THE UNIVERSITY OF CALGARY

Optical Music Recognition

BY

Todd Reed

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 1995

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Optical Music Recognition" submitted by Todd Reed in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor, J.R. Parker

Department of Computer Science

Jon Rokne

Department of Computer Science

David Eagle

Department of Music

Sept 20, 1995

Date

Abstract

This thesis investigates the problem of automatically converting printed music scores into a computer-readable music representation language – or more succinctly, endowing a computer with the ability to "read" music. This task is commonly termed *optical music recognition*, and abbreviated OMR. Several computer vision techniques are examined and used to implement a working OMR system. Among the techniques investigated are template matching, the Hough transform, line adjacency graphs, character profiles, and graph grammars. Initial experiments indicate recognition rates in excess of 95% are obtainable for good quality music of moderate complexity. Furthermore, the developed OMR system betters the performance of several commercially available systems.

Acknowledgements

Thanks to Jim Parker, my supervisor, for providing me with both freedom and direction in my research; to Brian Gaines, for his patience while I considered many research topics; and to Brian Wyvill and Ian Witten for tempting me with alternative research topics. Thanks to my fellow students, who provided suggestions, inspiriations, and distractions during my research. Thanks to Curtis Jensen and the Integrated Arts Media Lab at the University of Calgary. This research was supported by a scholarship from the Natural Sciences and Engineering Research Council of Canada.

Todd Reed
The University of Calgary
September 1995

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The paperless society is rapidly approaching, whether we like it or not.
— F.W. Lancaster, Towards Paperless Information System, 1978

Nowadays, the electronic document is a well established publishing medium. The appeal of electronic documents is that they are much more amenable to modification, distribution, storage, and retrieval than their paper counterparts. However, despite the once promised vision of a paperless society, computer technology has fueled rather than curtailed the printing of paper documents. Consequently, paper and electronic documents will coexist as popular media formats for the foreseeable future. This situation has generated much interest in developing automated techniques for converting paper documents to an equivalent electronic representation. Even when paper becomes passé (if ever), the vast accumulation of already existing paper documents will likely secure the need for computerized document recognition systems for many more years.

This thesis investigates the problem of automatically converting printed music scores into a computer-readable music representation language – or more succinctly,

endowing a computer with the ability to "read" music. This task is commonly termed *optical music recognition*, and abbreviated OMR.

The motivation for developing an OMR system is provided by the numerous applications that could benefit from such a system [Blostein 92a, Duggan 92, Kassler 70, SF94]:

- The adaptation of scores for other instrumentations, such as producing an instrument-specific scores from an orchestral score, or vice versa.
- The reproduction of old scores where the quality of the original is unsatisfactory or the notation dated.
- The preparation of braille or large-printed scores for visually impaired musicians.
- The creation of music databases for musicological research.
- The audio rendition of a score.
- Computer-based editing and inexpensive publishing of new music.
- Teaching students music theory through interactive music applications capable of displaying music notation and producing audio output.
- Electronic distribution of music.

Although applications have been identified, there are at least two reasons computer applications in music have not flourished:

- There does not exist a universally accepted music representation language, hence the free exchange of music information is not possible. To understand the significance of this, consider the analogous problem that would exist for textual

information in the absence of ASCII or a similar standard. A survey of the literature revealed a superfluity of music representation languages of research and commercial origin: MIDI, DARMS[1], SMX, MUSICA, MUSTRAN, ALMA, and SCORE, to name a few.

An ANSI committee has been formed to generate a standard, currently termed SMDL (Standard Music Description Language). Although drafts of the proposed standard have been available since 1988, SMDL has yet to gain acceptance and faces the possibility of being stillborn when finally completed.

Recently, a new industry supported format called NIFF (Notation Interchange File Format) was announced. Many music software companies have pledged to support the format, likely securing it as the de facto standard. However, universal acceptance remains an elusive goal; Coda Technology, one of companies who sponsored and participated in the development of NIFF, has announced they will publish their own proprietary format in lieu of supporting NIFF.

