Document Image Decoding Using Markov Source Models

Gary E. Kopec and Phil A. Chou

Abstract—**This paper describes a communication theory approach to document image recognition, patterned after the use of hidden Markov models in speech recognition. In general, a document recognition problem is viewed as consisting of three elements— an image generator, a noisy channel and an image decoder. A document image generator is a Markov source (stochastic finite-state automaton) that combines a message source with an imager. The message source produces a string of symbols, or text, that contains the information to be transmitted. The imager is modeled as a finite-state transducer that converts the one-dimensional message string into an ideal two-dimensional bitmap. The channel transforms the ideal image into a noisy observed image. The decoder estimates the message, given the observed image, by finding the a posteriori most probable path through the combined source and channel models using a Viterbi-like dynamic programming algorithm. The proposed approach is illustrated on the problem of decoding scanned telephone yellow pages to extract names and numbers from the listings. A finite-state model for yellow page columns was constructed and used to decode a database of scanned column images containing about 1100 individual listings. Overall, 99.5% of the listings were correctly recognized, with character classification rates of 98% and 99.6%, respectively, for the names and numbers.**

Keywords—**document recognition, text recognition, image decoding, stochastic grammars, Markov sources**

I. Introduction

The success of automatic speech recognition systems based on hidden Markov models (HMMs) [12] and other stochastic grammar frameworks has motivated recent attempts to apply similar methods to document image recognition problems [3-4], [7–9], [27]. The central issue in adapting any type of speech technology to image analysis, of course, is that speech is a one-dimensional signal whereas images are two-dimensional. In the case of HMMs, two approaches to technology transfer have been explored. The first approach is to focus on a subset of image analysis problems that may be viewed as actually one-dimensional. An example of such a problem is transcription of a single line of text that has been excised (through other means) from a page image [3–4], [7–8], [27]. While encourging results have been obtained on this problem, the basic strategy of searching for one-dimensional subproblems is unlikely to lead to a general framework for document recognition.

The second approach to developing two-dimensional analogs of HMMs is to extend the fundamental grammar concepts on which HMMs are based to make them powerful enough to describe two-dimensional structure. Previous attempts along these lines have generalized string grammar formalisms by replacing the notion of a one-dimensional phrase with that of a two-dimensional rectangular region [9], [25]. A typical production rule in such a grammar describes how a region corresponding to some non-terminal is formed by combining a pair of horizontally or vertically abutting sub-regions. Rectangular subregions may be combined only if they do not overlap.

One disadvantage of the region-based approach to image grammars is that the two-dimensional counterparts to regular (finite-state) string grammars are not particularly useful for image modeling [9]. As a result, most effort has been directed towards context-free two-dimensional grammars. Context-free grammars are relatively expensive computationally, since parsing time, in general, is $\mathbf{O}(n^3)$ in the number of terminal symbols (e.g. pixels), compared with $\mathbf{O}(n)$ for regular grammars.

A second problem with current region-based grammars is the requirement that rectangular subregions do not overlap. Complex graphical notations such as music are not well described in terms of non-overlapping rectangular components. Even for text and equations, bounding box overlap occurs as a result of typographic letterspacing rules that allow characters to have negative sidebearings.

This paper presents an alternative approach to using formal grammars for document image modeling and recognition that addresses the above problems. The basis of our approach is a stochastic model of image generation that incorporates a deterministic finite state machine for sequentially imaging glyphs onto the plane. The finite state imager is procedural and resembles, in spirit, a greatly simplified version of PostScript.[1] In particular, it employs the sidebearing model of character positioning that is widely used in digital typography. The sidebearing model does not require that the glyphs have disjoint bounding boxes. Our only requirement is that the glyphs have disjoint support. In the language of parsing two-dimensional image grammars, this means that overlapping sub-regions may be combined as long as their black pixels do not overlap. Indeed, bounding boxes and rectangles play no particular role in our approach, a consequence of a simple normalization of the likelihood function used for template match scoring. Furthermore, the imager does not restrict the displacement from one glyph to the next to be horizontal or vertical. Rather, displacements may be arbitary two-dimensional vectors. This finite state model of image generation appears to be powerful enough to describe a wide variety of image types, including kerned and accented text, sheet music, and simple equations.

A key element of our approach is that maximum a posteriori (MAP) decoding, or parsing, of the observed image into an underlying finite state process can be accomplished using a segmental Viterbi algorithm (a dynamic programming algorithm), whose complexity is linear in the number of image pixels. Image decoding is analogous to Viterbi speech decoding and involves computing a set of recursively-defined likelihood functions at

G. E. Kopec and P. A. Chou are with the Information Sciences and Technologies Laboratory, XEROX Palo Alto Research Center, Palo Alto, CA, 94304.

[1]PostScript is a registered trademark of Adobe Systems Incorporated.

each point of the image plane. A potential difficulty that arises because of the two-dimensional nature of our imaging model is that the sequence in which the Viterbi algorithm should compute the likelihoods is not immediately clear. The root of this problem, which does not appear in the context of one-dimensional HMMs, is that the imager is allowed to place the next glyph at an arbitrary two-dimensional position relative to the current glyph (e.g., above and to the left, below and to the right, etc.). We solve this sequencing problem using concepts and methods from linear scheduling theory for regular iterative algorithms, as developed in the VLSI literature. Indeed, the decoding algorithm is quite regular and conceivably could be implemented in hardware as a systolic array.

The rest of this paper is organized as follows. Section II presents an overall formulation of the document image recognition problem in terms of classical communication theory. The three main elements in this formulation— image source, channel and decoder— are discussed in Sections III, IV and V, respectively. Section VI describes the application of the proposed method to the problem of decoding scanned telephone yellow pages to extract names and numbers from the listings. Section VII contains a final discussion and summary. Finally, the Appendix describes an algorithm for scheduling image decoders.

II. Problem Formulation

Fig. 1 shows a formulation of the document recognition problem based on classical communication theory, patterned after a similar formulation of speech recognition [6]. A stochastic *message source* selects a finite string M from a set of candidate strings according to a prior probability distribution $\Pr\{M\}$. A message may be a plain text string, text with embedded layout or logical structure tags (e.g. LaTeX [18] or SGML [10]), or any other encoding of information into a linear sequence of discrete symbols.

The *imager* converts the message into an ideal binary image $Q = \{q_i \mid i \in \Omega\}$, drawn from a universe of binary images whose pixels are indexed by Ω. The pixel values $q_i \in \{0, 1\}$, where the foreground color (usually black) is denoted 1 and the background color (usually white) is 0. Throughout this paper, we will assume that images are rectangular, so that Ω is the integer lattice $[0, W) \times [0, H)$, where W and H are the image width and height, respectively.[2] Ω may represent a single document page or multiple pages arranged in a scroll. Collectively, the message source and imager form the *image source*.

The *channel* maps the ideal image into an observed image $Z = \{z_i \mid i \in \Omega\}$ by introducing distortions due to printing and scanning, such as skew, blur and additive noise.

The *decoder* receives image Z and produces an estimate $\hat{M}$ of the original message. The decoder achieves minimum probability of error if it chooses $\hat{M}$ according to the maximum a posteriori (MAP) criterion

$$\Pr\left\{\hat{M} \mid Z\right\} = \max_M \Pr\{M \mid Z\} \tag{1}$$

[2]We use an image coordinate system in which x increases to the right, y increases downward, and the upper left corner is at $x = y = 0$.

where

$$\Pr\{M \mid Z\} = \frac{\Pr\{M, Z\}}{\Pr\{Z\}}.$$

The joint probability $\Pr\{M, Z\}$ represents the combined effects of the imager and channel. These can be separated by noting that

$$\begin{aligned}\Pr\{M, Z\} &= \sum_Q \Pr\{Z, Q, M\} \\ &= \sum_Q \Pr\{Z \mid Q, M\} \Pr\{Q, M\}.\end{aligned}$$

If we assume that the imager and channel act independently, then we can write

$$\Pr\{M, Z\} = \sum_Q \Pr\{Z \mid Q\} \Pr\{Q, M\}. \tag{2}$$

Since $\Pr\{Z\}$, the prior probability of observing Z, does not depend on M, it is irrelevant to choosing $\hat{M}$ and may be omitted. More generally, the MAP decision rule can be implemented by maximizing any monotone function of the right hand side of (2). In particular, it will be turn out to be convenient to maximize the normalized probability

$$\tilde{\Pr}\{M, Z\} = \sum_Q \tilde{\Pr}\{Z \mid Q\} \Pr\{Q, M\} \tag{3}$$

where

$$\tilde{\Pr}\{Z \mid Q\} \equiv \frac{\Pr\{Z \mid Q\}}{\Pr\{Z \mid Q_0\}} \tag{4}$$

and $\Pr\{Z \mid Q_0\}$ is the probability of observing Z given that Q is the all-white background image Q_0.

As (3) suggests, there are three main problems to be solved in the design of a document image recognition system. First, it necessary to develop a model of the image source, in order to compute $\Pr\{Q, M\}$. Second, it is necessary to understand the characteristics of the channel, in order to compute $\tilde{\Pr}\{Z \mid Q\}$. Finally, a search algorithm is necessary in order to perform the maximization of (3) over M in a computationally efficient manner. As we will see, obtaining an efficient decoding algorithm will involve additional assumptions and approximations to simplify (3). The next three sections of this paper present solutions to the three document recognition system design problems.

