Automatic Computer Recognition of Printed Music

K. Todd Reed J.R. Parker

Laboratory for Computer Vision
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada
T2N 1N4
E-mail: {reedt,parker}@cpsc.ucalgary.ca

Abstract

This paper provides an overview to the implementation of Lemon, *a complete optical music recognition system. Among the techniques employed by the implementation are: template matching, the Hough transform, line adjacency graphs, character profiles, and graph grammars. Experimental results, including comparisons with commercial systems, are provided.*

Keywords: Optical Music Recognition
Categories: Applications, Pattern Analysis, Segmentation

1. Introduction

This paper summarizes the implementation of *Lemon*[1], an optical music recognition system capable of recognizing printed music of reasonable quality. Like hand printed character recognition, optical music recognition (OMR) is a problem that has yet to be solved in a satisfactory manner. Although commercial systems are currently available, the slow adoption of OMR as a music acquisition method attests to the immaturity of the technology. Recognition rates are still relatively low, even for good quality machine printed documents.

Many of the techniques adopted in *Lemon* are derived from previous work in OMR [2, 4] or other areas of document analysis [5]. Hence, *Lemon* represents an evolutionary and incremental advancement, having chosen and refined the methods already proven to be successful. Some new approaches to music recognition are also discussed.

The paper provides an overview to the implementation techniques used, presented (more or less) in the order they are performed by the system: staff line detection, text segmentation, line detection, symbol recognition, note head recognition, and semantic interpretation. Finally, some experimental results are presented, comparing *Lemon* with two commercial systems, *MIDISCAN* and *NoteScan.*

2. Staff Lines

Finding the staff lines is a logical precursor to all other recognition tasks, since their orientation and location reveals any rotation in the image and where to look for all other music symbols.

The algorithm developed for staff detection uses the fact that there are five staff lines that are equally spaced in every staff. The algorithm operates by examining vertical columns from the image and recording all *staff samples* – five equally spaced, equally sized run-lengths of black pixels (Figure 1). The staff samples are grouped based on their spacing, thickness, and vertical position on the page. Within each group, the angles between all of the samples are computed, yielding a median angle α. Then the angles between adjacent samples are compared; if the angles between a sample and its neighbours disagree wildly with α, then the sample can be discarded. The remaining samples in each group are fitted to a straight line. When a staff line cannot be represented by a single line[2], the staff line is piecewise approximated by dividing the staff line until each division is linear, within a small tolerance (Figure 2).

Because the staff lines intersect almost all other symbols, their presence generally interferes with subsequent recogni-

[1] The (unfortunate) name *Lemon* was adopted during the initial development stages, when the notation editor *Lime* was going to be used. Since then, we adopted a different notation editor, but the original name stuck.

[2] Bowed staff lines can occur, for example, when the original score is taken from a book whose binding prevents the entire page from being flush against the photocopier/scanner glass

1015-4651/96 $5.00 © 1996 IEEE

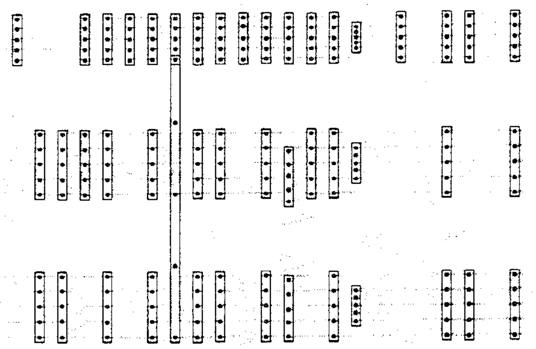

Figure 1. Staff samples used for detecting staff lines.

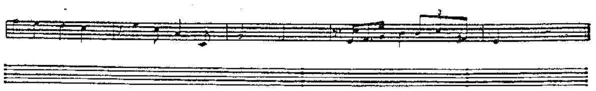

Figure 2. Curved staff lines are approximated by linear segments.

