Journal of Electronic Imaging 5(1), 7–14 (January 1996).

Markov source model for printed music decoding

Gary E. Kopec and Philip A. Chou
Xerox Palo Alto Research Center
Information Sciences and Technologies Laboratory
3333 Coyote Hill Road
Palo Alto, California 94304
E-mail: kopec@parc.xerox.com

David A. Maltz*
Carnegie Mellon University
School of Computer Science
Pittsburgh, Pennsylvania 15213

Abstract. *A Markov source model is described for a simple subset of printed music notation that was developed as an extended example of the document image decoding (DID) approach to document image analysis. The model is based on the Adobe Sonata music symbol set and a finite-state language of textual music messages. The music message language is defined and several important aspects of message imaging are discussed. Aspects of music notation that appear problematic for a finite-state representation are identified. Finally, an example of music image decoding and resynthesis using the model is presented. Development of the model was greatly facilitated by the duality between image synthesis and image decoding that is fundamental to the DID paradigm.* © *1996 SPIE and IS&T.*

1 Introduction

Document image decoding (DID) is an approach to document image recognition that is based on an explicit communication theory view of the processes of document creation, transmission, and recognition.[1–3] DID views a document recognition problem as consisting of four elements—a message source, an imager, a noisy channel, and a decoder. The most important activity in applying DID to a particular problem involves developing stochastic models for the source and imager. Much of the work in DID has focused on Markov (stochastic finite-state) source models with a path-based approach to imaging that is motivated by the sidebearing model for letterform shape description and positioning.[3] Models of this type have been successfully used for decoding formatted text such as simple text columns, telephone yellow pages, dictionary entries, and baseball box scores.

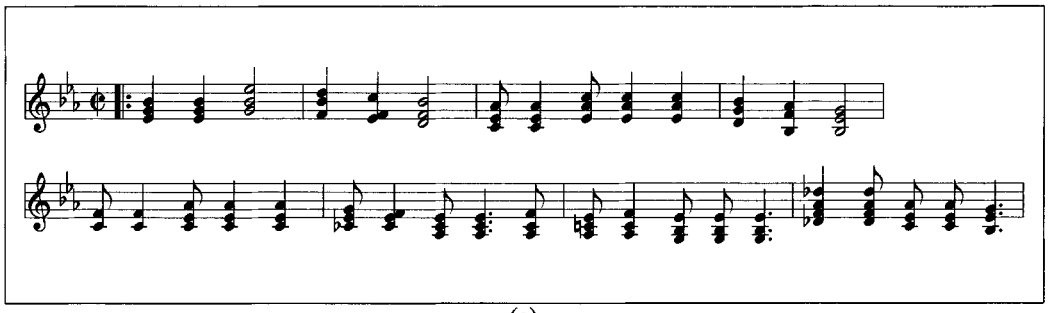
This paper describes an initial attempt to a develop a Markov source model for printed music notation. Music decoding is a much more complex task than decoding printed text because of the richness and variety of music notation.[4] In particular, printed music exploits the two-dimensionality of an image to a much greater extent than does printed text. For example, in printed text, the character denoted by a glyph is usually completely determined by the glyph shape, independent of its position relative to adjacent glyphs. The musical event denoted by a printed note, on the other hand, is determined by both note shape (which specifies duration) and note position (which specifies pitch). Thus, music represents a good vehicle for investigating the two-dimensional generality of finite-state image source models.

We have developed an image source model for a restricted subset of printed music notation, using a message language of our own design and a subset of the Adobe Sonata music font.[5] Fig. 1(a) shows a music image that illustrates some of the elements of notation handled by the model. The corresponding message string is shown in Fig. 1(b). The music model contains about 700 nodes and 1200 branches.

Unlike most other investigations in music recognition (see Ref. 6 for a survey of music image analysis), our work was not motivated by an interest in music recognition per se, but rather by the desire to develop a large and relatively complex image source model as an illustrative and pedagogical application of DID. In addition, we used music as a vehicle for exploring certain general issues in Markov image source theory and implementation. For example, music served as the driving example in our research on linear scheduling, a central issue in decoder implementation.[3] Because our primary objective was not to decode real printed music, we freely ignored many important notational con-

*Work performed while author was a summer intern at Xerox Palo Alto Research Center.

Paper DDI-08 received July 11, 1995; revised manuscript received Nov. 15, 1995; accepted for publication Nov. 16, 1995.
1017-9909/96/$6.00 © 1996 SPIE and IS&T.