III. Image Source Model

The image source model combines the effects of the message source and the imager. Our approach to source modeling is motivated by the *sidebearing model* [23] of character shape description and letterspacing. This model is used widely in digital typography and is formalized in page description languages such as PostScript [1]. This section very briefly reviews character shape and letterspacing and then formally defines our image source model. It concludes by presenting a simple model for a column of text that illustrates many of the concepts introduced.

Fig. 2 depicts a simplified version of the sidebearing model for describing character shape and positioning. The shape of a character is defined in terms of a local coordinate system aligned so that the origin of the character is at $(0, 0)$. The *set width* of a character is the *vector* displacement $\vec{\Delta} = (\Delta x,\, \Delta y)$ from the character origin to the point at which the origin of the next character is normally placed when imaging consecutive characters of a word. In most Indo-European alphabets, including Roman, $\Delta x > 0$ and $\Delta y = 0$. In other writing systems, however, Δx may be negative (e.g. Semitic) or Δy may be non-zero (e.g. Oriental glyphs). When $\Delta y = 0$, the character origins in a line of text are colinear and define the *baseline* of the text line.

The *bounding box* of a character is the smallest rectangle, oriented with the character coordinate axes, that just encloses the character. The *width* of a character is the corresponding dimension w of the bounding box. The *left sidebearing* is the horizontal displacement λ from the origin of the character to the left edge of the bounding box.[3] Similarly, the *right sidebearing* is the horizontal displacement ρ from the right edge of the bounding box to the origin of the next character. The *depth below baseline* is the vertical distance μ from the character origin to the bottom of the character bounding box, while the *height above baseline* (not shown in Fig. 2) is defined analogously. It is often convenient to define the baseline height and depth of a *font* to be the maximum of the corresponding metrics of its constituent characters.

The horizontal component of set width is related to the sidebearings and bounding box width by the relation

$$\Delta x = \lambda + w + \rho. \tag{5}$$

One or both of the sidebearings may be negative. For example, the left sidebearing of 'j' in Fig. 2(b) is negative, with the result that the bounding boxes of the 'b' and 'j' overlap. However, as Fig. 2(b) suggests, typefaces are typically designed so that even if adjacent character bounding boxes overlap, the characters themselves do not.

The sidebearing model of character shape and letterspacing, while primarily motivated by conventional letterform typography, is useful for a much wider class of images. For example, Adobe Sonata is a PostScript font used for printed music notation. As in ordinary text fonts, each Sonata character has a set of metric parameters, including set width, sidebearings and height above baseline. Unlike text, however, music glyphs are seldom placed one after another on a line. The Sonata font metrics are primarily used to control glyph alignment and the location of the current point after printing each character and were carefully designed to simplify the development of music typesetting programs. Our approach to image source modeling is motivated by the generality of the sidebearing model.

A. Markov Source Model

The structure of a set of images is captured formally by modeling image generation as a Markov source, as depicted in Fig. 3. A Markov source consists of a finite set of states (nodes, vertices) $\mathcal{N}$ and a set of directed transitions (branches, edges) $\mathcal{B}$. Each transition t connects a pair of states, L_t and R_t, that are called, respectively, the *predecessor* (left) state and the *successor* (right) state of t. Two distinguished members of $\mathcal{N}$ are the initial state n_I, and the final state n_F. It is assumed that no transition has n_F as its predecessor, so that the final state is a *trap* state.

With each transition t is associated a 4-tuple of attributes, $(Q_t, m_t, a_t, \vec{\Delta}_t)$, where Q_t is the *template*, m_t is the *message string*, a_t is the *transition probability* and $\vec{\Delta}_t$ is the vector *displacement* of t (analogous to set width for characters). The template Q_t is defined over the entire image plane Ω, although normally its support (set of non-zero, foreground pixels) will be localized within a small region near the origin of the local template coordinate system. Associating displacement with transitions, rather than with templates, allows context-dependent spacing rules (e.g. pairwise kerning) to be accommodated easily. A *path* π in a Markov source is a sequence of transitions $t_1 \ldots t_P$ for which $L_{t_1} = n_I$ and

$$R_{t_i} \;=\; L_{t_{i+1}} \tag{6}$$

for $i = 1, \ldots, P-1$. A *complete path* is a path for which $R_{t_P} = n_F$. A *cycle* or *loop* is a sequence of transitions $t_1 \ldots t_P$ for which $L_{t_1} = R_{t_P}$. Note that a loop does not necessarily include a transition to/from n_I, and thus is not necessarily a path.

Associated with each path π is a composite message

$$M_\pi \;=\; m_{t_1} \cdots m_{t_P} \tag{7}$$

formed by concatenating the message strings of the transitions of the path. The set of messages generated by complete paths through an image source is a regular language and the source model itself is a finite-state automaton that accepts that language. A Markov source defines a probability distribution on complete paths by

$$\Pr\{\pi\} \;=\; \prod_{i=1}^{P} a_{t_i} \tag{8}$$

and induces a probability distribution on messages by

$$\Pr\{M\} \;=\; \sum_{\pi \mid M_\pi = M} \Pr\{\pi\} \tag{9}$$

where M_π is the message associated with path π and the summation is taken over complete paths.

Also associated with each path π is a sequence of positions $\vec{x}_1 \ldots \vec{x}_{P+1}$ recursively defined by

$$\vec{x}_1 = \vec{0} \tag{10}$$

$$\vec{x}_{i+1} = \vec{x}_i \;+\; \vec{\Delta}_{t_i}, \tag{11}$$

where $\vec{x}_{P+1}$ is introduced for convenience, and a composite image Q defined by

$$Q_\pi \;=\; \bigcup_{i=1}^{P} Q_{t_i}[\vec{x}_i] \tag{12}$$

where $Q[\vec{x}]$ denotes Q shifted so that the original of its local coordinate system is located at $\vec{x}$. For a path π we will define

$$\vec{\Delta}_\pi \;=\; \vec{x}_{P+1} \;=\; \sum_{i=1}^{P} \vec{\Delta}_{t_i} \tag{13}$$

[3] In is possible to generalize sidebearings to be vector displacements. Vector sidebearing are supported by PostScript, for example, but are seldom used.

to be the displacement of the path and will let Δx_π and Δy_π denote the x and y components of $\vec{\Delta}_\pi$, respectively.

Motivated by the observation that fonts are typically designed so that character glyphs do not overlap in text strings we will require our sources to be designed so that

$$Q_{t_i}[\vec{x}_i] \cap Q_{t_j}[\vec{x}_j] = \emptyset \tag{14}$$

for $i \neq j$, for every path π. The importance of this requirement will be clear shortly.

The imaging model defined by (12) has a simple interpretation. Imagine a collection of transparent plastic sheets, on each of which is painted one copy of some template Q, with the origin of Q at the center of the sheet. For each transition t_i of path π, a sheet containing Q_{t_i} is placed on top of a stack of transparencies with the center of the sheet aligned at $\vec{x}_i$. The complete stack of transparencies defines Q_π. Note that the individual painted sheets are likely to overlap significantly. It is also permissible for the template bounding boxes to overlap; bounding boxes play no particular role in our imaging model. However, by (14), the painted regions on the sheets are required to be disjoint.

An image source defines a relation between message strings and images via an underlying path and (7) and (12). While the primary concern in document recognition is recovering messages from observed images, a source model can also be used to generate an image of a specified message. As noted above, an image model defines a finite-state acceptor for the language of messages generated by the model. Thus, given a message string M, it is straightforward to determine if there is a complete path π for which $M_\pi = M$, and, if such a path exists, to find one. The image Q_π defined by (12) is then an image of M.

If the source model defines a *deterministic* [11] acceptor for the message language, the process of message imaging admits a simple procedural interpretation, similar to the "turtle geometry" [2] interpretation for strings generated by Lindenmayer systems in computer graphics [21]. Imagine an imager automaton that draws an output image under the control of an input message "program". The structure of the imager is defined by a finite-state source model. The imager begins at location $(0, 0)$ of the output image plane in internal state n_I. The imager examines the first input character, compares it with the message labels on the transitions from n_I, and selects the branch whose message matches the input character. If the selected branch template is non-null, the imager draws a copy of the template on the image plane, with the template origin aligned with the imager's current image position. The imager then increments its image position by the branch displacement and updates its internal state to the successor node of the selected branch. This process is repeated for each character of the input message.

