archival, and digital reprographics.**

I. INTRODUCTION

THIS PAPER demonstrates a specific solution to a general problem in pattern recognition: simultaneous segmentation and classification (a.k.a. scene analysis). Most of the published research has concentrated on isolating individual objects and then identifying them according to shape or texture features and possibly back-tracking to an alternative segmentation if the identification is not successful. Spatial relations between objects, when they are considered at all, are introduced at later stages. It is now becoming clear that this approach to the analysis of complex scenes is prone to failure. Segmentation and classification must be performed in tandem or, at least, very closely interwoven. Although we have not discovered the universal solution, for relatively well-structured *document images,* we have developed robust data structures and algorithms that may also provide a point of departure for more complex vision tasks.

Our specific objective is to extract the spatial structure of a digitized printed page from a technical article, as shown in Fig. 1. Among intended applications are those where key entry or optical character recognition (OCR) fail to capture important format-related aspects of the document and those where key entry is uneconomical and OCR is beyond the state of the art [26]. We are also studying other applications where layout analysis can be used for preprocessing documents for OCR.

With the advent of high-resolution, low-cost scanners and high-capacity storage devices, digitized document analysis has attracted many researchers from both universities and industrial laboratories. Applications include the selection of encoding methods for document archival, retrieval, high-quality facsimile, and digital reprographics as well as preprocessors for OCR. Diverse methods have been applied to postal addresses, business correspondence, newspapers, technical journals, repair manuals, maps, and engineering drawings. The methods are documented in the proceedings of specialized conferences on document image analysis [4], [5], [15], [23], and pattern recognition [2] as well as in recent special issues of technical journals [10], [24], [25]. Published bibliographies on the topic include [7] and [11].

Aside from the methodology, the goal of all of these projects differs from ours inasmuch as they do not attempt to differentiate a large number (several dozen) of categories of textual information solely on the basis of publication-specific layout information. We are not aware of any other formal system that allows detailed hierarchical description of the structure of families of technical documents in a form that is suitable for recursive segmentation and labeling of the significant components of a document image.

From a theoretical point of view, we present two complementary ideas. The first is the X-Y tree data structure, which transforms a 2-D image analysis problem into a hierarchy of quasi-independent 1-D (string) problems. (Successive string-analysis problems are quasi-independent in the sense that the results of analyzing a predecessor string can be neatly and concisely encapsulated as *a priori* knowledge for the analysis of its successors.) The X-Y tree is a nested decomposition of blocks into blocks. At each level, the decomposition is induced by partitions only in one direction (horizontal or vertical), but a block may have an arbitrary number of children. The leaves of X-Y tree decompositions represent only an asymptotically vanishing fraction of all possible decompositions of rectangles into rectangles [12], but such decompositions represent almost all technical page structures of interest (if only because other types of layouts are difficult to obtain with both current and classical page-composition tools).

Manuscript received December 2, 1990; revised September 6, 1992. This work was supported by U.S. West Advanced Technologies Sponsored Research Program, the University of Nebraska-Lincoln Center for Communication and Information Science (CCIS) and the U.S. Department of Education College Library Technology and Cooperation Grants Program. Recommended for acceptance by Editor-in-Chief A. K. Jain.

M. Krishnamoorthy is with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180.

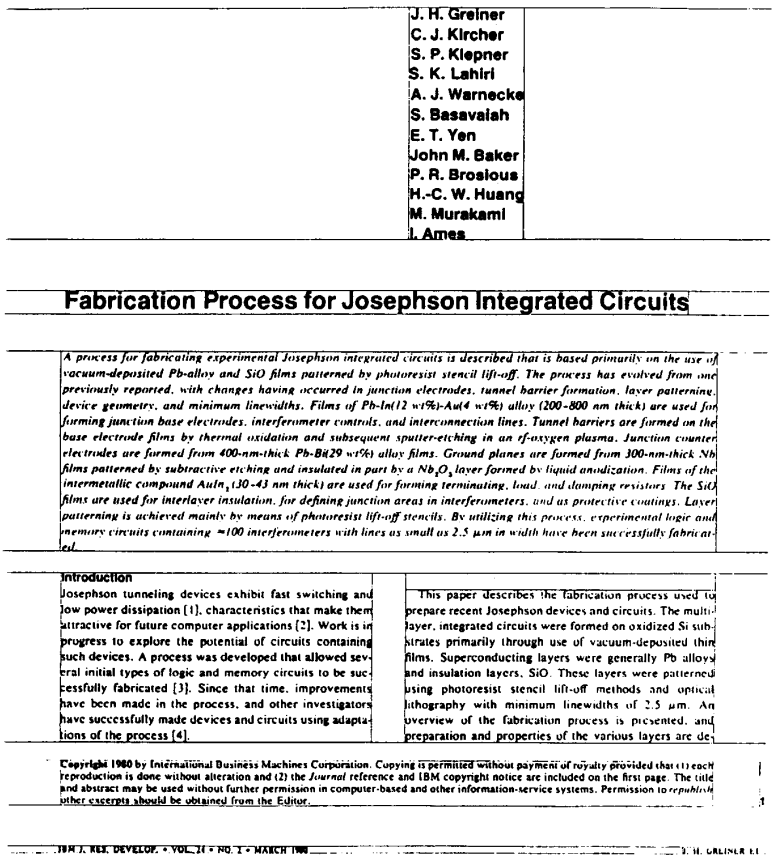
G. Nagy is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180.

S. Seth is with the Computer Science and Engineering Department, University of Nebraska-Lincoln, Lincoln, NE 68588.

M. Viswanathan is with IBM Pennant Systems, Boulder, CO 80301.

IEEE Log Number 9208435.

doi: 0.1109/34.221173



J. H. Greiner
C. J. Kircher
S. P. Klepner
S. K. Lahiri
A. J. Warnecke
S. Basavaiah
E. T. Yen
John M. Baker
P. R. Broslous
H.-C. W. Huang
M. Murakami
I. Ames

Fabrication Process for Josephson Integrated Circuits

A process for fabricating experimental Josephson integrated circuits is described that is based primarily on the use of vacuum-deposited Pb-alloy and SiO films patterned by photoresist stencil lift-off. The process has evolved from one previously reported, with changes having occurred in junction electrodes, tunnel barrier formation, layer patterning, device geometry, and minimum linewidths. Films of Pb-In(12 wt%)-Au(4 wt%) alloy (200-800 nm thick) are used for forming junction base electrodes, interferometer controls, and interconnection lines. Tunnel barriers are formed on the base electrode films by thermal oxidation and subsequent sputter-etching in an rf-oxygen plasma. Junction counter electrodes are formed from 400-nm-thick Pb-Bi(29 wt%) alloy films. Ground planes are formed from 300-nm-thick Nb films patterned by subtractive etching and insulated in part by a Nb_2O_5 layer formed by liquid anodization. Films of the intermetallic compound $AuIn_2$ (130-43 nm thick) are used for forming terminating, load, and damping resistors. The SiO films are used for interlayer insulation, for defining junction areas in interferometers, and as protective coatings. Layer patterning is achieved mainly by means of photoresist lift-off stencils. By utilizing this process, experimental logic and memory circuits containing ≈100 interferometers with lines as small as 2.5 μm in width have been successfully fabricated.