(a)

```
G-3Tc{=1
Qe1g1b1SS=2Qe1g1b1SS=2Hg1b1e2SS=3|=1
Qf1b1d2SS=2Qe1f1c2SSs=2Hd1f1b1SS=3|=1
Qc1e1a1SsF8=1Qc1e1a1SSs=2Qe1a1c2SsF8=1Qe1a1c2SSs=2Qe1a1c2SSs=2|=1
Qd1g1b1SS=2Qbf1a1SSs=2Hbe1g1SS=1$
<11010010000
G-3Qc1f1SSF8=1Qc1f1SSS=2Qc1e1a1SsF8=1Qc1e1a1SSs=2Qc1e1a1SSs=2|=1
-c1Qc1e1g1SsF8=1Qc1e1f1SSS=2Qac1e1SsF8=1QaDc1De1DSSs=2Qac1f1SSF8=1|=1
*c1Qac1e1SsF8=1Qac1f1SSS=2Qgbe1SsF8=1Qgbe1SsF8=1QgDbDe1DSSs=1|=1
-d1d2Qd1f1a1d2SS=2Qd1f1a1d2SF8=1Qc1e1a1SsF8=1Qc1e1a1SsF8=1QbDe1Dg1DSSs$
```

(b)

Fig. 1 Example of printed music, illustrating range of notation handled by model. (a) Music image. (b) Message corresponding to (a).

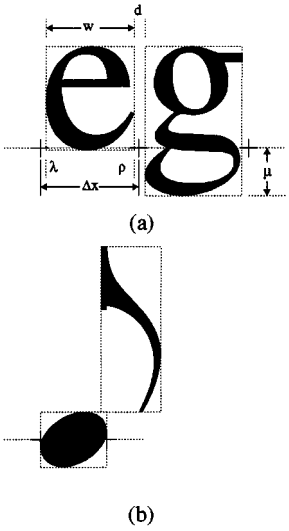
structs that did not seem to fit easily within the finite-state framework. Examples of such notational elements include beams and slurs. Similarly, we skipped many features that we believed would increase the size of the model without introducing new research issues. Thus, for example, all stems are directed upward in our notation. Despite these limitations, the resulting model does handle an interesting subset of music notation, as Fig. 1 illustrates.

The rest of this paper is organized as follows. Section 2 briefly reviews the concept of a Markov image source model introduced in Ref. 3. Section 3 introduces the music model by presenting a finite-state grammar for its message string language. Section 4 describes the finite-state model itself and discusses imaging issues associated with various aspects of the message language. Finally, Sec. 5 shows an example of music image decoding.

2 Markov Image Sources

The use of Markov sources for document image modeling is motivated by the sidebearing model[7] of character shape description and letterspacing, summarized in Fig. 2(a). This model is used widely in digital typography and is formalized in page description languages such as PostScript.[8] 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.

The bounding box of a character is the smallest rectangle, oriented with the character coordinate axes, that just encloses the character. The left sidebearing is the horizontal displacement λ from the origin of the character to the left edge of the bounding box. 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.



(a)

(b)

Fig. 2 Typographic model for character placement. Character origins are indicated by crosses. (a) Character spacing and alignment parameters ("font metrics"). (b) Sequential imaging of music characters: flag followed by notehead. The flag set width is zero.

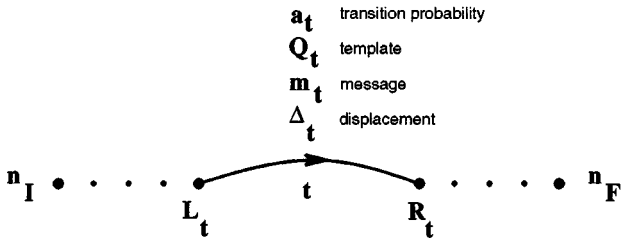


Fig. 3 Simple Markov source model for image generation.

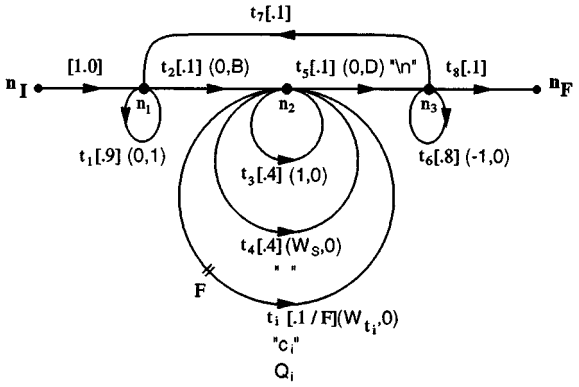


Fig. 4 Simple text column network. Transition probabilities are enclosed in brackets. The template and message associated with a branch are null unless indicated.

The general form of a Markov image source, illustrated in Fig. 3, is a directed graph consisting of a finite set of states (nodes, vertices) $\mathcal{N}=\{n_1 \ldots n_N\}$ and a set of directed transitions (branches, edges) $\mathcal{B}=\{t_1 \ldots t_B\}$. 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 states are the initial state n_I and the final state n_F. With each transition 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 $\vec{\Delta}_t=(\Delta x_t, \Delta y_t)$ is the two-dimensional integer vector displacement.