The relationship between M and Q induced by a Markov source can be expressed in terms of the set of complete paths through the source as

$$\Pr\{Q, M\} = \sum_\pi \Pr\{Q, M, \pi\} \tag{15}$$

$$= \sum_\pi \Pr\{Q \mid M, \pi\} \Pr\{M \mid \pi\} \Pr\{\pi\}. \tag{16}$$

Since the mappings from π to M_π and from π to Q_π defined by (7) and (12) are deterministic, it is easy to see that (16) can be written

$$\Pr\{Q, M\} = \sum_{\pi \mid M_\pi = M} \Pr\{Q \mid \pi\} \Pr\{\pi\}. \tag{17}$$

Finally, if we substitute (17) into (3) we can show that

$$\tilde{\Pr}\{M, Z\} = \sum_{\pi \mid M_\pi = M} \tilde{\Pr}\{Z \mid Q_\pi\} \Pr\{\pi\}. \tag{18}$$

B. Text Column Example

Text transcription is concerned with decoding images to extract simple text strings, without formatting or logical structure tags. A simple text column consists of a vertical sequence of text lines, alternating with whitespace. Horizontally, a text line is a sequence of characters typeset according to the sidebearing model shown in Fig. 2. Fig. 4 shows a simple source model for a column of text. State n_1 corresponds to the creation of vertical whitespace. Each time branch t_1 is traversed, the imager moves down one row without drawing anything on the output page, since no image template is associated with t_1. At some point, the imager reaches the top of a text line and follows branch t_2. The displacement $(0, B)$ of t_2 moves the cursor down to the text baseline; B is the font height above baseline.

State n_2 represents the creation of a horizontal text line. The self-transitions from n_2 to n_2 are of two types. The F transitions t_i that are labelled with image templates Q_i and single-character message strings "`c_i`" are used to image individual "printing" characters. The horizontal displacements of these branches are the character set widths W_{t_i}. Branches t_3 and t_4 have blank templates associated with them and represent whitespace. Branch t_3 represents a whitespace of minimal (1 pixel) width and is used for fine spacing adjustment. Branch t_4 corresponds to a real space "character" of font-dependent width W_s and is labeled with a space message " ". At the end of a text line the imager traverses t_5 ("line feed") and enters "carriage return" state n_3. The message on t_5 is the newline character "`\n`". The vertical displacement associated with t_5 is the font depth D. Each traversal of branch t_6 moves the imager left by one pixel. Finally, transition t_7 returns the imager to state n_1 and the process is repeated for the next text line. After the last text line has been created, the imager traverses t_8 into final state n_F.

The transition probabilities indicated in Fig. 4 have been set manually, although our experience suggests that their exact values are not very important. In general, the most likely path through the network, given an observed image, is determined more by the template match scores $\tilde{\Pr}\{Z \mid Q_\pi\}$ in (18) than by the path prior probabilities $\Pr\{\pi\}$. The priors are important primarily when the image alone is insufficient to reliably distinguish between alternative possibilites (e.g. because of noise). Nevertheless, we expect eventually to estimate transition probabilities (as well as other numerical parameters) automatically from training data, using methods similar to those for training speech HMMs.

IV. Channel Model

The purpose of channel modeling is to derive an expression for the normalized probability $\tilde{\Pr}\{Z \mid Q\}$ for use in (3) or (18). Ultimately, we expect that effective channel models for document decoding will emerge from models of the physics of printing and imaging processes. A similar philosophy has recently motivated the development of parameterized defect models for training and evaluating OCR systems [5]. As a first step in this direction, and to illustrate the role of analytic channel modeling in overall decoder design, we consider the simple asymmetric bit-flip noise model shown in Fig. 5. This model is very simple and unlikely to be adequate as a general image degradation model. Nevertheless, as will be seen, very encouraging results have been obtained using it as the basis for decoding scanned images.

The asymmetric bit flip model assumes that each pixel of the ideal image Q is independently perturbed. The probability of a 1 (black) pixel in the ideal image Q surviving as a 1 in the observed image Z is α_1. Similarly, the probability of a 0 being observed as a 0 is α_0. The noise parameters are assumed to be constant over the image. With these assumptions, the probability of Z given Q can be expressed

$$\Pr\{Z \mid Q\} \;=\; \prod_{i|q_i=0} \alpha_0^{(1-z_i)}(1-\alpha_0)^{z_i} \times \prod_{i|q_i=1} \alpha_1^{z_i}(1-\alpha_1)^{(1-z_i)}. \tag{19}$$

In particular, if Q is the all-white background image Q_0, then

$$\Pr\{Z \mid Q_0\} \;=\; \prod \alpha_0^{(1-z_i)}(1-\alpha_0)^{z_i}. \tag{20}$$

The notation is simplified if channel modeling is discussed in terms of the log normalized probability

$$\mathcal{L}(Z \mid Q) \equiv \log \tilde{\Pr}\{Z \mid Q\} \tag{21}$$

$$= \log \frac{\Pr\{Z \mid Q\}}{\Pr\{Z \mid Q_0\}}. \tag{22}$$

Moreover, the decoder structure we develop in the next section will be implemented in terms of $\mathcal{L}(Z \mid Q)$. If (19) and (20) are substituted into (22) we have

$$\mathcal{L}(Z \mid Q) = \sum_{i|q_i=1} \left[z_i \log \frac{\alpha_1}{1-\alpha_0} \;+\; (1-z_i) \log \frac{1-\alpha_1}{\alpha_0} \right] \tag{23}$$

$$= \sum_{i|q_i=1} \left[\log \frac{1-\alpha_1}{\alpha_0} \;+\; z_i \log \frac{\alpha_0 \alpha_1}{(1-\alpha_0)(1-\alpha_1)} \right] \tag{24}$$

$$= \left[(\#\text{ of 1's in } Q) \cdot \log \frac{1-\alpha_1}{\alpha_0} \right] + \left[(\#\text{ of 1's in } Q \wedge Z) \cdot \log \frac{\alpha_0 \alpha_1}{(1-\alpha_0)(1-\alpha_1)} \right]. \tag{25}$$

Note that the second term on the right-hand side of (25) can be simply computed by counting the nonzero bits in the logical **and** of images Q and Z, while the first term is a class-dependent bias that is independent of Z.

The log normalized probability possesses an important decomposition property. If template Q is expressed as a disjoint union of sub-templates $Q_1 \ldots Q_P$, so that

$$Q \;=\; \bigcup_{i=1}^{P} Q_i \tag{26}$$

and

$$Q_i \,\cap\, Q_j \;=\; \emptyset \tag{27}$$

for $i \neq j$, then from (25) it is clear that

$$\mathcal{L}(Z \mid Q) \;=\; \sum_{i=1}^{P} \mathcal{L}(Z \mid Q_i). \tag{28}$$

Note that (27) expresses a constraint on the *supports*, not the *bounding boxes*, of Q_i and Q_j. Thus, (28) can be used to compute the log normalized probability (25) for a composite template even when the bounding boxes of the constituent templates overlap, as long as the constituent supports are disjoint. In particular, because of (14), the preconditions for decomposition (28) are satisfied if Q_π is the composite image associated with path π through a Markov source and we have

$$\mathcal{L}(Z \mid Q_\pi) \;=\; \sum_{i=1}^{P} \mathcal{L}(Z \mid Q_{t_i}[\vec{x}_i]). \tag{29}$$

As noted previously, the domain of definition of each template in a Markov source is the entire image plane Ω. Thus, the term $\mathcal{L}(Z \mid Q_{t_i}[\vec{x}_i])$ in (29) is the log normalized probability that the entire observed image Z results from transmitting the shifted template $Q_{t_i}[\vec{x}_i]$ through the channel. With this interpretation, (29) may seem peculiar in that each pixel of Z is accounted for by each template. However, the effect of normalization (4) is that the contribution to $\mathcal{L}(Z \mid Q_{t_i}[\vec{x}_i])$ due to template pixels with background color is 0. Thus, as long as the supports (foreground regions) of the templates are disjoint, each pixel of Z affects $\mathcal{L}(Z \mid Q_\pi)$ via at most one template.

V. Decoder

MAP decoding of an observed image Z with respect to a given image source involves finding a message $\hat{M}$ that maximizes $\tilde{\Pr}\{M, Z\}$ in (18). One problem with direct implementation of the MAP rule is that (18) involves a summation over all complete paths π for which $M_\pi = M$. Depending on the source model, forming this sum can be computationally prohibitive. One standard approach to the analogous problem in speech recognition is to approximate the sum by its largest term, so that

$$\tilde{\Pr}\{M, Z\} \;\approx\; \max_{\pi|M_\pi=M} \Pr\{\pi\}\, \tilde{\Pr}\{Z \mid Q_\pi\} \tag{30}$$

In that case,

$$\max_M \tilde{\Pr}\{M, Z\} \approx \max_M \max_{\pi|M_\pi=M} \Pr\{\pi\}\, \tilde{\Pr}\{Z \mid Q_\pi\} \tag{31}$$

$$= \max_\pi \Pr\{\pi\}\, \tilde{\Pr}\{Z \mid Q_\pi\}. \tag{32}$$

Thus, MAP decoding reduces to finding a complete path $\hat{\pi}$ that maximizes

$$\mathcal{L}(\pi, Z) \;\equiv\; \log \Pr\{\pi\} + \mathcal{L}(Z \mid Q_\pi) \tag{33}$$

and then constructing

$$\hat{M} \;=\; M_{\hat{\pi}}. \tag{34}$$

Approximation (30) will be exact if the messages and images associated with different paths are distinct; a source with this property is said to be *unifilar* [13]. More generally, experience in speech recognition suggests that, at least for simple models, the performance of decoders based on approximation (30) (often called *Viterbi* decoders) is similar to that of decoders based on the exact likelihood (e.g. *stack decoders* [13]).