Introduction

Josephson tunneling devices exhibit fast switching and low power dissipation [1], characteristics that make them attractive for future computer applications [2]. Work is in progress to explore the potential of circuits containing such devices. A process was developed that allowed several initial types of logic and memory circuits to be successfully fabricated [3]. Since that time, improvements have been made in the process, and other investigators have successfully made devices and circuits using adaptations of the process [4].

This paper describes the fabrication process used to prepare recent Josephson devices and circuits. The multilayer, integrated circuits were formed on oxidized Si substrates primarily through use of vacuum-deposited thin films. Superconducting layers were generally Pb alloys and insulation layers, SiO. These layers were patterned using photoresist stencil lift-off methods and optical lithography with minimum linewidths of 2.5 μm. An overview of the fabrication process is presented, and preparation and properties of the various layers are de-

Fig. 1. X-Y tree for a title page from the *IBM Journal of Research and Development*. The tree represents a logical segmentation.

The second idea is the combination of a conventional syntactic formulation (and existing Unix compiler tools) with a branch-and-bound search algorithm. All legal decompositions of a block in a specified direction are prescribed by means of a context-free grammar applied to a string extracted from the block. Parsing the string effectively segments it into labeled substrings that specify both the partitioning of the block and the label of each partition. If, at any stage, a string is found invalid (i.e., it cannot be parsed with the assigned grammar), then the algorithm backtracks to an alternative grammar (if available) for a labeled predecessor-block higher up in the tree. The parsing stages will be described in detail in the next section.

Since both the X-Y tree and the document processing applications have been presented previously [9], [16], [18]–[22], [28]–[30], the emphasis here is on the formulation of the multistage syntactic analysis, which will be described in detail. As an alternative or adjunct to the method described here for document analysis, the use of knowledge bases and expert systems has also been suggested by us and others [6], [14], [17], [32].

II. Method

The essence of our approach is to transform a 2-D segmentation and labeling problem into a tree-structured set of 1-D segmentation and labeling problems. A block is segmented into sub-blocks by parsing its profile string, say, in the horizontal direction. Each sub-block then engenders a vertical profile string that can be similarly parsed for vertical segmentation. The segmentation process may be carried out recursively to any desired depth with alternating horizontal and vertical subdivisions. The parameters (i.e., the grammar) of the parse depend on the label of the block to which it was applied. The process terminates at leaf nodes, which are characterized by having labels for which no grammars are available.

The algorithm attempts to correct segmentation and labeling errors by backtracking to alternative grammars whenever a profile string cannot be parsed. Among partially labeled X-Y trees, it chooses the one whose labeled leaf blocks cover the largest area.

The preprocessing required by this method is simple and will be described first. Then, we will discuss the manner in which a 1-D string is generated from a block and explain the parsing process for recursively segmenting a single block and labeling the resulting sub-blocks. We modify this simple tree expansion by incorporating 1) backtracking for recovery from errors and 2) a branch-and-bound strategy to find the largest area of the root block that can be labeled.

A. Preprocessing

Each page is converted to digital form by scanning it horizontally at a sampling rate sufficient to preserve all sig-

TABLE I
NOISE FILTERING RESULTS

Filter Size	Dots Erased	Noise Erased.
0x0	0	0
1x1	0	30
2x2	34	40
3x3	154	55
4x4	171	67
5x5	171	73

nificant white spaces. Since the entire X-Y tree approach is extremely sensitive to skew, in our experimental work, each page is aligned on the scanner bed with extreme care. In a production system, this would be impossible, but excellent skew-correction methods are available.

There are two alternatives for accommodating specks of noise due to fiber flaws in the stock, imperfect reproduction, or digitization. Such noise does not bother human readers but complicates automated analysis. The first method is to make the grammars sufficiently robust to ignore such noise. This is quite feasible but tedious. The approach we have chosen instead is to remove all specks smaller than a given size in a preliminary pass (for which we use either a connected-components algorithm or transition segmentation [16]).

We studied the effect of such a preprocessing filter on one IBM nontitle page obtained by photocopying and scanning. Table I presents the result obtained on noise filtering the page. The page originally had 171 *dots* by manual count (*dots* are dots on i's and j's, periods, and decimal points). Filtering with a 1×1 window left all of the *dots* intact but removed 30 noise specks. Increasing the filter size to 2×2 eliminated 34 *dots* (mostly decimal points and dots on i's and j's with periods left intact). The 4×4 window eliminated all of the *dots*. Clearly, noise specks taper off to just six using the 5×5 window. Experiments on the CD-ROM database, photocopied and scanned pages, and synthesized (typeset using the troff formatter) and scanned pages yielded similar results. Of course, the number of noise specks is higher on photocopied pages. The size threshold therefore can be quite generous. Loss of a few periods or dots on the i's and j's does not affect the layout analysis. Our conclusion is that speckle noise cannot be filtered out by purely local means without degrading the legibility of the page. Therefore, after the analysis is completed, all of the specks are restored before any document component is presented for human inspection or OCR.

Block Segmentation and Labeling

Each block is segmented into sub-blocks by extracting a profile string and parsing it with a context-free grammar. The parse divides the string into a sequence of labeled substrings, each of which corresponds to one dimension of a sub-block. Each block is processed either horizontally or vertically. For the sake of concreteness, in the following description, we will assume that the block is segmented by parsing the thresholded *horizontal* profile of the block.

The *horizontal profile* of a block of m rows and n columns of pixels consists of the m row sums of the array [1]. The *thresholded horizontal* profile is the binary string of length m obtained by replacing each element of the horizontal profile by 1 if its value exceeds the threshold and by 0 otherwise. With a threshold of 1/2, the thresholded profile string will have zeroes only for rows of pixels that are completely white. The program that generates the binary profile needs to scan each row only until the first black pixel is encountered.

Although, in principle, a single context-free grammar can be constructed for parsing a profile string, in practice, it is easier to divide the process into four separate stages. The *nonterminal* symbols of each stage are the terminals of the following stage. The parameters of the analysis (which are called a *block grammar*) depend on the label assigned to the block by the parse at the level above it. The grammar for the root block is called a *page grammar*.

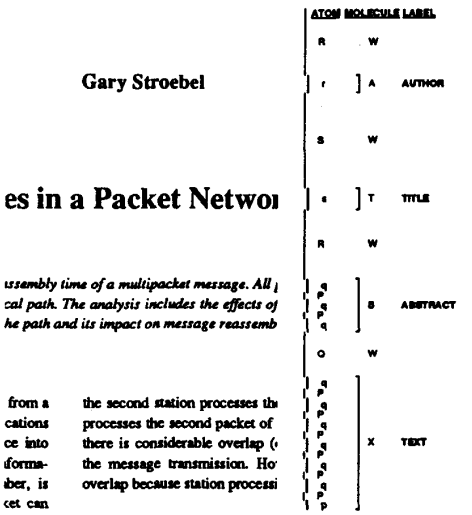
All of the block grammars, regardless of the level or label of the block being analyzed, contain a number of similar productions. These constructs can be readily parametrized. The seemingly eccentric notation used below for the parametrization was chosen to avoid bias towards the label of the block, the direction of segmentation, and the level of segmentation.