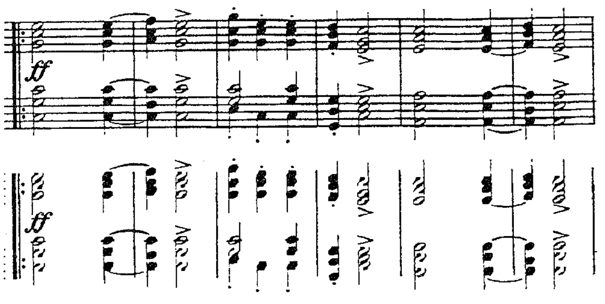

Figure 3. Top: original image. Bottom: after removing staff lines; many half notes become fragmented.

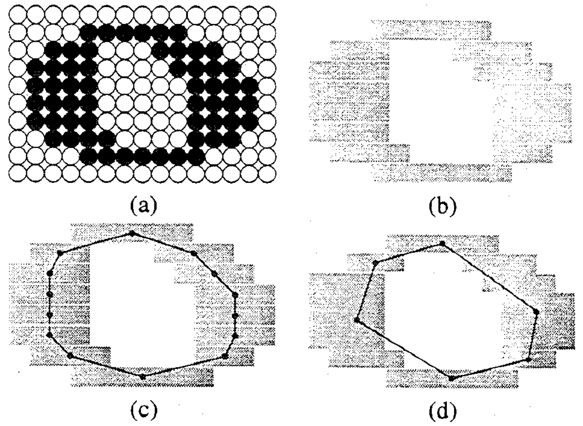

Figure 4. Line adjacency graphs. (a) A whole note. (b) Horizontal runs used to construct the LAG. (c) The resulting LAG. (d) The LAG after compression

tion. It is, therefore, advantageous, to remove the staff lines, attempting not to distort any intersecting symbols. The simple technique of traversing along each line, deleting black pixels except where the thickness exceeds a specified threshold (dependent on the staff line thickness) is used. Figure 3 shows an image before and after staff line removal. The technique works well, except that some symbols, such as whole and half notes, can become fragmented. Problems with note fragmentation are completely avoided by working with the original image when detecting note heads, as discussed later. For other symbols, however, fragmentation remains a problem.

3. Segmentation

Segmentation identifies regions of the image that contain music, text, and possibly decorative artwork. The regions containing music can them be addressed, leaving the rest as noise (or possibly calling upon an OCR engine to recognize the text).

Segmentation proceeds after the staff lines have been removed from the original image. The first task is identifying connect components. While the connected components can be found in a recursive tracing strategy, one efficient way to find and represent them uses line adjacency graphs (LAG). When traversing the image along either rows or columns, each run of black pixels becomes a node, the coordinates of which are those of the center pixel in the run. An edge is placed between two nodes if the runs associated with the nodes overlap, and the runs are on adjacent rows (columns). The basic idea is illustrated in Figure 4.

Now the LAG can be compressed, so as to occupy less space. This is accomplished by merging nodes A and B that have the following properties (assuming horizontal runlengths): (i) A is above B, (ii) A has degree $(a, 1)$ and B has degree $(1, B)$, (iii) A and B have nearly the same runlength, and (iv) the resulting amalgamated node can be represented approximately by a straight line. The *degree* of a node is the number of edges on each side (above and below). A node has degree (2,3) if there are 2 edges connecting to it from above and 3 edges connecting below. Figure 4 attempts to make this situation more clear.

When the overall connected component analysis is complete, we have a LAG for each component in the score (Figure 5). Since the LAG is found after the staff lines are removed, there is a good chance that each represents a symbol,

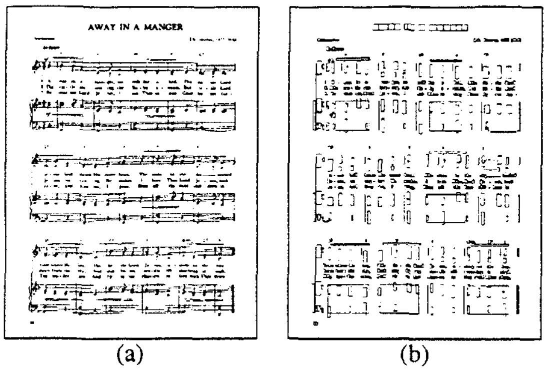

Figure 5. (a) Sample score. (b) Bounding boxes for all the connected components after staff lines have been removed.