If (8) and (29) are substituted into (33) we obtain

$$\mathcal{L}(\pi, Z) \;=\; \sum_{i=1}^{P} \left[\mathcal{L}(Z \mid Q_{t_i}[\vec{x}_i]) \;+\; \log a_{t_i}\right] \qquad (35)$$

as the MAP decision function for image decoding that is to be maximized over all complete paths π. The right-hand side of (35) is an additive combination of independent terms corresponding to the branches of π. This suggests using the standard approach of dynamic programming to determine the best path. Indeed, if we use (35) to define $\mathcal{L}(\pi, Z)$ for all paths (i.e. not just complete paths) and define

$$\mathcal{L}(n; \vec{x}) \;\equiv\; \max_{\{\pi \mid \vec{\Delta}_\pi = \vec{x} \,\wedge\, R_{t_P} = n\}} \mathcal{L}(\pi, Z) \qquad (36)$$

then it is not difficult to show that $\mathcal{L}(n; \vec{x})$ satisfies the recursive relationship

$$\mathcal{L}(n; \vec{x}) \;=\; \max_{t \mid R_t = n} \left\{\mathcal{L}(L_t; \vec{x} - \vec{\Delta}_t) \;+\; \log a_t \;+\; \mathcal{L}(Z \mid Q_t[\vec{x} - \vec{\Delta}_t])\right\} \qquad (37)$$

for each $(n, \vec{x}) \in \mathcal{N} \times \Omega$. Furthermore, if a record of the maximizing transition for each $(n, \vec{x})$ is maintained during the computation of (37) then it is is straightforward to recover the best path, e.g. by backtracing from (n_F, W, H). The practical significance of (37) is that decoding time is $\mathbf{O}(\|\mathcal{B}\| \times \|\Omega\|)$, i.e. bilinear in the number of branches and the number of image pixels.

A. Decoder Scheduling

Image decoding may be visualized in terms of path-finding in a 3-dimensional *decoding trellis*. Fig. 6 shows a portion of a trellis for the text column model of Fig. 4. Each node of the trellis represents the value of $\mathcal{L}(n; \vec{x})$ for some $(n, \vec{x}) \in \mathcal{N} \times \Omega$. As indicated in Fig. 6, the trellis nodes may be viewed as forming a stack of image planes, one for each node of the source model, where the dimensions of each plane are the dimensions of the input image. The trellis contains a directed branch from $(n_1, \vec{x}_1)$ to $(n_2, \vec{x}_2)$ if the source contains a branch t from n_1 to n_2 and $\vec{x}_2 = \vec{x}_1 + \vec{\Delta}_t$.

There is a one-to-one correspondence between paths in the trellis and paths in the source. In particular, MAP decoding involves finding the most likely trellis path from location $(0, 0)$ of the n_I plane to location (W, H) of the n_F plane. A Viterbi decoder finds this path by using (37) to recusively compute the most likely path from $(0, 0)$ of the n_I plane to *every* node of the trellis.

One issue that arises is specifying an order in which to visit the trellis nodes. Such a ordering specification is called a *schedule* for the recursion. The fundamental constraint on a valid schedule is that the data dependencies implied by (37) must be respected. That is, if there is a path in the trellis from $(n_1, \vec{x}_1)$ to $(n_2, \vec{x}_2)$ then $\mathcal{L}(n_1; \vec{x}_1)$ must be computed before $\mathcal{L}(n_2; \vec{x}_2)$.[4] It is easy to see that the schedule constraints can be satisfied if the trellis is acyclic. Moreover, it is also clear that the trellis will be acyclic if $\vec{\Delta}_\pi \neq \vec{0}$ for every cycle π in the source.

Although it is clear that any acyclic trellis can be scheduled, it is not straightforward to design a computationally efficient algorithm to test an arbitrary source for zero-displacement cycles and to find a valid schedule if no such cycles exist. The source of the difficulty is that the branch displacements are *vectors* whose components are unconstrained in sign. We have developed an approach to decoder scheduling based on the general theory of linear schedules for regular iterative algorithms [14], [22]. In this section we will briefly survey the decoder scheduling problem. An algorithm for computing schedule parameters is presented in the Appendix.

Formally, a decoding schedule for a Markov source is a set of non-negative, integer-valued functions $\{\tau(n; \vec{x}) \mid n \in \mathcal{N}\}$ that assigns to each node $n \in \mathcal{N}$ and image point $\vec{x} \in \Omega$ the "time" at which $\mathcal{L}(n; \vec{x})$ is computed. To be consistent with the data dependencies in (37) a valid schedule must satisfy the constraint that

$$\tau(R_t; \vec{x}) \;\geq\; \tau(L_t; \vec{x} - \vec{\Delta}_t) + 1 \qquad (38)$$

for each $t \in \mathcal{B}$ and $\vec{x} \in \Omega$.

To impose exploitable structure on the scheduling problem it is usual to constrain the scheduling functions to be affine transformations, so that

$$\tau(n; x, y) = \lambda_n^x x \;+\; \lambda_n^y y \;+\; \gamma_n \qquad (39)$$

where $\gamma_n \geq 0$ but λ_n^x and λ_n^y are unconstrained in sign. A schedule of the form (39) is called a *linear schedule*. A fundamental result of linear scheduling theory is that it is possible to find a linear schedule that satisfies (38) if and only if there is no cycle in the graph whose total (vector) displacement is zero. For this reason, a graph that contains no zero-displacement cycle is said to be *computable*.

One simple and very important type of linear schedule is the *uniform linear schedule*, in which λ_n does not depend on n. A basic scheduling result is that a graph can be uniformly scheduled if and only if there is no *set* of (possibly disjoint) cycles with zero total displacement. This is a stronger condition than the absence of a *single* zero-displacement cycle required for computability. Uniform linear schedules are commonly used in the implementation of multi-dimensional digital filters.

Document image source models are typically *not* uniformly schedulable. However, a related form of schedule can be defined that is appropriate when images are to be decoded in a row-wise manner, from top to bottom. A *row-major schedule* is a linear schedule in which $\tau(n; \vec{x})$ has one of two forms, depending on n; either

$$\tau(n; x, y) = \lambda W K(y - y_n^0) \;+\; \lambda W p_n \;+\; \lambda(x - x_n^0) \;+\; \gamma_n \qquad (40)$$

[4] It is usually assumed that the template match scores $\mathcal{L}(Z \mid Q_t[\vec{x}])$ can be computed in any order necessary and thus that they impose no constraints on scheduling.

or

$$\tau(n;x,y) =$$

$$\lambda WK(y-y_n^0) + \lambda Wp_n + \lambda[(W-1)-(x-x_n^0)] + \gamma_n \quad (41)$$

for

$$0 \leq x - x_n^0 \leq W-1 \quad (42)$$

where λ and K are positive constants, W is the width of the input image and p_n and γ_n are non-negative and satisfy the constraints

$$p_n \leq K-1 \quad (43)$$

$$\gamma_n \leq \lambda - 1. \quad (44)$$

The constants x_n^0 and y_n^0 define a spatial offset by which the domain over which $\mathcal{L}(n;\vec{x})$ is computed is displaced with respect to the nominal image domain.

Because of (42), (43) and (44), a row-major schedule defines a 4-level nested iteration for computing the values of $\mathcal{L}(n;\vec{x})$ in a row-wise fashion, from top to bottom. The levels of the iteration correspond, respectively, to the 4 terms on the right hand side of (40) or (41).

The outermost level iterates over y, computing $\mathcal{L}(n;\vec{x})$ for each n and $\vec{x}$ of a given row before proceeding to the next row. The second level (fixed y) iterates over a sequence of K "passes" indexed by p_n. Each pass corresponds to a subset of the nodes. The third level (i.e. fixed y and p_n) iterates over x. If p_n corresponds to a pass for which $\tau(n;x,y)$ has the form (40) this iteration proceeds in order of increasing x (left-to-right). Such a pass will be called a *forward* pass. On the other hand, if $\tau(n;x,y)$ has the form (41) the iteration proceeds right-to-left, and the pass will be called a *reverse* pass. Finally, the innermost level of the iteration (fixed y, p_n and x) evaluates $\mathcal{L}(n;\vec{x})$ for the nodes of the pass in order of increasing γ_n.

It is shown in [16] that a row-major schedule can be found for any computable image decoder that contains no cycle with negative y displacement. As described in the Appendix, the parameters of a row-major schedule can be computed by linear programming with an objective function that minimizes the storage requirements of the resultant decoder.

B. Schedule for Text Column Example

Fig. 7 shows a row-major schedule for the decoder of Fig. 4 constructed using the algorithm described in the Appendix. The assumed font baseline and depth are $B = 36$ and $D = 11$, respectively. Decoding proceeds from top to bottom, for $y = 0 \ldots H-1$. The processing for each y consists of two passes. The forward pass proceeds left to right, for $x = 0 \ldots W-1$; at each x, the likelihood $\mathcal{L}(n_I;x,y-46)$ is computed first and then $\mathcal{L}(n_1;x,y-46)$ and $\mathcal{L}(n_2;x,y-11)$ are computed in either order. The reverse pass proceeds right to left, for $x = W - 1 \ldots 0$; at each x the values of $\mathcal{L}(n_3;x,y)$ and $\mathcal{L}(n_F;x,y-46)$ are computed in turn.