Stage 1—Atom Generation: The first stage, which is written in C, simply counts the lengths of the all-one and all-zero substrings (which are called *atoms*) in the profile and assigns them to equivalence classes according to their length. For instance, black strings of ranges of length [30–40], [41–45], and [46–70] may be assigned to three classes p, q, and r. In subsequent stages, atoms of type p or q may be considered candidates for lines of text, whereas atoms of type q or r may be candidates for title lines. The ambiguity of the nonterminal symbol q is removed in subsequent stages.

Stage 2—Molecule Generation: The second stage (a *lex* program [13]) assigns the atoms into groups of contiguous atoms called *molecules*, according to a set of regular expressions based on permissible sequences of atoms. For example, an alternating sequence of black atoms corresponding to candidate *title lines* and white atoms corresponding to candidate *intertitle line spaces* will be tentatively labeled as *title*. The number of repetitions in the sequence (which is called *valence*) is taken into account; for instance, *title* may be restricted to no more than three *title lines*.

Stage 3—Labeling: The third stage (a *yacc* parser with single-token lookahead for context-free grammars [8]) assigns permanent (*entity*) labels to each molecule according to the permissible sequences (*precedence*) and number (*cardinality*) of molecules in each class of entity. For instance, a page may contain multiple instances of some entity (such as a column or a paragraph) but only a single author block, and the author block must be above the title block. The precedence constraint allows the third stage to disambiguate entities even if their corresponding molecules appear similar because they are set in the same font and have the same number of text lines. If a parse according to the given grammar cannot be constructed, *yacc* reports failure.

Stage 4—Merger: After the third stage, the string has been segmented and labeled. It is possible, however, that some entities of the same type (such as paragraphs of text) were unnecessarily separated in the second stage because of wide separation that might indicate a change in entity type. The fourth stage just merges contiguous entities of the same type.



Stage 1			Stage 2			Stage 3		
p	→	$1^{\{40-50\}}$	A	→	$r \mid (rP)^{\{1-11\}}r$	Title_page	→	W_A AUTHOR W_AT TITLE W_TB ABSTRACT W_BX TEXT
q	→	$1^{\{55-70\}}$	T	→	$s \mid (sP)^{\{1-2\}}s$	W_A	→	W
r	→	$1^{\{85-95\}}$	B	→	$q \mid (qP)^{\{1-12\}}q$	AUTHOR	→	A
s	→	$1^{\{101-110\}}$	X	→	$((p\|q)P)^{\{3-29\}}p\|q$	W_AT	→	W
P	→	$0^{\{25-55\}}$	W	→	Q|R|S	TITLE	→	T
Q	→	$0^{\{201-300\}}$				W_TB	→	W
R	→	$0^{\{301-400\}}$				ABSTRACT	→	B
S	→	$0^{\{401-500\}}$				W_BX	→	W
						TEXT	→	X

Fig. 2. Simplified example of a block grammar. In Stage 1, runs of ones or zeros are condensed into atoms. In Stage 2, atoms are grouped into molecules. In Stage 3, molecules are interpreted as document entities.

The net result of the profile parsing process described above is to convert a string of 1's and 0's (the terminal symbols) into a legal sequence of labeled substrings (nonterminals) or report failure. Each of the substrings defines a labeled block whose profile in the orthogonal direction must be extracted and parsed at the next (lower) level of analysis.

A simplified example of a block grammar for a two-column title page is shown in Fig. 2. The top part shows a fragment of the page on the left and its interpretation by successive stages of the block grammar (shown at the bottom) on the right. The runs of one's and zero's in the horizontal profile are shown by vertical bars; the thicker bars correspond to runs of 1's (i.e., black atoms). These are grouped into four black (p, q, r, s) and four white (P, Q, R, S) atoms in Stage 1, according to their length. For example, black runs of between 55 and 70 pixels are called q. In Stage 2, atoms are grouped into molecules, e.g., the molecule T is formed by combining from one to three s black atoms separated by (an appropriate number of) the P white atoms (see grammar in Fig. 2). In Stage 3, molecules are labeled as layout entities, such as *TITLE*. In Fig. 2, W_A is the top margin, and W_AT is the white space between the *AUTHOR* and *TITLE*. White spaces are not explicitly stored in the X-Y tree and are therefore not shown in the last column of the top part of Fig. 2. The first production of this stage ensures the correct top-to-bottom order of entities for this publication. In this example, Stage 4 is not illustrated.

Nested Block Grammars

The four-stage analysis described in the previous section is based essentially on a single context-free block grammar. In this section, we will extend the analysis to be able to segment and label a document page consisting of a set of nested blocks.

We start with the definition of a *block grammar* g_L as a context-free grammar described by the standard four-tuple:

$$g_L = (V_N, V_T, S, P)$$

where

V_N	set of nonterminal symbols
V_T	set of terminal symbols
S	start nonterminal
P	set of productions for a context-free grammar.

A block grammar interprets a given block as a particular entity represented by the label L. For example, a page may be parsed according to a block grammar for a *title* page or a *nontitle* page (i.e., an *intermediate page* or a *last page*) of a technical journal article. At the sub-block level, block grammars may be written for an *abstract,* a *column of regular text,* a *reference entry,* etc. During the parsing of a block, a unique label is associated with each sub-block; the correctness of the interpretation at the block level hinges on the verification that *all* its sub-blocks are correctly labeled. In a block grammar, the sub-block labels are represented by the nonterminal symbols. Blocks that are assigned terminal symbols for which no grammars are provided cannot be further segmented and are assumed to be correctly labeled. Only the nonterminal blocks with labels for which grammars are provided need to be further verified for correctness of label assignments. Thus, the nonterminal symbols of a block grammar provide the link between processing a block and its sub-blocks. We associate a block grammar with each nonterminal symbol that may be equated with the start symbol of that grammar.

The parsing of nested blocks according to a set of block grammars (one for each label assignable to a nonterminal block) can be carried out using a standard AND search. The search strategy could either be depth first or breadth first. Neither traversal order is intrinsically superior in terms of performance. For a successful parse, the two algorithms would take essentially the same time, but in case of a failure, one or the other may give an early indication depending on the location of the failing block.

There are two noteworthy features of this search strategy.

1) AND Search Tree: The solution may be described by a search tree identical in structure to the hierarchy of blocks. A boolean value representing the outcome of parsing its corresponding block is associated with each node of the tree. Since the parsing of a block can fail due to the failure of any of its nested blocks (at *any* depth), it is easily seen that each node value in the search tree is determined by AND-ing the node values of its children.

2) Independent Parsing: A block is segmented and labeled completely before any of its sub-blocks. The parsing is tentative in that it can be invalidated by subsequent parsing of a segmented sub-block. However, after a block is parsed, all its sub-blocks that are assigned a "nonterminal" label can be parsed independently; there is no interaction between their parsing processes.

It is certainly possible to postulate more complex interactions between processing of blocks than that represented by independent parsing. For example, the outcome of parsing a sub-block may be used to determine further segmentation and labeling of a block. Independent parsing, however, may be more easily adapted to parallel processing than competing schemes. After a block has been segmented and labeled, it spawns a new parsing process for each of its sub-blocks that requires further analysis.