- The acquisition of music scores to a machine readable form remains a tedious and time consuming task. Machine readable music representations have been sought since 1970, when punch-card representations of music were manually keyed [Kassler 70]. Current methods include keyboard entry, entry via graphical-oriented music editors, or MIDI instrumental performances. In practice, the latter technique requires additional editing to capture lyrical text, dynamic markings, etc. Direct sound-to-score systems for monophony have been developed, but extending such systems for polyphony remains an open area of research. See [Carter 88] for a complete discussion of music acquisition techniques.

This thesis addresses the latter problem of music acquisition. Unlike traditional

[1]Formerly called the Ford-Columbia Music Representation.

input methods, the operator of an OMR system needs no musical background or knowledge of music encoding schemes used for keyboard entry. A reliable OMR system would make many applications cited above more practical and others feasible. Many authors have even suggested that a practical OMR system will be the saviour of musicology. Kassler, writes:

> It is remarkable, nonetheless, that (of all things) this technology may cause return of musicologists' attention to the core concepts of their field which constitute musical theory [Kassler 72].

More recently, Fujinaga similarly notes:

> A practical and relatively inexpensive optical music recognition system would allow a revitalization of computer-assisted research in musicology, ...[Fujinaga 88].

1.1 Research Goal

The primary goal of this research was to develop a complete optical music recognition system, capable of audibly playing or printing a recognized score. The direction of research was governed by previous work in the field. Because this effort was preceded by several similar research projects, many techniques for optical music recognition have already been investigated. An attempt was made to either investigate alternative algorithms or develop improvements to existing algorithms.

1.2 Thesis Overview

Chapter 2 provides a comprehensive introduction to optical music recognition. The chapter includes a discussion of pattern recognition aimed at familiarizing the reader

with the problems of computer vision. A survey of previous work in the field is also presented.

Chapters 3, 4, and 5 describes the experiments conducted and algorithms implemented in the developed OMR system.

Chapter 6 evaluates the results of this research. The developed OMR system is tested using a small corpus of selected scores, from which recognition rates are obtained. The results are analyzed and compared with results reported in the literature and those obtained by two commercial OMR systems.

Finally, Chapter 7, summarizes the results and contributions made in this thesis, and outlines avenues for continued research.

Chapter 2

Background

The purpose of this chapter is to familiarize the reader with all aspects of optical music recognition. First, an introduction to pattern recognition, and specifically visual pattern recognition is given. Then, the problem of optical music recognition is defined. The next section presents a survey of past research in the field, whilst introducing pattern recognition techniques. Finally, an evaluation of the state-of-the-art in optical music recognition is made.

2.1 Pattern Recognition

Pattern recognition is most easily defined in terms of its primary goal: to identify objects based on their measurment patterns. The term *object* must be interpreted in its most generic sense, since the objects of interest may not be physical. A simple example of a pattern recognition problem would be determining the season based on average temperatures over several days. The objects to be identified are the four seasons, and the measurement patterns consist of daily temperature readings. Given temperature readings of $-22°$, $-12°$, and $-19°$, a reasonable season classification

would be "winter". Of course, this classification depends on the geographic location where the temperatures were taken. The use of *domain knowledge* is usually a critical part of any pattern recognizer. Had the above temperatures been taken at Alert, Canada's most northern community, "spring" would likely be a better season classification.

Formal definitions of pattern recognition vary. Schalkoff describes pattern recognition as an information reduction process, transforming large data samples, usually obtained empirically, to a more compact and elucidative form [Schalkoff 92]. Parker, on the other hand, describes pattern recognition as an analysis, and then synthesis, process [Parker 94a]. Analysis decomposes the subject into fundamental parts, and synthesis combines these parts, yielding the desired result. Much of the science of pattern recognition aims at finding optimal techniques for synthesis.