It is not difficult to verify that the data dependencies in (37) are satisfied by the schedule in Fig. 7. For example, consideration of the branches into n_1 in Fig. 4 shows that the computation of $\mathcal{L}(n_1;x,y-46)$ requires $\mathcal{L}(n_I;x,y-46)$, $\mathcal{L}(n_1;x,y-47)$ and $\mathcal{L}(n_3;x,y-46)$. The node order within the forward pass insures that $\mathcal{L}(n_I;x,y-46)$ is computed before $\mathcal{L}(n_1;x,y-46)$. The value $\mathcal{L}(n_1;x,y-47)$ is available because it was computed during the forward pass for $y-1$. The value of $\mathcal{L}(n_3;x,y-46)$ was computed during the reverse pass for $y-46$ since $y_{n_3}^0 = 0$.

Similarly, the computation of $\mathcal{L}(n_2;x,y-11)$ requires $\mathcal{L}(n_1;x,y-47)$ and $\mathcal{L}(n_2;x-\Delta x,y-11)$ for various $\Delta x > 0$. The value of $\mathcal{L}(n_1;x,y-47)$ was computed during the forward pass for $y-1$ while the values of $\mathcal{L}(n_2;x-\Delta x,y-11)$ are available because the forward pass proceeds in order of increasing x.

Finally, the computation of $\mathcal{L}(n_3;x,y)$ during the reverse pass requires $\mathcal{L}(n_2;x,y-11)$ and $\mathcal{L}(n_3;x+1,y)$. The value of $\mathcal{L}(n_2;x,y-11)$ is available because the forward pass occurs before the reverse pass. The value of $\mathcal{L}(n_3;x+1,y)$ is available since the reverse pass proceeds in order of decreasing x.

VI. Decoding Telephone Yellow Pages

We have successfully used Markov source models to decode a variety of document types, including simple text, printed music, baseball box scores, and telephone yellow pages. As an example, this section briefly summarizes the typography of telephone yellow pages, defines a simple decoding task and model, and describes experimental results obtained using the model to decode a database of scanned pages.

A. Typography of Yellow Pages

Fig. 8 shows the reduced image of a sample page that illustrates most of the features found in the yellow pages published by one telephone company [19]. Horizontally, the page is divided into five $1.75in$ columns. Each column is $10in$ high, measured to the bottom of the black heading bar across the top of the page. Columns contain two main types of material— *listings* and *display ads*. The width of a display ad is always an integral number of columns and the height is a multiple of $2.5in$ (one fourth the column height). If a column contains both listings and ads, the ads are placed below the listings. The boundary between adjacent columns is marked by a thin vertical ruling, except where stradled by a multi-column ad.

In addition to listings and display ads, yellow pages contain a variety of publisher-supplied material, such as "reverse video" subject headings (e.g. top of first column in Fig. 8), page numbers, cross-references, and filler material below the listings in a column.

The internal composition of a display ad is relatively unconstrained. The formats of listings, on the other hand, are highly stylized and limited to roughly a half dozen basic types.

Listings fall into two main classes— individual listings and group listings. The most important elements of an individual listing are the name and telephone number(s). These are printed in specified fonts and occur at relatively fixed locations within the listing. Typically, the name occurs in the upper left corner and the number in the lower right corner, with the number immediately preceeded by one or more periods.

There are three main types of individual listings. The most common type, by far, is the *standard* individual listing, exemplified by "Intecom Inc " and "Mall Telecommunications" near the middle of the first column. In its usual form, a standard listing consists of a name line, one or more lines of address (or other information), and a telephone number. The font used for the name is slightly larger than the address font. A standard listing may be as short as a single line (e.g. "Mr Telephone Man" near middle of last column). On the other hand, the name alone may span several lines and a standard listing may contain multiple addresses or phone numbers (e.g. "The Good Guys" near the top of the third column).

A *bold* listing (e.g. "ROYAL TELECOMMUNICATIONS" at top of fifth column) is similar to a standard listing except that the name and phone number are in uppercase bold. A bold listing may have several phone numbers, in either bold or normal font; at least one number will be bold. One variation of the bold listing has a larger name, illustrated by "GOOD GUYS THE" near the middle of the first column.

A *boxed* listing (e.g. "TELSYSTEMS" near top of third column) has a bold name and a standard phone number and is enclosed by thick rulings along its top, left and bottom edges. The interior of a boxed listing contains relatively unconstrained text in a variety of fonts, possibly with a ruled frame around some of the text.

A *group* listing is a collection of individual listings typically used for franchises, businesses with multiple branches, brand names, etc. There are two types of group listings. A *standard group* listing (e.g. "PANASONIC AUTHORIZED SERVICENTERS" near bottom of fourth column) begins with a bold group name terminated by a short dash. The members of a group are divided into subgroups introduced by italic subgroup names (e.g. "PARTS & SERVICE"). The individual listings in a standard group are called *stdgrpmem* listings and are similar to standard individual listings except that the name is uppercase and the number may be bold (e.g. "SIEMANS BUSINESS TELEPHONES" near middle of second column). A standard group is terminated by a bottom horizontal rule that spans about 75% of the column width.

A second type of group listing is the *boxed group*, (e.g. "NORTHERN TELECOM INC" at middle of fourth column) which has thin rulings along both top and bottom, similar to those of an individual boxed listing. The group members are formatted as either standard or bold individual listings. Boxed group listings frequently contain logos or other graphics.

B. Decoding Task and Model

We define an illustrative yellow page decoding task in terms of three objectives. The primary objective is to recognize and parse the subset of individual listings consisting of those containing a single telephone number. Such listings are called *simple* individual listings. The simple individual listings to be decoded include those that occur at top-level of the column as well as those contained within groups. The desired output information for each listing is the name, telephone number and type (standard, bold, boxed, stdgrpmem).

The second objective is to recover the hierarchical structure of groups and subgroups whose member listings are all simple (called *simple* groups and subgroups). The desired output for each subgroup is the subgroup name and information about its members. The output for a simple group is the group name and a description of the contained subgroups.

The third objective is to recognize and transcribe the reverse-video subject headings. The subject headings are included because, as will be seen, they represent an interesting situation. Collectively, the individual listings, subgroups, groups and subject headings of a column are called the *constituents* of the column; the subset of these to be decoded are called the *simple constituents*.

Fig. 13 shows a decoded LaTeX-like message for the fourth column of Fig. 8 that illustrates the type of information to be extracted. The column contains 22 simple constituents, comprising 18 simple individual listings (10 standard, 5 bold, 1 boxed, 2 stdgrpmem) one simple standard group listing ("PANASONIC AUTHORIZED SERVICENTERS"), two simple subgroups and one subject heading. Of the 18 simple individual listings, 14 occur at top-level and 4 are contained within groups. The column also includes one display ad and one non-simple group ("NORTHERN TELECOM"), which contains one non-simple subgroup, which contains one non-simple individual listing.

We have developed a yellow page column model for extracting the above information that contains over 6000 branches and 1700 nodes. We will briefly describe the top-level model and some aspects of the models for ***boxed*** listings and ***subject*** headings.

Fig. 9 shows the top-level model for a yellow page column. The basic structure of a column is a sequence of top-level constituents separated by rows of vertical whitespace. Top-level constituents comprise ***standard***, ***bold***, and ***boxed*** individual listings, ***standard group*** and ***boxed group*** group listings, ***subject*** headings and ***junk*** lines. By convention, when a constituent subnetwork is entered the graphics cursor is at the upper left corner of the item; when the network exits the cursor is at the lower right. The ***CR*** network returns the cursor to the left edge of the column and essentially duplicates node n_3 and transition t_6 of the text column model in Fig. 4.

Fig. 10 shows the top-level vertical structure of a simple ***boxed*** listing. The first and last transitions of the model are labeled with the text fragments ("`\boxed{`" and "`}`") that bracket the message of a boxed listing in the overall column message. The ***name*** and ***number*** subnetworks generate the listing-specific information fields included in the output message and have similar bracketing transitions. The ***body*** and ***bottom rule*** subnetworks model regions of the image of a boxed listing but do not contribute to the message string.

Fig. 11(a) shows the overall model for the name field of a boxed listing. The ***first line*** model, shown in Fig. 11(b), accounts for both the first line of the name and the horizontal ruling that follows it along the top of the listing. Transition t_1 moves the cursor down from the top of the name to the text baseline. Each of the two ***char*** nets is simply a set of parallel branches with character templates drawn from the uppercase bold name font, analogous to the transitions around node n_2 of Fig. 4. The network shown in Fig. 11(b) constrains a name line to contain at least two characters. The top ruling begins a short distance S after the final name character. The template $\boldsymbol{Q}$ associated with t_4 represents a short (4 pixel wide) segment of top ruling. The character metrics of this template are defined so that $\boldsymbol{Q}$ is displaced the proper amount above the text baseline. Finally, t_5 moves the cursor down from the baseline by an amount equal to the font depth D. Thus, the overall line model moves the cursor from the upper left to the lower right corners of the text line, consistent with the convention noted previously.