Multiple Interpretations

In technical documents, the occurrence of the same logical entity in different forms is all too common. The text may be set in one or in two columns, on different pages (or different parts of the same page), paragraphs may be flush with the left margin or indented, the first lines of paragraphs may be left justified or indented, etc. To accommodate multiple interpretations of a block with the same label, a publication-specific entity grammar is defined for each label that can be assigned to a block in the segmentation of a page. The *entity grammar* G_L for label L is defined as a list of block grammars $\{g_L\}$, where each list element g_L represents a distinct interpretation.

The parsing algorithm based on AND search must be modified for entity grammars. The basic change is the action taken when a failure occurs: Instead of reporting a failure to higher levels, the algorithm must try another interpretation for the block if it is available. For a block at any level of segmentation, the algorithm attempts the applicable grammars in the list one by one until one of them succeeds in segmenting the block into sub-blocks. At this point, each sub-block is processed recursively with the complementary direction of segmentation. The strategy is to expand each node into a sequence of nodes representing the available alternatives for the block corresponding to that node and repeating this at each subsequent node until the whole page is processed, that is, the search can be described as an AND-OR tree.

Incomplete Interpretation—A Branch-and-Bound Algorithm

Thus far, we have been discussing exact algorithms for multiple interpretations (i.e., when there is more than one grammar for any block). Either all the blocks are segmented and labeled, or no blocks are labeled. On the other hand, we may want to obtain the best labeling possible with the available entity grammars, even if the labeling is not complete. The best labeling is defined here as that with the maximum cumulative area of the labeled blocks. This may be an acceptable objective function in situations where the designer of the syntactic model has either an incomplete or incorrect understanding of the details of the page layout.

The algorithm SEGLABEL (see Fig. 3) is a branch-and-bound method to achieve this goal. It avoids trying an alternative grammar unless it can increase the total labeled area. This algorithm is similar to the AND-OR search described earlier, except that the AND nodes are replaced by SUM, the OR nodes are replaced by the MAX operation, and a bound check is added to avoid trying alternatives wherever possible. The algorithm avoids processing a sub-block if the maximum labeled area cannot be increased over the current bound even with a complete segmentation and labeling of the sub-block. SEGLABEL uses the following four parameters:

- the top-level block to be parsed (A)
- the direction in which the profile of A is generated (D)
- the entity grammar $G_{L(A)}$, where $L(A)$ is the label sought for A
- the *lowerbound,* indicating the currently labeled area of a block (by the most successful interpretation).

SEGLABEL (A, D, $G_{L(A)}$, *lowerbound*).
The input parameters are as follows:

A	block to be parsed
D	direction in which the profile of A is generated

```
begin
  if (G_L(A) is empty) then {A is a leaf block} return (area_A);
  Extract profile of A in the direction D;
  while (G_L(A) ≠ empty do
    begin
E0:   Parse profile of A according to first( G_L(A)) using Lex and Yacc;
        {Parsing results in segmentation and tentative labeling of block A}
      if (parsing not successful) then {do nothing; lowerbound is valid}
      else { Let A_i be a sub-block assigned to the nonterminal L(A_i)}
        begin
          max ← area_A; {max is an upper bound on labeled area of A by G_L(A)}
          for each A_i do
            begin
              subblock_lowerbound ← maximum(0, lowerbound - (max - area_A_i));
E1:           bound_A_i ← SEGLABEL( A_i, ¬D, G_L(A_i), subblock_lowerbound);
              max ← max - ( area_A_i - bound_A_i); {subtract area not labeled}
E2:           if (max ≤ lowerbound) then exit {for loop};
            end {for loop};
          if (max > lowerbound) then lowerbound ← max;
        end {else clause};
      G_L(A) ← rest(G_L(A));
    end {while loop};
  return(lowerbound);
end {SEGLABEL}.
```

Fig. 3. SEGLABEL algorithm.

$G_{L(A)}$ entity grammar for the label $L(A)$ to be assigned to A

lowerbound area of A labeled by most successful previous interpretation of A.

The output parameters are as follows: The functional value of SEGLABEL is the maximum area of A successfully labeled by $G_{L(A)}$ or a previous interpretation of A. (See Fig. 3 for the main program of the SEGLABEL algorithm.)

The algorithm is called in the main program as follows:

$$\text{area} \leftarrow \text{SEGLABEL}(page - block, D, G_{L(page-block)}, 0).$$

When SEGLABEL has finished the search for the best solution for a block, the value of *lowerbound* is returned as the functional value of SEGLABEL. In the initial call to SEGLABEL, *lowerbound* is zero.

At statement E0 of SEGLABEL, we try to label and segment the current block A with the next alternative block grammar for label $L(A)$. The variable *max* is an upper bound on the labeled area of A under the current interpretation. Initially, its value is set to the total area of A, but as SEGLABEL is applied to each sub-block of A in statement E1, the value of *max* is decreased by the area of the sub-block that is not labeled. The recursive call in statement E1 leads to a depth-first traversal of the sub-blocks of A. The value of *subblock_lowerbound,* which is computed in the previous statement, is used in the call at E1. This value can be computed from the parent block's *lowerbound* value by assuming that $G_{L(A)}$ is able to label everything outside of the current block A_i, that is, an area equal to $(max - area_{A_i})$. Each time *max* is updated, a check is made in statement E2 to see if the updated value of *max* is at or below the area of A labeled by a previous interpretation of A. If so, it is not necessary to process the remaining sub-blocks of A, and an early exit from the for loop occurs. After the for loop, if *max* is still greater than *lowerbound,* this can be only because the current interpretation of A has been able to label a larger area than any of the previous interpretations. Hence, the value of *lowerbound* must be modified accordingly.

Example

Consider the following entity grammars:

H 1 $G_{text-block} = (g_{text-block.a}, g_{text-block.b})$

It is not hard to find problems for document analysis to solve, or systems designed to address the problems.

Look at the stacks of paper documents around the workplace. Some may be computer generated — inevitably by different computers and software. Some include both formatted text and labels as well as handwritten entries. The documents come in different sizes, from small business cards to large engineering drawings. Many of the businesses use imaging systems to store pictures of the paper and to make retrieval more efficient.

Department	Paper Used for 1991	Useful
Administration	1000 tons.	10%
Arts	10 tons.	70%
Engineering	500 tons.	50%
Sciences	200 tons.	30%

Future document analysis systems will be able to recognize types of documents and enable extractions of documents.

DOC-PAGE 1

Fig. 4. Three-column text block with imbedded table and an accidental alignment of white spaces.

V 2 $G_{composite-block} = (g_{composite-block.a}, g_{composite-block.b})$

V 2 $G_{table} = (g_{table})$

H 3 $G_{column} = (g_{column})$

where V and H denote the direction in which the block profile is extracted, and the numbers denote the block level. These entity grammars are used to parse a sample document, which is a three-column, all-text text block of a technical article shown in Fig. 4. The first cut or direction of analysis will be horizontal in an attempt to extract the three-column table in its entirety.