A complete path π in a Markov source is a sequence of transitions $t_1 \ldots t_P$ for which $L_{t_1}=n_I$, $R_{t_P}=n_F$, and $R_{t_i}=L_{t_{i+1}}$ for $i=1,\ldots,P-1$. Associated with each complete path π is a composite message

$$M_\pi = m_{t_1} \cdots m_{t_P} \tag{1}$$

formed by concatenating the message strings of the transitions of the path. Also associated with each path π is a sequence of position $\vec{\xi}_0 \ldots \vec{\xi}_P$ recursively defined by $\vec{\xi}_i = \vec{\xi}_{i-1} + \vec{\Delta}_{t_i}$, where $\vec{\xi}_0$ is an initial position, normally $\vec{0}$. (This paper uses an image coordinate system in which x increases to the right, y increases downward, and the upper left corner is at $x=y=0$.) A path defines a composite image Q by

$$Q_\pi = \bigcup_{i=1}^{P} Q_{t_i}[\vec{\xi}_{i-1}], \tag{2}$$

where $Q[\vec{x}]$ denotes Q shifted so that the origin of its local coordinate system is located at $\vec{x}$.

An image source defines a relation between message strings and images via an underlying path and Eqs. (1) and (2). 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 discussed in Ref. 3. Briefly, 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 Eq. (2) is then an image of M.

If the source model defines a deterministic[9] acceptor for the message language, the process of message imaging admits a simple procedural interpretation, similar to the "turtle geometry"[10] interpretation for strings generated by Lindenmayer systems in computer graphics.[11] 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 nonnull, 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.

Figure 4 shows a simple Markov source model for a text column that illustrates the above concepts. A text column consists of a vertical sequence of text lines, alternating with white space. Horizontally, a text line is a sequence of characters typeset according to the sidebearing model shown in Fig. 2(a). State n_1 corresponds to the creation of vertical white space. 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 white space. Branch t_3 represents a white space of minimal (one 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 la-

beled 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.

Maximum a posteriori (MAP) decoding using a Markov source involves finding a complete path $\hat{\pi}$ (and thus $\hat{M}$) that maximizes a path likelihood function defined by

$$\mathscr{L}(\pi)=\sum_{i=1}^{P}\,[\mathscr{L}(Z|Q_{t_i}[\vec{\xi}_{i-1}])+\log\ a_{t_i}],$$

where $\mathscr{L}(Z|Q_{t_i}[\vec{\xi}_{i-1}])$ is the template match score for Q_{t_i} aligned at position $\vec{\xi}_{i-1}$ and depends on the channel model.[3] For a simple asymmetric bit-flip channel, $\mathscr{L}(Z|Q_{t_i}[\vec{\xi}_{i-1}])$ can be computed by binary correlation. MAP decoding can be implemented by computing the recursively defined function

$$\mathscr{L}(n;\vec{x})=\max_{t|R_t=n}\{\mathscr{L}(L_t;\vec{x}-\vec{\Delta}_t)+\log\ a_t + \mathscr{L}(Z|Q_t[\vec{x}-\vec{\Delta}_t])\} \tag{3}$$

for each $n\in\mathscr{N}$ and image position $\vec{x}$ using a two-dimensional segmental Viterbi (dynamic programming) algorithm. If a record of the maximizing transition for each $(n,\vec{x})$ is maintained during the computation of Eq. (3) then it is straightforward to recover the best path, e.g., by backtracing from (n_F,W,H). The practical significance of Eq. (3) is that decoding time is $\mathbf{O}(\|\mathscr{B}\|\times H\times W)$, i.e., bilinear in the number of branches and the number of image pixels.

Development of the music model was greatly facilitated by the duality between image synthesis and image decoding provided by Markov sources. Although our ultimate objective concerned decoding, most of the development proceeded by using the evolving model for image synthesis, since it is computationally far less costly to image a message than to decode an image. The one aspect of decoder design that we found not adequately addressed by the synthesis paradigm was enforcing the template disjointness constraint, discussed in Ref. 3, that prohibits templates associated with different branches of a path from overlapping in support. This constraint should be satisfied by all paths through the network, a condition difficult to establish simply by observing the results of imaging sample strings. As a result, some changes to the model were required after its development as an imager was completed and we began using it for decoding.

3 Music Message Grammar

An important first step in developing an image source model is to specify the language of message strings, since the message language defines the information that is carried by and extracted from the image. A minor complication in the case of printed music is the absence of an established convention for the representation of music notation as text strings. Thus, we invented a simple music language as part of this investigation. Figure 5 shows a set of regular expressions defining our language of music messages. (The actual music image model takes the form of a finite-state network. However, it is more convenient to discuss the message language in terms of its grammar.) A music *message* describes a page of printed music. A page consists of a sequence of one or more music lines, separated by white space. In a message, each *music_line* is terminated by a '$'. The relative horizontal alignment of consecutive lines is specified using a *backspace*, a string of '0's and '1's (preceeded with '<') that specifies in binary the horizontal displacement (in pixels) from the end of a line to the beginning of the following line. A music line consists of a clef (either 'G' or 'F'), optional key and time signatures, and a sequence of chords, bar markers, and intervals of blank staff. The key signature is specified by the number of sharps (preceeded with '+') or flats (preceeded with '−'). The time signature is either common time ('TC') or cut time ('Tc'). An interval of blank staff is specified by '=' followed by an integer that gives the length in units of the set width of the quarter-note head. A bar mark is a simple bar line ('|') or a left or right repeat sign, indicated in a message by '{' and '}', respectively.