A boxed name ***other line*** is similar to a ***first line*** except that it includes a fixed-width segment of side rule at the start of the line rather than a variable-width run of top rule at the end. The side rule segment extends from B rows above to D rows below the baseline. Finally, a ***side rule row*** functions as a row of vertical "whitespace" between consecutive name lines, but begins with one row of side rule.

The model for the ***body*** of a boxed listing is shown in Fig. 12. As Fig. 12(a) indicates, the body is a sequence of body lines vertically padded with side rule rows. The model for ***body line*** is shown in Fig. 12(b). The left part of the network, through transition t_1, models the vertical side ruling and is identical to the corresponding part of the ***other line*** component of the boxed ***name*** model.

The purpose of ***junk string*** is to account for all the material in the interior of a boxed listing to the right of the side rule. Since the decoder is not required to transcribe this material, the essential task of the junk model is to prevent the interior of a boxed listing from being decoded as part of the name and/or number or as separate individual entries. We have found that an adequate model for this purpose is a simple loop of character template transitions formed from the font for standard listing names. Of course, there are no message strings associated with these transitions. The ***junk string*** network is also used in the standard, bold, stdgrpmem and group listing models and a similar net is used in ***junk*** of the top-level column model.

The reverse-video subject headings represent an interesting situation because, compared with the other printing, the colors of foreground and background are reversed. It is possible to model images that are entirely reverse-video simply by exchanging the values of the channel parameters α_0 and α_1. Eventually, we expect to extend the imaging model to accommodate images containing both reverse-video and normal regions in a uniform way. For the yellow pages, however, the issue can be circumvented using a simple trick. The subject heading font contains no characters with negative left or right sidebearings.[5] Thus, in a line of subject text the character bounding boxes will not overlap nor will the slightly wider boxes defined by the character set widths. For each character we consider a box whose width is the set width of the character and which extends from D below to B above the baseline, where D and B are the font depth and height above baseline, respectively. The character templates of the ***subject*** model consist of these boxes, filled in black, with the white character etched out. Similarly, an all-black "space" template is included. A reverse-video subject heading is modeled as lines of text set in this (normal polarity but strange) font, separated by rows of appropriately-defined "blackspace".

C. Experimental Results

The dataset for the experiments consisted of ten pages selected from [19] and scanned at 300 dpi. The pages were informally chosen to contain relatively few display ads and a high proportion of simple individual listings.[6] One page included a pair of adjacent columns devoted entirely to double-width ads; each of the remaining 48 columns contained at least a few listings. The pages were automatically segmented into columns by decoding, using a simple model to trace the vertical inter-column rulings. The segmented columns were individually deskewed by vertical shearing; skew angle was estimated using the standard technique of maximizing the variance of the projected pixel density profile as a function of projection angle.

The font metrics and character templates required by the column model were obtained by using a text image editor [15] to extract samples of text in each required font from several of the columns in the database. The sample glyphs were clustered by binary correlation using a trellis-structured preclassifier [26] and the appropriate character label was manually assigned to each cluster. The font metric estimation procedure described in [17] was then used to estimate the sidebearings, set width and depth below baseline of each character. A character template was created from each cluster by aligning the sample glyphs and taking the median binary image value at each pixel. The reverse-video subject character templates were obtained by first complementing the sample subject text, performing the above steps, and then complementing the resulting templates.

The asymmetric bit flip channel parameter α_1 was estimated as the fraction of 1 bits among the pixels of the aligned sample glyphs, counted over the supports (1 bits) of the templates. This value, about .97, was also used as α_0. The complemented subject character templates and sample glyphs were included in this process.

A reference transcription was created for each column by manually correcting the decoded output generated by an early version of the decoder. The 48 columns contain 1244 constituents, of which 1208 (97.1%) are simple. The composition of the column database in terms of the distribution of simple constituents is summarized in Table I. The name fields comprise about 24000 characters; the numbers contain about 8000.

Each of the 48 columns in the dataset was independently decoded; decoding required, on average, 45 mins/column when run on a SUN SPARCstation 10. Fig. 13 shows the result of decoding the fourth column of Fig. 8. Comparison with Fig. 8 shows that all 22 simple constituents were correctly recognized.[7] The non-simple individual listing in the "NORTHERN TELECOM" boxed group was decoded as 2 simple listings, corresponding to the two telephone numbers. The ad at the bottom of the column was decoded as a random boxed listing. Eighteen of the 20 phone numbers were decoded exactly. There were a significant number of character errors in the constituent names.

Table I summarizes the performance of the decoder on the complete database of yellow page columns. Computation of the decoding statistics is based on a standard dynamic programming (DP) sequence alignment algorithm that minimizes the number of insertions, deletions and substitutions required to transform a correct sequence to a guess [24]. Constituent decoding statistics are computed by DP alignment of the sequences of decoded and true constituents for each column. During column alignment, two individual listings are taken to match if their types agree and DP alignments of the name and number strings both match 66% of the true characters.[8] Two subgroups match if their

[5] Where sidebearings, bounding box, etc are defined appropriately for white printing on black background.

[6] The actual pages are 41–44, 337, 338, 451, 452, 1105 and 1106.

[7] Many of the decoded names and numbers contain character errors, but the constituents per se were all detected and correctly segmented.

[8] The 66% threshold is arbitrary but not critical. In any case, it affects how decoder performance is measured but has nothing to do with the decoder itself.

types, names and member individual listings match. Similarly, two groups match if their types, names and subgroups match. Finally, two subject headings match if their name strings match.

Name and number string decoding statistics are computed over the constituents that match after column alignment. The string match rates are computed by requiring an exact match between the decoded and actual strings. Character match rates are computed by DP alignment.

Overall, 99.5% of the simple constituents were matched during decoding. The overall character recognition rates were 98.1% and 99.6% for the names and telephone numbers, respectively, with 98.5% of the number strings decoded exactly. The character insertion rate for standard listing names was relatively high, 6.5%. A main cause appears to be single-line listings that include address information (e.g. "Data-Tel" in the fourth column of Fig. 8). There were no character insertions for the telephone numbers.

VII. Conclusion

This paper has proposed a communication theory approach to document image recognition, called document image decoding. A class of images to be decoded is described in terms of a Markov source model. This model may be used either to synthesize images from message strings or to drive a MAP decoder that recovers messages from images. A linear scheduler automatically constructs a row-major decoder schedule from the image model. An important contributor to the generality of this approach is a normalization of the log-likelihood matching function for the asymmetric bit-flip channel that allows the bounding boxes of the templates to overlap.

VIII. Appendix

The objective of row-major scheduling is to obtain values for K, λ, y_n^0, p_n, x_n^0 and σ_n for use in the iterative decoding algorithm of Section V. The first part of this appendix develops a basic algorithm for obtaining schedule parameters by solving a sequence of linear programming (LP) problems. A set of LP objective functions is then derived for optimizing these parameters for minimum decoder storage.

The primary purpose of this appendix is to introduce scheduling issues and algorithms to the document image understanding community. The discussion will be relatively informal and will focus on providing a complete description of a particular scheduling algorithm that has been found useful. A number of theoretical issues will be ignored and not all aspects of the algorithm will be justified. A more formal theoretical analysis and alternative scheduling algorithms can be found in [16].

A. Computation of Schedule Parameters

The development of the basic scheduling algorithm will occur in three stages. First, we will describe the computation of y_n^0, p_n, x_n^0 and γ_n assuming that K, λ and σ_n are known. Then we will present simple modifications to the algorithm to obtain K, λ and a minimum permitted value for W. Finally, we will describe a procedure for determining σ_n. The exposition will be simplified if (40) and (41) are both written as

$$\tau(n;x,y) =$$

$$\lambda W K(y-y_n^0)+\lambda W p_n+\lambda\left[\sigma_n(x-x_n^0)\;+\;\epsilon_n(W-1)\right]+\gamma_n \tag{45}$$

where

$$\sigma_n \quad \equiv \quad \begin{cases} +1 & \text{if } n \text{ belongs to a forward pass} \\ -1 & \text{if } n \text{ belongs to a reverse pass} \end{cases} \tag{46}$$

is the *polarity* of node n and

$$\epsilon_n \quad \equiv \quad \begin{cases} 0 & \text{if } \sigma_n = 1 \text{ (forward pass)} \\ 1 & \text{if } \sigma_n = -1 \text{ (reverse pass).} \end{cases} \tag{47}$$

If K, λ and σ_n are known, the schedule parameters y_n^0, p_n, x_n^0 and γ_n can be computed by solving a sequence of four linear programs that may be viewed as scheduling at successively finer time scales. Each time scale defines a quantization of $\tau(n;\vec{x})$ and corresponds to one of the four levels of the iterative decoding algorithm. Equivalently, the four time scales correspond to approximating $\tau(n;\vec{x})$ by successive partial sums from the right side of (45) beginning with the leftmost term. Parameters computed at one level are used during the computations at succeeding levels.