The example grammar shows that there are two alternate ways of parsing the text-block horizontally using the grammars $g_{text-block.a}$ and $g_{text-block.b}$. Each yields child objects we will call composite blocks. A composite block can be a three-column object $(g_{composite-block.a})$, a two-column object $(g_{composite-block.b})$, or a table (g_{table}). Each column can then be further subdivided by G_{column} into paragraphs. The paragraphs are the monolithic (terminal) blocks for this entity grammar. We make two assumptions about the grammars used here: a) The grammar $g_{text-block.b}$ expects a larger white space between the top composite-block and the following table than depicted in Fig. 4; b) the grammar g_{column} does not accept a footer as part of a column.

The execution of this branch-and-bound algorithm is summarized in Fig. 5. Grammar $g_{text-block.a}$ is applied first yielding three composite blocks. The first composite-block yields three columns at the next level. Each column is, in turn, parsed into paragraphs. The total labeled area of this composite block is 45%. (From here on, all the area percentages refer to the fraction of the text-block area. White entities are ignored in this example.) The next composite block is then parsed. This block is labeled "table" due to the width of its intercolumn white spaces. The total area of this composite block is 30%. (Further analysis of the table block is avoided since "table" is a terminal symbol.)

Next, the third composite block is parsed into three columns, and then, each column is processed. The column grammar fails to parse the third column because of the footer. Hence, the labeled area for composite block is 14% and 89% for the whole text block. It should be noted that $g_{composite-block.b}$ is applied in an attempt to obtain a larger labeled area for the

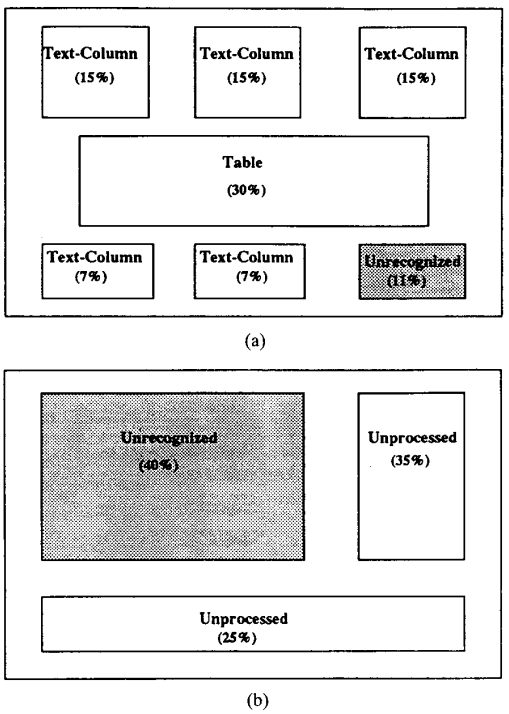


Fig. 5. (a) Labeled document from execution of the SEGLABEL algorithm for the document shown in Fig. 4 by the $g_{composite-block.a}$; (b) result of applying $g_{composite-block.b}$.

third composite block (which has a maximum labelable area of 25%), but it fails to parse the block because there are three columns of text. Fig. 5(a) illustrates the labeling that results at this stage.

At this point, the processing is back at the text-block level with the *lowerbound* value for the block set at 89% due to the first interpretation. Grammar $g_{text-block.b}$ parses the block into two composite blocks, incorrectly merging the first composite block and the table. The grammar $g_{composite-block.a}$ fails on the first composite block because the word "administration" in the table decreases the gutter width between the first two columns to below the permissible value for the grammar. Next, $g_{composite-block.b}$ is tried, and it finds two columns due to the fortuitous alignment of white (gutter) space between the second and third columns of the top text block and the table. However, grammar g_{column} fails to parse the left column (40%) due to misalignment of text lines. Now, even if the second column (35%) and composite block (25%) were correctly parsed, the maximum labeled area would be only 60%. Since this is less than the current lowerbound of 89%, the search tree is "pruned," and SEGLABEL returns without any further analysis. Fig. 5(b) illustrates the labeling attempts made at this stage. The value returned to the original call is 89%, representing the area labeled by grammar $g_{text-block.a}$.

Adequacy of Context-Free Grammars for Technical Documents

Algorithm SEGLABEL searches for a solution based on the results of profile parsing. Although our implementation is restricted to context-free grammars, from a theoretical point of view, the following question regarding a syntactic model for profile parsing is of some interest. Where in the Chomsky hierarchy can the class of profile strings of technical documents be placed?

For simplicity, we will limit our discussion to profile strings generated by text blocks. Further, we assume that the analysis is to be carried out entirely in terms of the binary profile strings.

There is a trivial answer to the question that we will disregard; regular grammars should suffice in all cases since, for a given scanner resolution, the binary strings have a fixed finite bound. In general, the bound is sufficiently large in practice to rule out effective use of the finiteness of the domain (of possible binary strings) in analysis. The answer is also unsatisfactory since it relies heavily on the specifics of the scanner technology. Although any syntactic analysis based on profile strings must deal with scanner resolution, we prefer that the technology-dependence of the analysis be limited to the lowest level of processing, i.e., formation of atoms as described earlier.

Repetition of entities is a very common feature of document layouts: repeated letters, words, lines, paragraphs, columns, etc. The resulting effect on the binary profile string is generally an alternating sequence of blocks of one's and zero's in which the lengths of one blocks are approximately equal; similarly, the lengths of zero blocks are also approximately equal but, usually, different from the lengths of one blocks. Ideally, the grammar should be able to recognize strings in which the block lengths are identical. The *language* to be recognized may be expressed as $\{(1^n0^m)^k1^n | m, n, k > 0\}$, which is context-sensitive. Thus, context-free grammars are not sufficient for document analysis.

This establishes the need for a mechanism that can keep arbitrarily large counts and match them to augment syntactic analysis. The next question is whether there are other aspects of document analysis that are not captured by such augmentation. Indeed, can we say that a counting mechanism is all that is necessary to augment a finite state machine (regular grammar) for document analysis? We have not yet been able to come up with a good example of a document feature that cannot be handled by the augmented finite state machine mechanism.

We are not the only researchers to use context-free grammar (augmented with counting mechanism to handle limited forms of context sensitivity) for syntactic pattern recognition. Tanaka [27] points out that almost all researchers use context-free grammar in their syntactic pattern recognition studies. The reasons that he gives are as follows: First, a context-sensitive grammar is hard to treat. Second, parsers and error-correcting parsers for a context-sensitive language are very complicated and costly.

III. Experimental Results

Syntactic Segmentation

Twenty-one photocopied pages of the *IBM Journal of Research and Development* (from 1979–1984 issues; see Fig. 1) were scanned on a MicroTek flatbed scanner at 300 dots/in.