Document analysis, and hence optical music recognition, is concerned with recognizing 2-dimensional visual patterns. Visual patterns are essentially pictures, called *digitial images* or just *images* in computer science. An image is usually mapped onto a discrete Cartesian plane, and each possible (x, y) coordinate, called a *pixel*, is assigned a numerical value, denoted $I(x, y)$. The interpretation of $I(x, y)$ is application dependent, but typically represents a color. It is from the collective values of all the $I(x, y)$'s that pattern recognition tries to extract meaning.

Humans are expert visual pattern recognizers. Furthermore, we recognize visual patterns effortlessly. Despite our natural ability to interpret what we see, our attempts to automate this ability have produced comparatively feeble vision systems. Unlike the vision systems found in nature, computer-based vision systems rely on specialized mathematical techniques that manipulate numerical data to deduce relevant meaning.

To illustrate why algorithmic visual pattern recognition is so hard, consider the task of analyzing a picture without actually looking at the picture. The only information provided is the set of numbers, $I(x, y)$, representing the intensity of the picture

at various locations. To appreciate the difficulty of this, the reader is invited to solve the puzzle in Figure 2.1.

68	57	36	63	56	26	44	44	23	45	37	22	33	45	22	30	37	22	22	44	23	38	22	20	20
58	62	57	53	58	55	44	33	45	47	31	42	31	27	36	34	25	33	25	29	23	32	25	22	27
45	61	59	42	58	65	53	45	59	40	36	51	32	47	33	29	42	30	22	35	26	21	36	33	24	29	19
60	53	37	62	58	39	63	56	32	48	47	24	30	43	24	47	22	29	42	26	24	44	22	40	23	36	19
44	38	46	48	41	41	42	33	32	35	31	28	23	24	24	26	29	30	29	42	24	27	29	25	23	21	26
29	33	43	28	34	47	25	24	39	25	24	39	24	20	29	22	19	26	19	34	26	20	31	20	25	28	18	25	31
36	30	25	46	41	26	44	35	22	42	24	42	29	34	19	22	29	19	22	31	19	22	34	21	22	35	20	19	34	23	21
29	27	41	51	41	44	48	32	39	48	37	40	42	30	24	22	23	21	22	24	22	23	25	20	24	25	22	26
22	34	54	39	51	64	35	27	47	29	28	48	29	22	34	22	18	24	18	31	26	18	27	27	18	29	33	19	25	35
38	38	34	61	55	33	33	33	24	38	37	22	22	31	19	22	30	20	21	31	20	23	34	19	20	34	23	38	25	21
35	30	31	39	31	25	28	30	27	32	29	22	22	25	21	22	25	24	22	29	24	31	25	25	34	23	27
20	32	40	21	27	50	27	33	50	25	19	37	21	19	33	21	19	33	26	18	33	29	20	36	45	25	45	40	21	28	38
31	26	22	32	37	32	47	58	35	28	27	20	25	30	19	22	29	21	20	33	22	33	32	25	51	45	31	43	26	21
30	25	23	34	29	32	60	67	47	41	27	21	27	29	24	27	29	22	29	25	24	32	26	47	50	36	48	56	27	21
20	24	38	22	31	57	51	56	69	42	21	39	25	22	45	29	20	39	32	29	56	41	24	45	66	39	52	62	40	42	40
29	25	24	32	43	46	65	72	63	62	40	25	29	35	29	34	44	33	44	64	60	47	49	44	68	69	60	45	66	59	36
33	24	22	39	40	44	69	71	62	70	56	28	31	39	31	49	59	48	64	73	62	60	67	65	76	78	68	55	64	55	42
20	21	36	24	37	65	58	60	76	71	63	63	37	25	51	49	44	71	70	55	77	73	59	81	82	76	78	71	44	61	69
23	24	27	31	38	53	71	76	80	77	73	66	49	39	43	58	66	33	44	71	71	79	80	85	86	83	79	74	69	73	71
31	27	26	46	38	45	79	82	81	84	82	77	80	76	61	70	79	69	79	84	82	86	86	84	85	84	76	70	80	75	71
20	33	51	29	36	69	64	75	85	83	82	85	84	81	82	81	79	87	89	86	87	88	87	86	87	73	84	84	62	80	84
27	42	44	40	57	53	60	74	73	82	84	84	87	87	85	85	83	86	88	88	88	89	87	83	84	83	85	83	80	82	73
37	33	29	58	62	46	70	62	50	78	85	80	85	85	81	87	87	83	85	88	85	86	86	80	80	87	83	80	84	76	54
23	29	49	37	55	73	61	50	67	71	84	76	70	83	82	72	86	83	67	85	85	77	86	87	75	86	86	67	76	75
24	28	34	62	74	72	75	80	78	73	79	78	74	83	82	74	83	82	74	83	85	85	84	81	84	85	81	80	78	71	74	66
39	29	23	45	56	50	79	76	59	78	82	70	81	81	64	78	82	77	82	86	74	80	84	80	76	84	67	58	78	69	53
24	27	39	25	36	64	60	58	76	75	73	82	78	72	80	77	85	84	78	82	86	79	79	66	79	57	73	53	73	75
28	30	34	26	43	54	52	50	60	69	71	78	76	77	74	77	78	80	79	76	75	70	70	51	62	65	55	69	69	64	75	69
59	38	21	40	45	29	55	50	31	61	61	39	69	76	44	47	69	67	76	78	55	67	76	44	50	70	53	52	74	62	73	57	78	71	54
47	34	38	28	30	42	35	34	50	42	35	47	51	42	62	59	67	71	49	49	54	49	69	72	62	73	75	61	74	79
46	51	45	33	35	38	27	34	50	41	42	52	45	45	56	45	48	67	55	48	62	60	53	70	70	59	75	76	58	76	81
76	65	36	59	51	26	44	44	28	53	60	33	52	60	33	51	67	45	47	69	53	56	76	53	60	79	73	66	80	74	65
67	67	67	62	50	58	49	37	53	51	42	55	64	38	52	54	45	56	58	52	72	71	55	60	75	68	78	81	75	80	79
63	69	76	60	59	70	55	55	67	56	51	63	59	51	71	66	51	74	70	56	76	75	55	74	81	72	84	84	73	82	84