The computation of y_n^0 is based on a quantization of $\tau(n;\vec{x})$ to image rows, defined by

$$\tau_y(n;\vec{x}) \equiv y - y_n^0. \tag{48}$$

At this granularity, schedule constraints (38) become

$$\tau_y(R_t;\vec{x}) \geq \tau_y(L_t;\vec{x}-\vec{\Delta}_t) \tag{49}$$

or, equivalently,

$$y_{L_t}^0 \;-\; y_{R_t}^0 \geq -\Delta y_t. \tag{50}$$

Since the left-hand side of (50) only involves differences between y_n^0 values, a solution to (50) remains valid if a constant is added to each y_n^0. Thus, it may be assumed that $y_n^0 \leq 0$ for each node. Given the known branch y displacements, values for the N unknowns y_n^0 can be obtained by solving the linear program defined by the B constraints (50).

A linear program for p_n follows from approximating $\tau(n;\vec{x})$ as the total number of elapsed passes, given by

$$\tau_p(n;\vec{x}) \equiv K\tau_y(n;\vec{x}) \;+\; p_n \tag{51}$$

$$= K(y-y_n^0) \;+\; p_n. \tag{52}$$

The schedule constraints on pass assigment are

$$\tau_p(R_t;\vec{x}) \geq \tau_p(L_t;\vec{x}-\vec{\Delta}_t) \;+\; \delta_t \tag{53}$$

where

$$\delta_t \quad \equiv \quad \begin{cases} 0 & \text{if } \sigma_{L_t} = \sigma_{R_t} \\ 1 & \text{otherwise.} \end{cases} \tag{54}$$

These constraints are easily put into the form

$$p_{R_t} - p_{L_t} \geq -K(y^0_{L_t} - y^0_{R_t} + \Delta y_t) + \delta_t \tag{55}$$

where p_n is also constrained to satisfy (43). The quantity

$$\Delta\tau^{(y)}_t \equiv y^0_{L_t} - y^0_{R_t} + \Delta y_t \tag{56}$$

is non-negative by (50) and is called the *branch y delay*. It may be interpreted as the delay, in number of rows, between the computation of $\mathcal{L}(L_t;\vec{x})$ and its use in computing $\mathcal{L}(R_t;\vec{x})$. The terms δ_t on the right sides of (53) and (55) insure that L_t and R_t are assigned to different passes if the branch y delay is zero and the polarities of the nodes differ.

The calculation of x^0_n follows from a quantization of $\tau(n;\vec{x})$ to individual pixel times, defined by

$$\tau_x(n;\vec{x}) \equiv W\tau_p(n;\vec{x}) + \sigma_n(x - x^0_n) + \epsilon_n(W-1) \tag{57}$$

$$= WK(y - y^0_n) + Wp_n + \sigma_n(x - x^0_n) + \epsilon_n(W-1) \tag{58}$$

with schedule constraints

$$\tau_x(R_t;\vec{x}) \geq \tau_x(L_t;\vec{x} - \vec{\Delta}_t). \tag{59}$$

If (59)is expanded, the result

$$\begin{aligned} \sigma_{L_t}x^0_{L_t} - \sigma_{R_t}x^0_{R_t} &\geq \\ WK(y^0_{R_t} - y^0_{L_t} - \Delta y_t) &+ W(p_{L_t} - p_{R_t}) \\ -\sigma_{L_t}\Delta x_t + (\epsilon_{L_t} - \epsilon_{R_t})(W-1) &+ (\sigma_{L_t} - \sigma_{R_t})x \end{aligned} \tag{60}$$

differs from (50) and (55) in that the variable x appears on the right side. However, (60) must be satisfied for all $0 \leq x \leq W-1$. By considering the four possible pairs of values for σ_{L_t} and σ_{R_t} and the definition of ϵ_n, it is not difficult to show that the last two terms of (60) can be replaced by $\delta_t(W-1)$ to give the equivalent constraints

$$\begin{aligned} \sigma_{L_t}x^0_{L_t} - \sigma_{R_t}x^0_{R_t} &\geq \\ WK(y^0_{R_t} - y^0_{L_t} - \Delta y_t) + W(p_{L_t} - p_{R_t}) &+ W\delta_t \\ -\sigma_{L_t}\Delta x_t &- \delta_t \end{aligned} \tag{61}$$

where $-\sigma_n x^0_n$ is non-negative and satisfies the constraint

$$-\sigma_n x^0_n \leq W - 1. \tag{62}$$

Finally, a linear program for γ_n is obtained by returning to the original time scale defined by (45). The original schedule constraints (38) can be written

$$\begin{aligned} \gamma_{R_t} - \gamma_{L_t} &\geq \\ \lambda WK(y^0_{R_t} - y^0_{L_t} - \Delta y_t) + \lambda W(p_{L_t} - p_{R_t}) &+ \lambda W\delta_t \\ -\lambda\sigma_{L_t}\Delta x_t - \lambda\delta_t + \lambda\sigma_{R_t}x^0_{R_t} &- \lambda\sigma_{L_t}x^0_{L_t} \\ &+ 1 \end{aligned} \tag{63}$$

where γ_n is also constrained by (44).

The above algorithm assumes that K, W and λ are known. However, it is not difficult to modify the linear programs to compute values for these parameters. For example, since the first term on the right side of (55) is linear in K, K can be treated as an unknown and determined along with the p_n. Similarly, W and λ can be computed by rearranging (61) and (63), respectively. Theoretical lower bounds for K, W and λ as well as alternative methods for obtaining legal values are described in [16].

Since W is the width of the input image, treating it as a variable to be determined requires explanation. It is not difficult to verify that a set of schedule parameters y^0_n, p_n, x^0_n and γ_n that satisfies (50), (55), (61) and (63) for some values of K, W, and λ continues to satisfy these constraints if K, W, or λ is increased. Thus, if K, W, and λ are determined by linear programming, the LP objective functions can be chosen so that the resulting values are the *minimum* values of these parameters for which a schedule can be found. In particular, the computed W value will be the smallest image width for which the iterative decoding algorithm is equivalent to a linear schedule.

One important issue that arises are the conditions under which the scheduling linear programs have solutions. It can be shown that row-major scheduling is possible for a computable Markov source if every cycle has non-negative y displacement [16]. A related issue is whether the solutions to the linear programs are integer-valued, as required. Because of the form of the constraint equations it can be shown that they are [20].

A computable Markov source that has no cycle with negative y displacement can be transformed into an equivalent graph in which no *branch* has negative y displacement [16]. In particular, the non-negative branch delays $\Delta\tau^{(y)}_t$ defined by (56) form such a set of transformed displacements. The procedure for determining the node polarities σ_n is based on the decomposition of the transformed source model summarized in Fig. 14. Consider the graph $\mathcal{G}'$ formed by deleting from $\mathcal{G}$ all branches for which $\Delta\tau^{(y)}_t > 0$ and then all nodes that have no remaining incoming or outgoing branches. Let $\mathcal{N}_+$ denote the set of deleted nodes. It is clear that any cycle with zero y displacement must lie entirely within $\mathcal{G}'$.

Decompose $\mathcal{G}'$ into its tightly connected components (TCCs) [22] $\mathcal{G}''_1, \ldots, G''_R$ and let $\mathcal{N}_0$ denote the nodes of $\mathcal{G}'$ that belong to no TCC. It is clear that any cycle in $\mathcal{G}$ with zero y displacement must lie entirely within one of the $\mathcal{G}''_i$. Since $\mathcal{G}$ is computable and every cycle in $\mathcal{G}''_i$ has zero y displacement, it follows that $\mathcal{G}''_i$ contains no cycle with zero x displacement. More strongly, $\mathcal{G}''_i$ has the property that every cycle within it has the same sign. If the sign is positive, assign $\sigma_n = 1$ to the nodes of $\mathcal{G}''_i$; otherwise assign $\sigma_n = -1$. Either value of σ_n may be assigned to the nodes of $\mathcal{N}_0$ and $\mathcal{N}_+$.

B. Decoder Storage Requirements

During image decoding, each value of $\mathcal{L}(n;\vec{x})$ must be retained from the time it is computed until all values are computed that depend on it by (37). The storage requirement for a decoder depends on the schedule and can be significant for a complex image model. This section develops a set of objective functions for the linear programs of the previous section that minimize storage requirements.

The amount of storage required to buffer $\mathcal{L}(n;\vec{x})$ for a given node while decoding an image of width W can be written

$$\rho_n(W + \beta_n) \tag{64}$$

where ρ_n is the number of buffer rows and β_n is the storage for boundary values (padding) beyond the ends of the nominal rows. The row storage ρ_n can be expressed in terms of the outgoing branch y delays as

$$\begin{aligned}\rho_n &= 1 + \max_{t \in \mathcal{O}_n} \Delta\tau_t^{(y)} \\ &= 1 + \max_{t \in \mathcal{O}_n} (y^0_{L_t} - y^0_{R_t} + \Delta y_t) \end{aligned} \tag{65}$$

where the 1 accounts for storage of the current row of $\mathcal{L}(n; \vec{x})$.

The overall row storage for a decoder is

$$\rho = \sum_{n \in \mathcal{N}} \rho_n. \tag{66}$$

The only schedule parameters that appear in (65) are the y^0_n, suggesting that (66) can be used as an objective in linear program (50). A minor difficulty arises because ρ_n is a non-linear function of y^0_n and thus ρ is not directly appropriate as an LP objective. However, if the ρ_n are treated as auxiliary variables, then the linear program for y^0_n can be expanded to compute ρ_n as well, with objective (66). The constraints for the auxiliary variables are

$$\rho_{L_t} \geq y^0_{L_t} - y^0_{R_t} + \Delta y_t + 1 \tag{67}$$

for each $t \in \mathcal{B}$.