As an example, the first line of Fig. 1(b) specifies a music line that begins with a 'G' clef followed by a key signature of three flats ('−3'), cut time ('Tc') and the start of a repeat interval ('{'). The correspondence to the beginning of the first line of the image in Fig. 1(a) is direct.

The most important constituents of a music message are the chords. A *chord* begins with an optical sequence of *accidentals*, each of which specifies a set of pitch positions for sharp ('+'), flat ('−'), or natural ('*') signs, and a specification of whether the chord contains quarter notes ('Q') or half notes ('H'). The body of a chord string is a sequence of pitch positions for the noteheads. A notehead is dotted if its pitch specifier is followed by a 'D'. The imager automatically generates that portion of the chord stem that extends from the lowest to the highest notes of the chord. The stem may be extended beyond the highest note by a multiple of the vertical staff line spacing ('S'), or by half that amount ('s'). A *quarternote_chord* may include an eighth or sixteenth note flag, specified by 'F8' and 'F16' respectively.

message → (*music_line*[*backspace*])**music_line*
backspace → **<1(0 | 1)***
music_line → *clef*[*key_signature*][*time_signature*](*chord* | *blank_staff* | *bar_mark*)$^+$**$**
clef → **G | F**
key_signature → **(+ | -)(1 | 2 | ... | 7)**
time_signature → **T(C | c)**
blank_staff → **=(1 | 2 | 3 | 4)**
bar_mark → **| | { | }**
chord → *accidentals**(*quarternote_chord* | *halfnote_chord*)
quarternote_chord → **Q***dotted_pitch*$^+$**S***[**s**][**F8** | **F16**]
halfnote_chord → **H***dotted_pitch*$^+$**S***[**s**]
accidentals → **((+ | - | *)***pitch*$^+$**)**
dotted_pitch → *pitch*[**D**]
pitch → **f | g | a | b | ... | g2**

Fig. 5 Music message grammar, as regular expressions. Terminal symbols are in **bold** and nonterminals are in *italics*. The symbols ( ), [ ], |, →, *, and $^+$ are metalinguistic.

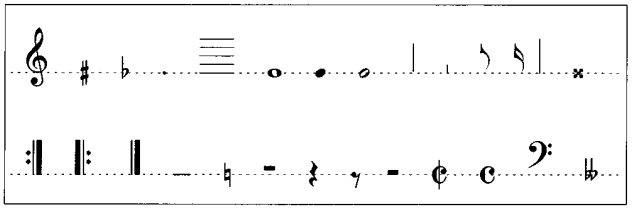


Fig. 6 The 27-character subset of Adobe Sonata used in the music model. Glyph origins are aligned on the dashed baselines.

For example, the initial portion of the fourth line of Fig. 1(b) describes the third measure of Fig. 1(a). The first chord of the measure is a quarternote chord with notes at c1, e1, and a1. The chord stem is extended by one and a half staff line spaces and is topped with an eighth note flag.

The pitch sequences for accidentals and chords are defined in Fig. 5 as *pitch* $^{+}$ and *dotted_pitch**, respectively. This is a simplification of the actual grammar, which requires the pitches to be given in increasing order. Our pitch specifiers differ from normal register symbols in that they are based on spatial position on the staff, rather than on actual pitch. Thus, for example, the bottom and top staff lines are denoted 'e1' and 'f2', respectively, independent of the clef. Fig. 7 defines the subset of register symbols that fall on the staff lines or in the interline spaces. As the *pitch* production of Fig. 5 suggests, notes above and below the staff are also represented.

It is clear from the grammar that the music language is suitable for representing single-voice scores only, although intervals and chords are supported. This limitation is shared with many other score recognition systems and is not a problem for some applications. In our case, the motivation for the single-voice limitation is that it is difficult, using a finite-state string grammar, to represent two or more distinct, but mutually constrained, event trains. A related consideration leads to the far more serious limitation that our language excludes beams and slurs. The reason for this is that the length and slope of a beam (or beam segment) are determined by the number and composition of all the chords under the beam. Such dependence is difficult to capture directly in a finite-state system. It might be possible to include beams using an incremental generation technique similar to that described below for the staff, but we have not investigated this possibility.