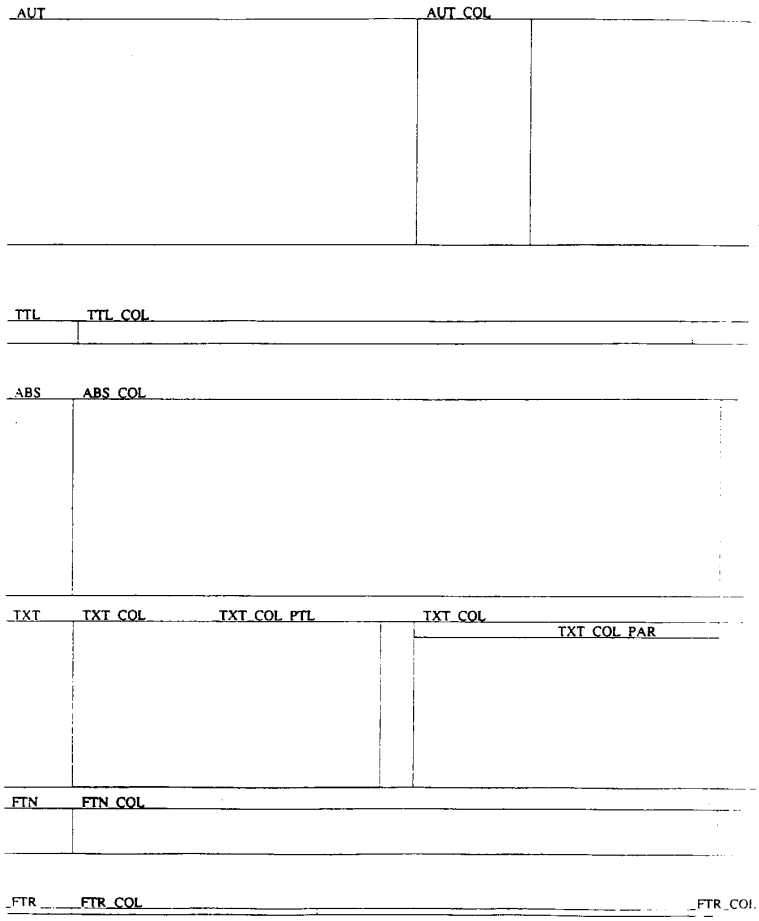


Fig. 6. Automatically constructed and labeled tree for the example shown in Fig. 1. For clarity, only the labels and node identities of nonwhite nested blocks are shown.

(Earlier results using a single grammar at each level on *IBM Journal* pages *synthesized* using TROFF were reported in [28].) Multiple block grammars were written for the 20-odd entities (such as, *header, author, footer, footnotes, abstract, paragraphs,* and *references*) that occur in the Research Contributions section of the journal. Each page was separately processed in the manner described down to the paragraph level. As discussed in the last section, Fig. 2 shows the results of the first three stages of analysis for only the root block of another page (in this case, Stage 4 is vacuous). The final segmentation and labeling results for a sample page are displayed in Fig. 6. All 21 pages were eventually processed completely and correctly after modifications to the grammars in the course of the experiment.

To reduce the time necessary to develop grammars, we simplified the specification of the relatively restricted grammatical constructs needed for block analysis. The simplification consists of a tabular method of describing page components that avoids the need for familiarity with X-Y trees, programming languages, or *lex* and *yacc*. The table includes information about the following parameters: direction of cut; the ranges of atom lengths, valences, and cardinality; the type (black or white) of atoms; the logical label of the node itself and of its preceding and succeeding nodes; and the number of possible succeeding nodes. Programs were written to translate the parameter table to *C*, *lex,* and *yacc* code. We repeated the experiments reported above using these *parameter tables* instead of the hand-coded grammars. All 21 pages were processed correctly, with the single exception of a *table* being incorrectly identified as a *text* paragraph. All subsequent experiments were conducted using parameter-table grammars. A total of 39 block grammars were developed on a training set of 20 title and nontitle pages of articles from the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. The entities recognized were similar to those listed above for the *IBM Journal,* and the number of grammars per entity ranged from one to four. These pages were also processed correctly most of the time.

To determine the performance of the method on test data, six title and six nontitle pages were chosen *at random* from each of the *IBM Journal* and *IEEE PAMI*. The test-document characteristics in terms of the total number of blocks and the number of leaf-level blocks at various levels of the X-Y tree are shown in Table II. The *IBM Journal* pages were

TABLE II
TEST DOCUMENT CHARACTERISTICS

Test Doc. Number	Total # blocks	Leaf Nodes @ Level				
		1	2	3	4	5
1	17	0	1	3	0	5
2	22	0	4	9	0	0
3	15	0	4	2	0	0
4	20	0	4	7	0	0
5	23	0	1	3	0	9
6	-	-	-	-	-	-
6*	22	0	4	9	0	0
7	20	0	1	13	0	0
8	18	0	1	11	0	0
9	17	1	1	10	0	0
10	13	0	1	6	0	0
11	18	0	1	2	0	7
12	24	0	1	17	0	0

TABLE III
SEGLABEL PERFORMANCE

Test Doc. Number	# Gram. Applied	Time (min:sec)	% Area Labeled	Missed Labels†
1	69	24:18	69	T A Fr B K C
2	12	4:14	100	*None*
3	42	16:10	73	*None*
4	11	4:08	100	*None*
5	61	23:13	91	T A Fr B K C S
6	-	F	-	-
6*	15	5:00	100	*None*
7	9	3:27	100	*None*
8	13	4:20	100	*None*
9	10	3:59	100	Fm(3)
10	8	3:20	100	S(3)
11	13	4:48	100	S(3) Fg(2) Fm
12	11	3:44	100	*None*

† T - Title, A - Author, Fr - Footer, B - Abstract, K - Keywords, C - Copyright, S - Section-title, Fm - Formula, and Fg - Figure.

all scanned on the MicroTek scanner, but some of the *IEEE-PAMI* pages were obtained from the IEEE Compact Disk publication-image database. These pages were deskewed using an experimental program obtained from Olivetti, Italy. The results on *IEEE PAMI* are shown in Table III (the results on the *IBM Journal* are comparable). The algorithm failed completely only on one page from the CD database (Test Document 6 in the table), which had a high residual skew. When this page was scanned with careful alignment (Test Document 6*), it was processed correctly. The errors made by the program could easily be avoided by modifying the parameter tables, but several cycles of design and testing (on successive batches of *previously unseen* pages) may be required to achieve acceptable performance.

All of the grammars were precompiled; even so, processing on a Sun 3/60 takes more than 3 min for each new page (Table III). The bulk of this time (>70%) is taken by recursive profile extraction and related disk access. The use of Unix shell scripts instead of direct coding also contributed to excessive input/output time. Optimized compilation of all our C programs improved the performance by 50%.

Considerable additional analysis of these experiments, including detailed examination of the errors and run-time characteristics, may be obtained from [29].

X-Y-Tree Statistics

There are many different X-Y trees that can be associated with a technical page. The syntactic approach described in this paper extracts a *logical* X-Y tree whose blocks coincide with the entities (title, paragraphs, etc) that are meaningful to the reader. Usually, the logical X-Y tree of a page is unique. In a *transition-cut* X-Y tree, a "cut" (imaginary line marking beginning and end of segment) is placed at each 0-1 and 1-0 transition in the projection profile with the blocks corresponding to each run of 1's and 0's forming a node in the X-Y tree. Hence, the segmentation of a page of text results in a set of lines, which are further divided into words (by vertical segmentation), and then into characters and character fragments. Here, the segmentation process stops only after all the block nodes around which imaginary boxes can be drawn are extracted. Such segmentation may be carried out in a preprocessing step to OCR, e.g., to deskew the scanned image [3].