The boundary condition storage β_n for a node is given by

$$\beta_n = \beta_n^{(r)} + \beta_n^{(l)} \tag{68}$$

where

$$\beta_n^{(r)} = \max\{\max_{t \in \mathcal{O}_n} (x^0_{R_t} - x^0_{L_t} - \Delta x_t), 0\} \tag{69}$$

and

$$\beta_n^{(l)} = \max\{\max_{t \in \mathcal{O}_n} (x^0_{L_t} - x^0_{R_t} + \Delta x_t), 0\} \tag{70}$$

represent the padding to the right and left of the row, respectively. The total boundary value storage for a schedule is

$$\beta = \sum_{n \in \mathcal{N}} \rho_n \beta_n. \tag{71}$$

Boundary padding is more complicated than row storage to analyze and minimize because each term

$$x^0_{L_t} - x^0_{R_t} + \Delta x_t \tag{72}$$

may be positive or negative and cannot, in general, be interpretted as a delay. Nevertheless, (71) can be used as an objective in the linear program for x^0_n if $\beta_n^{(r)}$ and $\beta_n^{(l)}$ are introduced as auxiliary variables and (61) and (62) are augmented with the additional constraints

$$\beta_n^{(r)} \geq x^0_{R_t} - x^0_{L_t} - \Delta x_t \tag{73}$$

and

$$\beta_n^{(l)} \geq x^0_{L_t} - x^0_{R_t} + \Delta x_t \tag{74}$$

for each t.

The relative costs of row and boundary value storage depend on the width of the image. For an 8.5×11 inch page at 300 dpi, $W = 2550$ and boundary storage is typically insignificant compared to row storage. However, minimizing boundary storage is desirable because doing so may lead to an additional opportunity for reducing row storage. If, for a given node, no value of $\mathcal{L}(n; \vec{x})$ is used outside the pass in which it is computed, the row storage for n can be eliminated. However, elimination of the row buffer incurs the cost of maintaining a length-β_n shift register of $\mathcal{L}(n; \vec{x})$ values. As β_n increases, the shift register overhead will eventually outweigh the potential advantage of reduced row storage. Thus, it is desirable to minimize β_n. The possibility of row storage elimination can be further increased by augmenting the linear program for p_n with auxiliary variables and constraints that minimize the "distances" (measured in passes) between connected nodes. Doing this will tend to maximize the number of times all dependents of a node are assigned to the same pass.

As noted previously, computation of the template match scores $\mathcal{L}(Z \mid Q_t[\vec{x}])$ may be taken as imposing no constraints on scheduling. However, these values require buffering and the question thus arises of whether their storage can be systematically minimized. One straightforward approach is to expand the source model by introducing one or more explicit input nodes, which are then scheduled along with the other nodes [16]. Each input node represents a set of templates whose match scores are a priori constrained to be computed simultaneously.

The schedule shown in Fig. 7 was obtained using all of the storage minimization techniques outlined in this section. It is straightforward to verify that 52 rows of node storage are required, two of which (n_I and n_F) can be replaced with shift registers. In addition, one row is required for each character template. The boundary pad storage depends on the font and equals 94 for the 12pt Adobe Times Roman used in the example.

References

[1] Adobe Systems Inc., *PostScript Language Reference Manual*, second edition, 1990, Reading: Addison-Wesley.

[2] H. Abelson and A. diSessa, *Turtle Geometry*, 1980, Cambridge: MIT Press.

[3] O. Agazzi, S. Kuo, E. Levin, R. Pieraccini, "Connected and degraded text recognition using planar hidden Markov models", *Proc. 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Minneapolis, MN, April 27–30, 1993.

[4] J. C. Anigbogu and A. Belaïïd, "Application of hidden Markov models to multifont text recognition", *Proc. International Conference on Document Analysis and Recognition*, pp. 785–793, Saint-Malo, France, September, 1991.

[5] H. Baird, "Document image defect models", *Proc. IAPR Workshop on Syntactic and Structural Pattern Recognition*, Murray Hill, NJ, June, 1990.

[6] L. Bahl, F. Jelinek, and R. Mercer, "A maximum likelihood approach to continuous speech recognition", *IEEE Trans. on Pattern Analysis and Machine Inelligence*, vol. PAMI-5, no. 2, March 1983.

[7] F. Chen, L. Wilcox and D. Bloomberg, "Word spotting in scanned images using hidden Markov models", *Proc. 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Minneapolis, MN, April 27–30, 1993.

[8] M. Chen, A. Kundu and J. Zhou, "Off-line handwritten word recognition using a hidden Markov model type stochastic network, to appear in *IEEE Trans. on PAMI.*

[9] P. Chou, "Recognition of equations using a two-dimensional stochastic context-free grammar", *SPIE Conf. on Visual Communications and Image Processing*, Philadelphia, PA, Nov. 1989.

[10] C. Goldfarb, *The SGML Handbook*, Oxford University Press, 1991.

[11] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 1979, Reading: Addison-Wesley.

[12] X. Huang, Y. Ariki, and M. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh: Edinburgh University Press.
[13] F. Jelinek, "Continuous speech recognition by statistical methods" *Proceedings of the IEEE*, vol. 64, no. 4, April, 1976.
[14] R. Karp, R. Miller and S. Winograd, "The organization of computations for uniform recurrence equations", *Journal of the ACM*, vol. 14, pp. 563–590, 1967.
[15] G. Kopec and S. Bagley, "Editing Images of Text", in *EP90*, R. Furuta (ed.), 1990, Cambridge: Cambridge University Press, also report P92-000150 (ISTL-92-3), XEROX Palo Alto Research Center, Palo Alto, CA, November, 1992.
[16] G. Kopec, "Row-Major Scheduling of Image Decoders", report P92-00061 (EDL-92-5), XEROX Palo Alto Research Center, Palo Alto, CA, June, 1992.
[17] G. Kopec, "Least-Squares font metric estimation from images", to appear in *IEEE Trans. on Image Processing*, Oct., 1993.
[18] L. Lamport, LaTeX*, A Document Preparation System*, 1986, Reading: Addison-Wesley.
[19] Pacific Bell, *Smart Yellow Pages, Palo Alto, Redwood City and Menlo Park*, 1992.
[20] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, 1982, Englewood Cliffs: Prentice-Hall.
[21] P. Prusinkiewicz and J. Hanan, *Lindenmayer Systems, Fractals, and Plants*, Lecture Notes in Biomathematics, no. 79, 1989, Berlin: Springer-Verlag.
[22] S. Rao, *Regular Iterative Algorithms and Their Implementations on Processor Arrays*, PhD thesis, Stanford University, Oct. 1985.
[23] R. Rubenstein, *Digital Typography*, 1988, Reading: Addison-Wesley.
[24] D. Sankoff and J. Kruskal (eds.), *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading: 1983.
[25] M. Tomita, "Parsing 2-dimensional language", *ACM International Workshop on Parsing Technologies*, 1989.
[26] K. Wong, G. Casey, and F. Wahl, "Document Analysis System", *IBM J. Res. Develop.*, vol. 26, no. 6, Nov. 1982.
[27] J. Vlontzos and S.Y. Kung, "Hidden Markov models for character recognition", *Proceedings of the 1989 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 1719–1722, Glasgow, Scotland, May 23–26, 1989.

Gary E. Kopec was born in Cleveland, OH on June 11, 1952. He received B.S., M.S. and Ph.D degrees from M.I.T. in 1973, 1976 and 1980, respectively. From 1980 through the present, he worked for M.I.T., Schlumberger, Atlantic Aerospace Electronics Corp, and Xerox, where he was involved variously, in VLSI design, speech processing and recognition, knowledge-based signal processing, and document image processing and recognition. His current research interests are signal processing, pattern recognition, and speech and image analysis. Currently he is with the Xerox Palo Alto Research Center in Palo Alto, CA. Dr. Kopec is a member of the ACM and the IEEE Computer and Signal Processing Societies. He was technical co-chairman for the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, held in San Francisco, CA.

Philip A. Chou was born in Stamford, CT on April 17, 1958. He received the B.S.E. degree from Princeton University in 1980 and the M.S. degree from the University of California, Berkeley, in 1983, both in electrical engineering and computer science, and the Ph.D. degree in electrical engineering from Stanford University in 1988. From 1977 through the present, he worked for IBM, AT&T Bell Laboratories, Princeton Plasma Physics Lab, Telesensory Systems, Speech Plus, Hughes, and Xerox, where he was involved variously in office automation, motion estimation in television, optical character recognition, LPC speech compression and synthesis, text-to-speech synthesis by rule, compression of digitized terrain, and speech and document recognition. His research interests are pattern recognition, data compression, and speech and image processing. Currently, he is with the Xerox Palo Alto Research Center in Palo Alto, CA. Dr. Chou is a member of Phi Beta Kappa, Tau Beta Pi, Sigma Xi, and IEEE Computer, Information Theory, and Signal Processing societies.