4 Music Imaging Model

The sidebearing model of character shape and letterspacing, while primarily motivated by conventional letterform typography, is useful for a much wider class of images, such as printed music. The music model is based on the 27-character subset of the Adobe Sonata font shown in Fig. 6. Sonata is typical of fonts designed for creating nontextual images in a PostScript environment. As in ordinary text fonts, each Sonata character has a set of metric parameters, including set width, sidebearings, and height above baseline. The symbols in each line of Fig. 6 are drawn with their origins on the dashed baseline. The music symbol font metrics were estimated using the least-squares procedure described in Ref. 12 from a font sample image consisting of a large number of glyphs aligned as in Fig. 6.

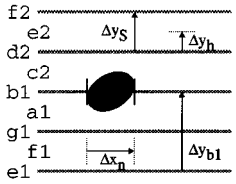


Fig. 7 Register symbols and staff spatial parameters.

Unlike text characters, most glyphs in printed music are not simply placed one after another along a horizontal 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.[5] Figure 2(b) shows, as an example, the result of sequentially imaging an eighth flag and a quarternote notehead. The set width of the flag is zero and the flag left sidebearing equals the width of the notehead minus the width of a stem (not shown). Thus, when the flag is drawn it appears in the correct location to the right and the cursor remains in position for drawing the notehead. After the notehead is drawn, the cursor is at the right edge of the notehead, at the right cross in the figure. The left sidebearing and set width of a stem are the same as those of a flag. Thus, sequentially imaging a stem, flag, and notehead produces the expected result.

Figure 7 defines a number of spatial parameters used in describing the placement of music glyphs on the staff. The vertical displacement from one staff line to that immediately above is denoted Δy_s. Note that $\Delta y_s<0$ in the coordinate system used in this paper. One-half the interline displacement is Δy_h. This is the vertical displacement between the origins of notes corresponding to successively higher pitches. The set width of a notehead is denoted Δx_n. Finally, the vertical displacement from the bottom staff line to the origin of a note at **x** is denoted $\Delta y_{\mathbf{x}}$. For example, $\Delta y_{\mathbf{b1}}$ is the displacement to the middle staff line, as indicated in Fig. 7.

The music page model is structured as a recursive source,[13] consisting of a top-level page model and a collection of named subsources. The top-level music model is shown in Fig. 8 and is similar to the top-level source of the recursive yellow page column model described in Ref. 13. We will briefly describe a few of the music subsources to illustrate some of the issues and modeling techniques involved. Figure 9 shows subsource *music_line*, the model for a single line of music. A *music_line* begins with a bar line (template Q_{bar}) and a segment of blank staff (template Q_{staff}) that is superimposed over the bar line, by virtue of

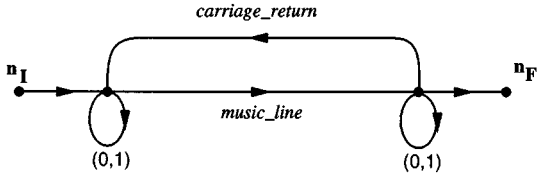


Fig. 8 Top-level subsource of recursive music page model.

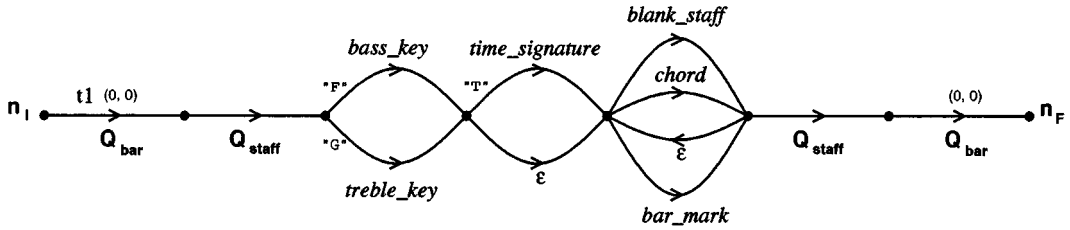


Fig. 9 Subsource *music_line.*

the (0,0) displacement of branch *t1*. The technique of incrementally generating the staff by overlaying staff segments on top of other constituents is used throughout the music network. This violates the template disjointness constraint reviewed previously but does not introduce decoding problems because the amount of overlap is small.

The structure of the line network mirrors the structure of the right-hand side of the *music_line* production in Fig. 5 except for the way in which the clef and key signature are imaged. In the message grammar, *clef* and *key_signature* appear as independent sequential constituents. In the image, however, the appearance of a given key signature on the staff depends on the clef. The *bass_key* and *treble_key* subsources each generate one of the two alternative clefs and the key signatures for that clef.

Chord imaging is the most complex part of the music network. As suggested by the *chord* message grammar production, the *chord* network invokes a subsource for *accidentals* followed by either a *quarternote_chord* or a *halfnote_chord*. These latter two subsources are very similar, except that *halfnote_chord* does not include stems or flags. Figure 10 illustrates the overall structure of the *quarternote_chord* network. The elements of a quarternote chord (noteheads, dots, stems, and flags) are imaged in roughly vertical order, from the bottom of the chord to the top. The network is organized as a set of interconnected modules, each of which is responsible for chord elements that are placed on a given staff line or the space immediately above it. For example, the *g1_a1_dptiches* subnetwork handles the noteheads and dots at g1 and a1 while the associated stem fragments and flags are imaged by *g1_a1_sflags*.