TABLE IV
COMPARISON OF X-Y TREE STATISTICS

Level	Syntactic		Transition-cut	
	# Nodes	Node Area	# Nodes	Node Areas
0	1	8251776	1	8251776
1	7	5296512	46	4018560
2	10	3980713	3278	1882817
3	9	2709645	3514	1132516
4	—	—	260	6754

The transition-cut X-Y tree is defined in the same way for any page and carries little information about typesetting conventions that are generic or specific to a given publication. As such, it provides a good point of reference to the level of abstraction achieved by a logical X-Y tree, as illustrated in Table IV for a page from the *IBM Journal*. Level 0 in the table corresponds to the whole page. At level 1, seven significant blocks are extracted in the syntactic approach, whereas the transition-cut method finds 46 gray nodes. At higher tree levels, even larger discrepancies in the number of nodes are found according to the two schemes [31].[1] A bottom-up analysis can be carried out using the leaf-block of the transition-cut X-Y tree. It is clear from the table that as a graphic entity, the storage overhead of a logical X-Y tree is minimal compared with the size of the page image.

Data Compression

The objective of this experiment was to ascertain the loss of compression in storing a page compressed as a whole versus compressed block by block. A sample of 65 pages from the *IBM Journal* and *PAMI* were compressed using the CCITT Group 4 scheme. The average compression achieved for entire pages was 10.9:1 compared with 9:1 for the blocks. These results show that there is little loss incurred in storing a page compressed at the block level. Thus, after page analysis, we could store individual objects (labeled by syntactic analysis), as might be desirable in certain applications.

IV. CONCLUSION

We have demonstrated that on printed pages, image segmentation and labeling can be successfully combined. The

[1] One of the major advantages of the syntactic approach is that page analysis can be terminated at any predetermined level with a complete understanding of page layout up to that point. In the transition-cut method, the page must be completely processed before any useful information can be derived from its X-Y tree.

X-Y tree representation allows recursive transformations from the 2-D domain to a set of string-parsing problems. Analysis of the strings by syntactic techniques decomposes the page image into nested blocks labeled according to their functional role.

The method differs from conventional syntactic approaches because the grammars themselves form a hierarchy. The labels obtained from the analysis at one level determine the grammars to be applied at the next level. Furthermore, the string data to which the child grammars are applied themselves depend on the results of the analysis at the previous level; they can be extracted from the page image only after the string at the level above has been segmented. Further, low-level grammars for paragraphs, lines, and footers could be reused across many different publications.

The provision of multiple grammars for some or all labels provides the opportunity for backtracking to correct mistakes. The resulting algorithm is similar to those used for searching AND-OR trees. If a page can be parsed correctly at all levels, then it is accepted; otherwise, it is rejected. In some applications, however, it is desirable to obtain the largest fraction of the page that can be labeled. This can be done efficiently with a branch-and-bound version of the above algorithm.

In order to incorporate our methods into practical systems, two key issues must be resolved: acceleration of the development of block grammars for different publications and reduction of the time required to process a page. We believe the performance can be improved substantially since very little attempt was made in the current implementation to optimize timings. However, some automation in the specification of grammars is essential if the proposed method is to find widespread use. To further reduce the time necessary to develop new grammars, we are now attempting to specify page layout in the form familiar to page editors and printers and develop the programs necessary for translating this form into the current tabular form. We are also examining the possibility of obtaining additional grammars from page-formatter and photo-composer macros for specific styles. This should lead to consistent analysis of an entire article (and, perhaps, eventually an entire journal) instead of only an isolated page.

As the number and generality of the grammars available to the system grows, a larger fraction of pages from new types of publications should be parsed correctly. The underlying assumption here is that pages accepted by the system are correctly parsed even if the parameters fall near the bounds of their permissible range, i.e., that the system tends to fail before it yields an incorrect interpretation. The ranges can then be reset, provided that they do not change the interpretation. This is a simple form of learning.

With regard to increased throughput, we are comparing different architectures, including signal-processing application-specific chips and array processors for speeding up profile extraction. Once that bottleneck is eliminated, we will consider streamlining the lex and yacc processors, or even recoding their function, to speed up parsing. Another avenue open to us is porting the analysis tools to a loosely coupled multiprocessor system where each processor would be responsible for the analysis of a particular node.

V. Acknowledgment

The authors acknowledge the contributions to the project of Professors S. Stoddard (now at Tandem Computers), T. Spencer, and M. Ancona and former and current students A. Bonebrake, A. Burkle, M. Choi, K. Damour, N. Ferraiuolo, J. Kanai, J. Li, M. Maculotti, N. Shirali, M. Thomas, and J. Yu.

VI. References

[1] R. Ascher, G. Koppelman, M. Miller, G. Nagy, and G. Shelton Jr, "An interactive system for reading unformatted printed text," *IEEE Trans. Comput.*, vol. C-20, no. 12, pp. 1527–1543, Dec. 1971.

[2] E. Backer and E. S. Gelsema, Eds., in *Proc. Int. Conf. Patt. Recogn.* (The Hague), Sept. 1992.

[3] H. S. Baird, "The skew angle of printed documents," in *Advance Printing Symp. Summaries, SPSE's 40th Ann. Conf. Symp. Hybrid Imaging Syst.*, May 1987, pp. 21–24.

[4] H. S. Baird, H. Bunke, and K. Yamamoto, Eds., *Structured Image Analysis.* New York: Springer Verlag, 1992.

[5] D. P. D'Amato, W. E. Blanz, B. E. Dom, and S. N. Srihari, Eds., in *Proc. SPIE (Int Soc. Opt. Engr.)—Machine Vision Applications Character Recogn. Ind. Inspection* (San Jose, CA), Feb. 10–12, 1992, vol. 1661.

[6] A. Dengel and G. Barth, "ANASTASIL: A hybrid knowledge-based system for document layout analysis," in *Proc. 11th Int. J. Conf. Artificial Intell.* (Detroit), Aug. 1989, pp. 1249–1254.

[7] F. Jenkins and J. Kanai, "A keyword-indexed bibliography of character recognition and document analysis," Inform. Sci. Res. Inst., Univ. Nevada, Las Vegas, Mar. 4, 1992.

[8] S. C. Johnson, "Yacc: Yet another compiler-compiler," Comp. Sci. Tech. Rep. 32, Bell Laboratories, Murray Hill, NJ, 1975.

[9] J. Kanai, "Text line extraction using character prototypes," in *Preproc. IAPR Workshop Structural Syntactic Patt. Recogn.* (Murray Hill, NJ), June 1990, pp. 182–191.

[10] R. Kasturi and L. O'Gorman, Eds., *Machine Vision Applications,* vol. 5, no. 3 (special issue on Document Image Analysis Techniques), Summer 1992.

[11] ——, "Document image analysis: A bibliography," in *Machine Vision Applications,* vol. 5, no. 3 (special issue on Document Image Analysis Techniques), pp. 231–243, Summer 1992.

[12] D. Klarner and S. Magliveras, "The number of tilings of a block with blocks," *Euro. J. Combinatorics,* vol. 9, pp. 317–330, 1988.

[13] M. E. Lesk, "Lex—A lexical analyzer generator," Comp. Sci. Tech. Rep. 39, Bell Lab., Murray Hill, NJ, 1975.

[14] S. Liebowitz, M. Lipshutz, and C. Weir, "Document structure interpretation by integrating multiple knowledge sources," in *Proc. Symp. Document Anal. Inform. Retrieval,* Mar. 1992, pp. 58–76.