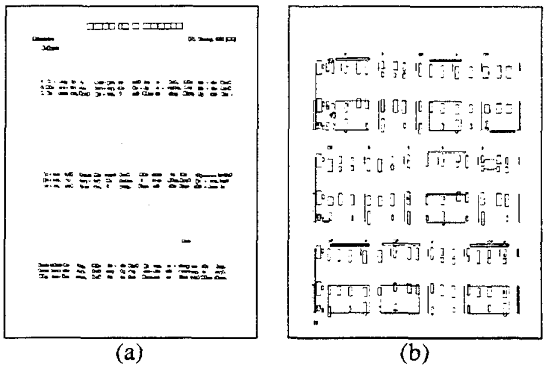

Figure 6. Text segmentation results. (a) Bounding boxes for text components. (b) Bounding boxes for non-text components.

or at least a set of related symbols. The final step of segmentation is to identify which connected components contain textual information, such as lyrics.

The text segmentation technique adopted is based on the algorithm described in [5], which was specifically designed for distinguishing between regions of text and graphics on a document page. The basic idea is that text consists of connected components that are oriented along a straight line. A set of collinear objects having a regular spacing forms a word, and may be removed since text is of no interest to the OMR system. The centroids of each connected region are fed into a Hough transform to determine collinearity. Then any LAG sets that are words, based on their spacing, are marked as such for later deletion. This is an oversimplification of a fairly complex algorithm. The result of this segmentation process is exemplified by Figure 6, in which a page from a sample score has had the text and non-text regions identified very accurately. This means that the recognition of music symbols can proceed, concentrating on the non-text regions. Some text may remain in the score, and

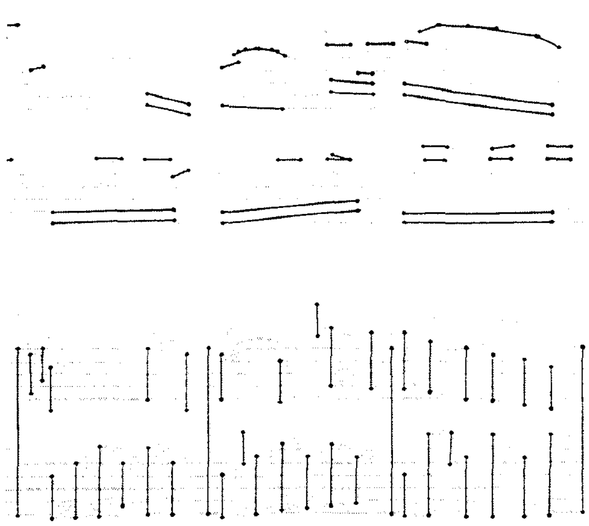

Figure 7. Detecting line segments using line adjacency graphs. Top: horizontal lines. Bottom: vertical lines.

may later be discarded by the matching process.

4. Music Symbol Recognition

Symbol recognition is split into three parts: line/curve detection, *character-symbol* recognition, and note head recognition. A character-symbol is any music symbol that appears in isolation, including accidentals, rests, and clefs.

4.1. Lines and Curves

Music notation consists predominately of straight lines: staff lines, stems, beams, etc. The primary reason for adopting the compressed LAG approach for connected component analysis was to aid the recognition of line segments. Every node in the compressed LAG represents a line segment, and adjacent lines can be merged if they are collinear to form longer lines. Curves can also be found by merging adjacent lines by relaxing the collinearity constraint. To find both horizontal and vertical lines, twos LAGs must be constructed: one by traversing the image along columns, the other along rows. Figure 7 shows sample results for line detection.

4.2. Accidentals, Rests, and Clefs

For recognizing accidentals, rests, and clefs, which are often isolated by staff line removal, the method of *character profiles* [6] was used. Character profiles are functions that

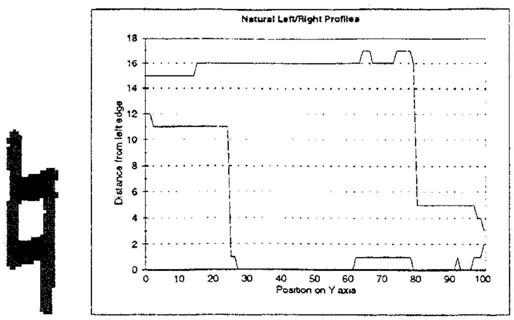

Figure 8. Sample left and right profiles for a natural symbol.

measure the perpendicular distance of the character's outer contour from some reference axis, usually the left or bottom edge of the character's bounding box. The right profile, for example, is obtained my measuring the horizontal distance between the left side of the character's bounding box (the reference axis) and the the right edge of the character. Left (L), right (R), top (T), and bottom (B) profiles can be measured; often height and width profiles are computed from $T - B$ and $R - L$ respectively. Figure 8 shows the left and right profiles for a sample natural.