Much of the complexity of chord imaging follows from a rule of music typography that requires the noteheads for adjacent pitches to be placed on opposite sides of the chord stem, as illustrated in the second measured of the top line of Fig. 1(a). This rule leads to a proliferation of cases for other typographic details such as dot placement. If there is a nor-

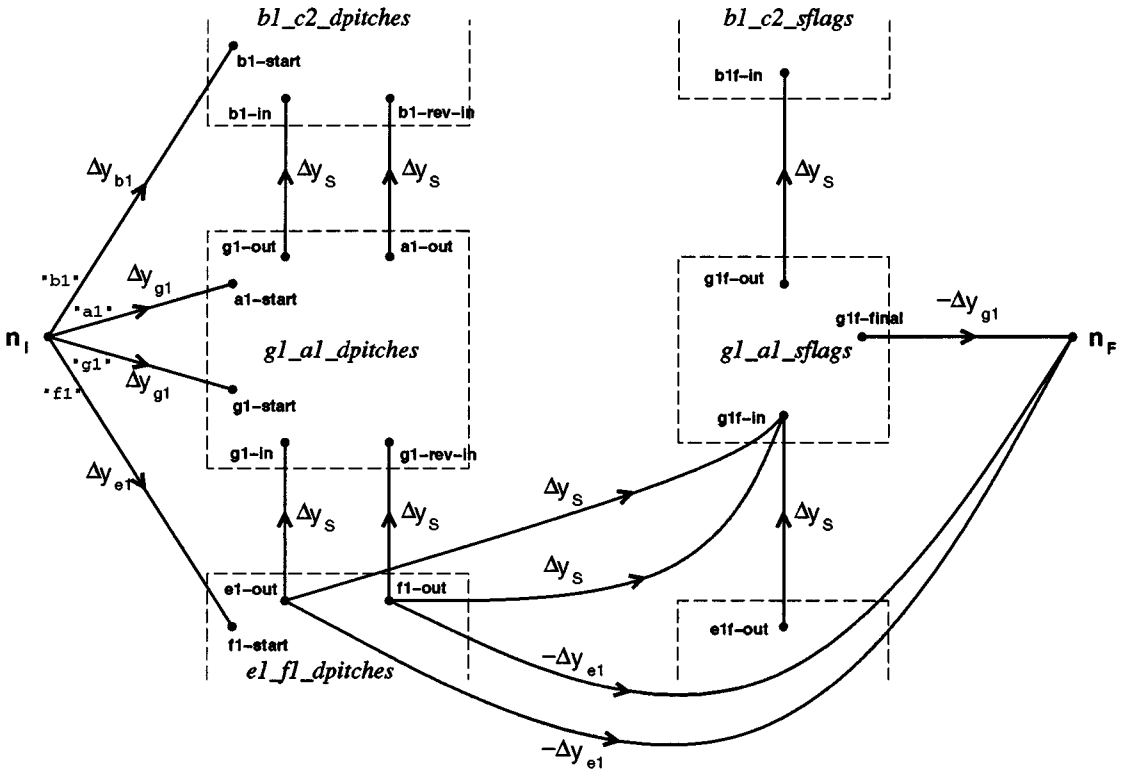


Fig. 10 Fragment of subsource *quarternote_chord.*

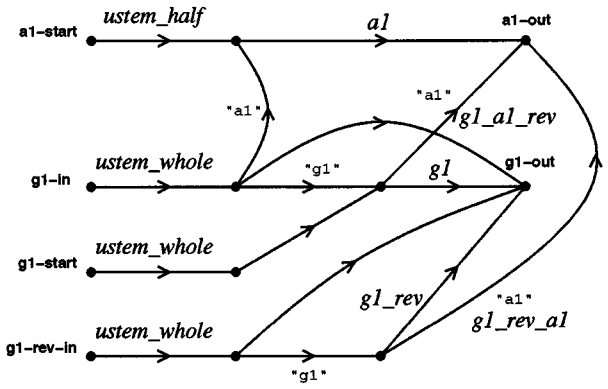


Fig. 11 Subsource *g1_a1_dpitches*.

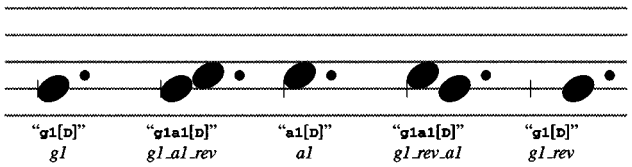


Fig. 12 Images and message fragments corresponding to subsources of *g1_a1_dpitches*. As indicated by the messages, the dots are optional.