[15] G. Lorette (Ed.), in *Proc. First Int. Conf. Document Anal. Recogn.* (Rennes, France), Oct. 1991.

[16] G. Nagy and S. Seth, "Hierarchical representation of optically scanned documents," in *Proc. 7th Int. Conf. Patt. Recogn.* (Montreal, Canada), 1984, pp. 347–349.

[17] G. Nagy, S. Seth, and S. D. Stoddard, "Document analysis with an expert system," in *Pattern Recognition in Practice II* (E. S. Gelsema and L. Kanal, Eds.). New York: Elsevier Science, 1986.

[18] G. Nagy, J. Kanai, M. S. Krishnamoorthy, M. Thomas, and M. Viswanathan, "Two complementary techniques for digitized document analysis," in *Proc. ACM Conf. Document Processing Syst.* (Santa Fe), Dec. 1988, pp. 169–176.

[19] G. Nagy, "Document analysis and optical character recognition," in *Progress in Image Analysis and Processing* (V. Cantoni, L. P. Cordella, S. Levialdi, and G. Sanniti di Baja, Eds.). Singapore: World Scientific, 1990, pp. 511–529.

[20] G. Nagy and M. Viswanathan, "Dual representation of segmented technical documents," in *Proc. First Int. Conf. Document Anal. Recogn.* (Rennes, France), Oct. 1991, pp. 141–151.

[21] G. Nagy, "Towards a structured-document-image utility," in *Structured Image Analysis* (H. S. Baird, H. Bunke, and K. Yamamoto, Eds.). New York: Springer Verlag, 1992, pp. 54–69.

[22] G. Nagy, S. Seth, and M. Viswanathan, "A prototype image analysis system for technical journals," *Computer,* vol. 25, no. 7 (special issue on Document Image Analysis Systems), pp. 10–22, July 1992.

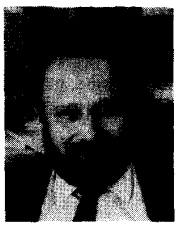
[23] T. A. Nartker (Editor), in *Proc. Symp. Document Anal. Inform. Retrieval* (Inform. Sci. Res. Inst., Univ. Nevada, Las Vegas), Mar. 1992.

[24] L. O'Gorman and R. Kasturi (Eds.), *Computer,* vol. 25, no. 7 (special issue on Document Image Analysis Systems), July 1992.

[25] T. Pavlidis and S. Mori (Eds.), *Proc. IEEE*, vol. 80, no. 7 (special issue on Optical Character Recognition), July 1992.
[26] S. V. Rice, J. Kanai, and T. A. Nartker, "A report on the accuracy of OCR Devices," Inform. Sci. Res. Inst., Univ. Nevada, Las Vegas, Mar. 4, 1992.
[27] E. Tanaka, "Theoretical aspects of syntactic pattern recognition," in Memo. Graduate Sch. Sci. Technol. (Kobe Univ.), 1992, pp. 111–126, vol. 10-A.
[28] M. Viswanathan and M. S. Krishnamoorthy, "A syntactic approach to document segmentation," in *Structural Pattern Analysis* (R. Mohr, T. Pavlidis, and A. Sanfeliu, Eds). Singapore: World Scientific, 1989, pp. 197–215.
[29] M. Viswanathan, "A syntactic approach to document segmentation and labeling," Ph.D. thesis, Dept. Elect., Comput. Syst. Eng., Rensselaer Polytechnic Inst., Dec. 1990.
[30] M. Viswanathan, "Analysis of scanned documents—A syntactic approach," in *Structured Image Analysis* (H. S. Baird, H. Bunke, and K. Yamamoto, Eds.). New York: Springer Verlag, 1992, pp. 115–136.
[31] M. Viswanathan and G. Nagy, "Characteristics of digitized images of technical articles," in *Proc. Int. Soc. Opt. Eng. (SPIE)—Machine Vision Applications Character Recogn. Ind. Applications*, 1992, pp. 6–17, vol. 1661.
[32] J. Yu, "Document analysis using x-y tree and rule-based system," Master's thesis, Dept. Elect. Comput. Syst. Eng., Rensselaer Polytechnic Inst., Troy, NY, Dec. 1986.

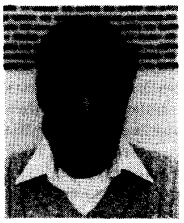
Mukkai Krishnamoorthy received the B.E. degree (with honors) from Madras University in 1969 and the M.Tech. degree in electrical engineering and the Ph.D. degree in computer science in 1971 and 1976, respectively, from the Indian Institute of Technology, Kanpur, India.

From 1976 to 1979, he was an Assistant Professor of Computer Science at the Indian Institute of Technology, Kanpur. From 1979 to 1985, he was an Assistant Professor of Computer Science at Rensselaer Polytechnic Institute, Troy, NY, and since 1985, he has been an Associate Professor of Computer Science at Rensselaer. His research interests are in the design and analysis of combinatorial and algebraic algorithms and programming environments.



George Nagy (SM'72) received the B.Eng. and M.Eng. degrees from McGill University, and the Ph.D. in electrical engineering from Cornell University in 1962 (on neural networks).

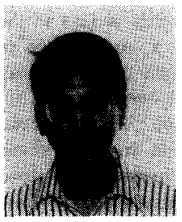
For the next ten years, he conducted research on various aspects of pattern recognition and OCR at the IBM T. J. Watson Research Center, Yorktown Heights, NY. From 1972 to 1985, he was Professor of Computer Science at the University of Nebraska-Lincoln, and worked on remote sensing applications, geographic information systems, computational geometry, and human-computer computer interfaces. Since 1985, he has been Professor of Computer Engineering at Rensselaer Polytechnic Institute. He has held visiting appointments at the Stanford Research Institute, Cornell, the University of Montreal, the National Scientific Research Institute of Quebec, the University of Genoa and the Italian National Research Council in Naples and Genoa, AT&T Bell Laboratories, IBM Almaden, McGill University, and the Institute for Information Science Research at the University of Nevada. In addition to document image analysis and character recognition, his interests include solid modeling, finite-precision spatial computation, and computer vision.



Sharad Seth (SM'82) received the B.Eng. degree from the University of Jabalpur, India, the M.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana-Champaign.

He is a professor in the Computer Science and Engineering Department at the University of Nebraska-Lincoln. Besides document analysis, his research has been in testing and testable design of microelectronic circuits.

Dr. Seth serves on the editorial board of the *Journal of Electronic Testing: Theory and Applications* (JETTA). He is a member of the IEEE Computer Society, ACM, and the VLSI Society of India.



Mahesh Viswanathan (M'90) received the B.Sc. degree in physics from Loyola College, Madras, India, in 1980 and the B.E. degree in electrical engineering from the Indian Institute of Science, Bangalore, India, in 1984. He received the M.S. degree in electrical engineering from San Diego State University, San Diego, CA, in 1986 with a thesis entitled "Matrix Quantization of Homomorphically Processed Images" and received the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1990 with a thesis entitled "A Syntactic Approach to Document Segmentation and Labeling."

He is currently an Advisory Staff Programmer with IBM Pennant Systems, Boulder, CO. His research interests include printing and document analysis.