A rule based classifier, using measurements obtained from the profiles, can be used to recognize the symbols. Common measurements include the location and magnitudes of local and absolute maxima and minima in the profiles. Rules are constructed by building a database of features from known symbols and identifying common and unique features for a symbol class. These rules are then used to determine what class an unknown symbol belongs to. The reader is referred to [6, 8] for a more detailed discussion of this technique.

The most significant shortcoming of this technique is that it requires the symbol to be isolated. Unfortunately, some symbols, especially accidentals, touch other symbols, preventing successful segmentation.

4.3. Note Head Recognition

Because note heads are usually attached to stems, and in the case of chords, attached to other note heads, it is not possible to isolate note heads by connect component analysis. This precludes using most stock algorithm for character recognition since they assume the character is already isolated. Hence, we resorted to template matching to recognize note heads. To avoid the problems of fragmentation introduced by the removal of staff lines, template matching is performed on the original image with the staff lines present. The staff lines, therefore, must be included in the template. This necessitates having several templates for each note head type to accommodate the different relative vertical positions a note can occur on a staff (Figure 9). Two refinements to the standard template matching technique were implemented to improve accuracy and run-time performance.

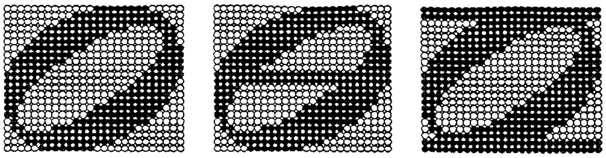

Figure 9. Templates used for detecting half notes : original template, on a staff line, and between staff lines. Though the current implementation of *Lemon* does not use them, additional templates are needed for notes appearing above or below a ledger line.

One shortcoming of conventional (binary) template matching techniques is that they only consider foreground pixels. Background pixels cannot contribute to a positive match. When the discriminating feature of a symbol is found in its interior background pixels (such as the holes found in half and whole notes), template matching accuracy may be inadequate. To improve the accuracy of template matching in such cases, template pixels are labelled as one of: *foreground pixel, background pixel*, or *interior background pixel*. The interior background pixels are matched against background pixels in the test image, and contribute to a positive match *only* when sufficiently many foreground pixels match. See [7] for details.

The most significant drawback of template matching is its poor run-time performance. Indeed, prior to implementating any optimizations, template matching consumed more than 90% of the total running of *Lemon*. An effective technique for optimizing template matching, which we call *partial templates*, is as follows. The original template is partitioned into multiple templates such that each partition samples the original. When testing for a match, each partition participates in one pass over the test image. After each pass, an interim match index is computed, which approximates the actual match index obtained by comparing each pixel. A sufficiently low interim match index aborts the template match, thus avoiding the cost of comparing every pixel when a mismatch is obvious. In the current implementation, ·· :te head templates are partitioned into four partial templates, such that each partial template uniformly samples a quarter of the pixels. Run-time performance is improved by 50-65% by this approach. This technique is a generalization of *two-stage template matching* [9].

5. Contextual Recognition

Once the individual symbols have been found, they are synthesized into more complex forms by using *graph gram-*

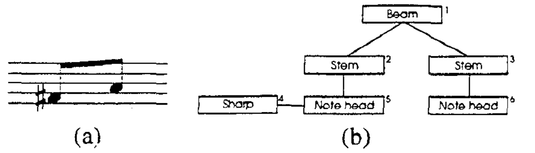

Figure 10. Using a graph representation to incorporate context. (a) A small sample containing six primitives. (b) The graph representation of (a).

mars [4, 7]. A disconnected graph is created from the primitive symbols in the score, and then "parsed" to perform high-level recognition. For example, a sharp, two note heads, two stems, and a beam, under certain geometric/spatial constraints, can be combined into a single high-level symbol, as illustrated in Figure 10. This technique provides the opportunity to use the context of symbols to determine their syntactic validity (using rules of music notation), and possibly correct some recognition errors. From the resulting graph representation, the score can be converted to some notation interchange format[3] or a MIDI file.