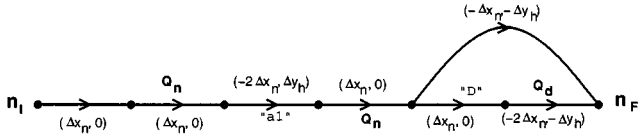


Fig. 13 Subsource *g1_rev_a1*.

mal (i.e., not reversed) notehead at f1, the path into *g1_a1_dpitches* from *e1_f1_dpitches* enters at node g1-rev-in and any note at g1 will be reversed. Otherwise, entry from *e1_f1_ dptiches* occurs at node g1-in. Similarly, the path from *g1_a1_dpitches* will exit at a1-out if a note is placed at a1, and at g1-out otherwise.

By programming convention, the displacement through each of the submodules of *quarternote_chord* is $\vec{0}$, as is the overall displacement through *quarternote_chord*. Thus, for example, it is clear from Fig. 10 that the *y* position entering and leaving *g1_a1_dpitches* is Δy_{g1} from what it is at n_I.

Figure 11 shows the structure of *g1_a1_dpitches* and Fig. 12 illustrates the notehead and dot configurations produced by the subsources of *g1_a1_dpitches*. The subsource, *ustem_whole* (''upstem whole'') produces a stem segment that extends from one staff line to the next; *ustem_half* generates the upper half of a stem segment when a1 is the bottom note of the chord. As before, the displacement through each of the subsources of *g1_a1_dpitches* is $\vec{0}$.

Figure 13 shows the internals of subsource *g1_rev_a1*. The reversed notehead at g1 (template Q_n) is drawn first, followed by the notehead at a1 and, finally, the optional dot (template Q_d).

The page model was initially designed as a recursive source structured as a top-level source plus 11 subsources. The separation procedure described Ref. 13 was used to create a separable model for use in decoding. The flat constrained source produced in the first step of the separation procedure contained 732 nodes and 1187 branches. After separation and simplification, the model consisted of a top-level subsource with 9 nodes and 11 branches plus two subsources corresponding roughly to ''carriage return'' and *music_line*. The simplified music line subsource contained 370 nodes and 821 branches.

5 Decoding Example

The music image in Fig. 1(a) was synthesized by using the image model to image the message in Fig. 1(b). All line baselines are perfectly horizontal in the synthesized image and the bitmaps of all instances of a given character are identical. The ''ideal'' image was degraded into an ''observed'' image by applying the bit-flip noise model described in Ref. 3. The noise parameters were $\alpha_1 = 0.9$ and $\alpha_0 = 0.51$, corresponding to inversion of 10% of the black and 49% of the white pixels of the ideal images. Fig. 14 shows the noisy image generated from Fig. 1(a). (The visual appearance of the noisy image depends on the printing process and might not accurately reflect the actual noise level.)

Figure 15 shows the result of using the music network to decode the image of Fig. 14. The decoded message is given in Fig. 15(a) and a reconstruction of the original image from the decoded message is shown in Fig. 15(b). Compar-

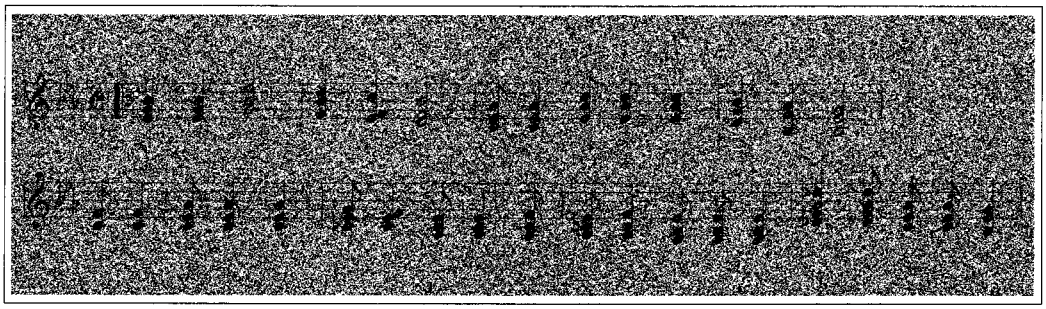
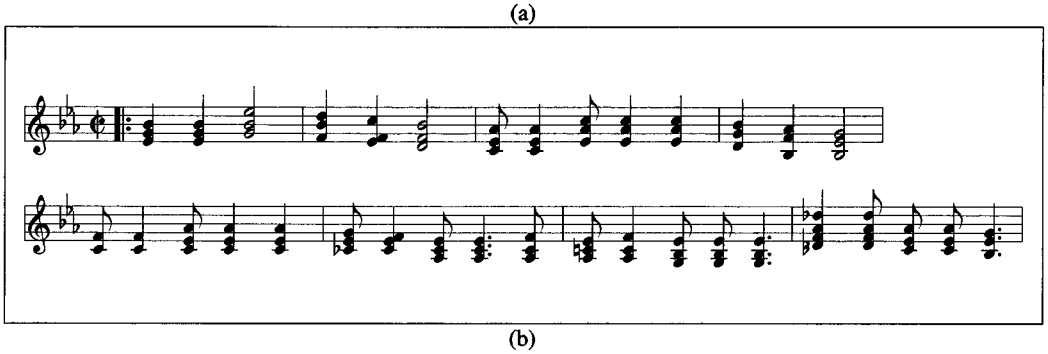


Fig. 14 Degraded music image formed by randomly inverting 10% of the black and 49% of the white pixels of Fig. 1(a).

```
G-3Tc{=1
Qe1g1b1Ss=2Qe1g1b1SSs=2Hg1b1e2SSs=3|=1
Qf1b1d2SS=2Qe1f1c2SSs=2Hd1f1b1SS=3|=1
Qc1e1a1SsF8=1Qc1e1a1SSs=2Qe1a1c2SsF8=1Qe1a1c2SSs=2Qe1a1c2SSs=2|=1
Qd1g1b1SS=2Qbf1a1SSs=2Hbe1g1SS=1$
G-3Qc1f1SSF8=1Qc1f1SSS=2Qc1e1a1SsF8=1Qc1e1a1SSs=2Qc1e1a1SSs=2|=1
-c1Qc1e1g1SsF8=1Qc1e1f1SSS=2Qac1e1SsF8=1QaDc1De1DSS=2Qac1f1SSF8=1|=1
*c1Qac1e1SsF8=1Qac1f1SSS=2Qgbe1SsF8=1Qgbe1SsF8=1QgDbDe1DSSs=1|=1
-c1d1d2Qd1f1a1d2SS=2Qd1f1a1d2SF8=1Qc1e1a1SsF8=1Qc1e1a1SsF8=1QbDe1Dg1DSSs$
```

(a)



(b)

Fig. 15 Example decoding of music image. (a) Music message decoded from Fig. 14. (b) Reconstructed music image synthesized from decoded message.

ing Figs. 15(b) and 1(a) [or Figs. 15(a) and 1(b)] reveals that the most significant decoding error is the insertion of an accidental flat at c1 before the first chord of the last measure. Note that the chord does not include a note at c1. The insertion error reflects the fact that accidentals and chords are specified and imaged independently in the current model. This error might be eliminated by imposing additional grammatical constraints. The reconstructed image also differs from the original in that some of the stem lengths are slightly different.

6 Summary

The design of a Markov source model for a limited but interesting subset of printed music notation was discussed. The model illustrates the application of the DID methodology to a document image domain that is significantly more complex than simple formatted text. The model accurately captures a number of important typographical conventions of music notation that lead to complexity in layout. The difficulty of expressing certain other musical constructs in a natural way was acknowledged. Although the primary motivation for developing the model was pedagogical, the experience makes us optimistic that further development into a useful decoder for scanned music images is feasible.

References

1. P. Chou, ''Recognition of equations using a two-dimensional stochastic context-free grammar,'' *Conf. on Visual Communications and Image Processing, Proc. SPIE* **1199**, 852–863 (1989).
2. P. Chou and G. Kopec, ''A stochastic attribute grammar model of document production and its use in document image decoding,'' *Document Recognition II*, L. Vincent and H. Baird, Eds., *Proc. SPIE* **2422**, 66–73 (1995).
3. G. Kopec and P. Chou, ''Document image decoding using Markov source models,'' *IEEE Trans. Patt. Analy. Mach. Intell.* **16**(6), 602–617 (June 1994).
4. G. Read, *Music Notation*, 2nd ed. Taplinger, New York (1979).
5. Adobe Systems, Inc., *Sonata Font Design Specification*, Technical Note No. 5045 (Mar. 31, 1992).
6. D. Blostein and H. Baird, ''A critical survey of music image analysis,'' in *Structured Document Image Analysis*, H. Baird, H. Bunke, and K. Yamamoto, Eds., Springer-Verlag, Berlin (1992).
7. R. Rubenstein, *Digital Typography*, Addison-Wesley, Reading, MA (1988).
8. Adobe Systems Inc., *PostScript Language Reference Manual*, 2nd ed., Addison-Wesley, Reading, MA (1990).
9. J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA (1979).
10. H. Abelson and A. DiSessa, *Turtle Geometry*, MIT Press, Cambridge, MA (1980).
11. P. Prusinkiewicz and J. Hanan, ''Lindenmayer Systems, Fractals, and Plants,'' *Lecture Notes in Biomathematics*, No. 79, Springer-Verlag, Berlin (1989).
12. G. Kopec, ''Least-squares font metric estimation from images,'' *IEEE Trans. Image Processing* **2**(4), 510–519 (Oct. 1993).
13. A. Kam and G. Kopec, ''Separable source models for document image decoding,'' *Document Recognition II*, L. Vincent and H. Baird, Eds., *Proc. SPIE* **2422**, 84–97 (1995).

Biographies and photographs of authors not available.