6. Experimental Results

Lemon was tested on a collection of ten scores[4], and the results compared against those obtained from two commercially available systems, *MIDISCAN* and *NoteScan*. Although *Lemon* was consistently slower than both commercial systems, it posted higher recognition rates and significantly fewer false identifications, on average. These results are summarized in Table 1 and Table 2. The average running for *Lemon* was slightly over 4 minutes, running on a 486/33 PC.

A critical limitation with the current implementation of *Lemon* is that it cannot recognize polyphonic scores, *i.e.* scores with more than one voice per staff. This limitation is not present in either commercial system.

Score	**Symbols**	**Recognition Rate**		
		Lemon	*MIDISCAN*	*NoteScan*
Magnificat	239	**99%**	84%	88%
Canon in D	206	**99%**	88%	73%
Concerto No. 1	247	95%	**98%**	94%
Polovetzian Dance	281	**98%**	93%	86%
The Entertainer	233	94%	**99%**	64%
Für Elise	534	**99%**	96%	83%
Moonlight Sonata	179	**96%**	**96%**	74%
Sonata in C Major	256	**99%**	95%	88%
The Four Seasons (Winter)	264	81%	**84%**	79%
Antiphonae Marianae	214	80%	87%	**90%**

Table 1. Comparison of per score recognition rates. The best result in each row is printed in **bold**.

Score	**Symbols**	**False Identifications**		
		Lemon	*MIDISCAN*	*NoteScan*
Magnificat	239	**4**	39	24
Canon in D	206	**1**	2	5
Concerto No. 1	247	9	**5**	**5**
Polovetzian Dance	281	**6**	9	**6**
The Entertainer	233	**3**	**3**	7
Für Elise	534	**5**	30	79
Moonlight Sonata	179	**0**	3	23
Sonata in C Major	256	**2**	5	16
The Four Seasons (Winter)	264	24	**18**	25
Antiphonae Marianae	214	11	16	**10**

Table 2. Comparison of per score false identifications. The best result in each row is printed in **bold**.

References

[1] H. Baird, H. Bunke, and K. Yamamoto, editors. *Structured Document Image Analysis*. Springer-Verlag, 1992.

[2] N. P. Carter and R. A. Bacon. Automatic recognition of printed music. In Baird et al. [1], pages 456–465.

[3] R. Erickson and A. B. Wolff. *Computers in Language Research 2 (Trends in Linguitics: Studies and Monographs 19)*, chapter The DARMS Project: Implementation of an Artificial Language for the Representation of Music, pages 171–219. Mouton Publishers, 1983.

[4] H. Fahmy and D. Blostein. Graph grammar processing of uncertain data. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition: Proceedings of the International Workshop on Structural and Syntactic Pattern Recognition*, pages 373–382, Bern, Switzerland, August 1992.

[5] L. A. Fletcher and R. Kasturi. A robust algorithm for text string separation from mixed text/graphics images. In L. O'Gorman and R. Kasturi, editors, *Document Image Analysis*, pages 435–443. IEEE Computer Society Press, 1995. Reprinted from *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 6, Nov. 1988, pp. 910–918.

[6] F. Kimura and M. Shridhar. Handwritten numerical recognition based on multiple algorithms. *Pattern Recognition*, 24(10):969–983, 1991.

[7] K. T. Reed. Optical music recognition. Master's thesis, University of Calgary, Calgary, Alberta, Canada, 1995.

[8] M. Shridhar and A. Badreldin. Recognition of isolated and simply connected handwritten numerals. *Pattern Recognition*, 19(1):1–12, 1986.

[9] G. J. Vanderbrug and A. Rosenfeld. Two-stage template matching. *IEEE Transactions on Computers*, 26(4):384–393, April 1977.

[3] Currently *Lemon* uses DARMS [3].

[4] Only the first page of each score was used.