Figure 2.1: What is it? A data set for a gray-scale image, organized as a Cartesian grid; each value provides the brightness of the image on a scale of 100 (100 being white). See footnote (2) on page 12 for the answer.

2.1.1 Document Image Analysis

Document image analysis is a sub-discipline of pattern recognition that addresses the task of automatic document interpretation. Research in the field is motivated by the commercial application of converting existing paper documents to electronic equivalents. The most common example of document image analysis is *optical character recognition* (OCR), which converts printed text to a computer readable ASCII[1] file.

There are five distinct steps commonly adopted by document image analysis systems:

Pre-processing Pre-processing aims at preparing an image for analysis. Image processing techniques are employed to improve image quality by removing noise. In document analysis, correcting the orientation of the image from unintentional rotation is necessary. For small rotations, the shear operation is sufficient and computationally cheaper than a proper rotation. If a rotation is required, the technique of reverse rotating has been shown to reduce fragmentation [Witten 94].

Segmentation Segmentation identifies and labels regions that require analysis. In OMR, the goal of segmentation is locating the systems, staves, music symbols, and possibly lyrical underlay and other textual information. The pre-processing and segmentation steps are not entirely disjoint in most OMR systems. Researchers almost unanimously agree that removing the staff lines is necessary before proceeding with segmentation and recognition of the remaining symbols; image rotation follows logically since the staff lines provide the best clue of orientation. In the context of OMR, segmentation is complicated by the intricate layout of music notation. The common technique of region filling cannot be directly applied since symbols are superimposed

[1] American Standard Code for Information Interchange – *the* standard computer representation for textual data.

and intersecting.

Feature Extraction Feature extraction is the process of obtaining discriminating measurements from an input image. There are an infinite number of measurements that can be made: mass, area, perimeter, moments, etc. Closely related to feature extraction is feature selection: from the infinite set of available measurements, selecting a useful subset from which the underlying symbol contained in the image can be unambiguously identified. Clearly, feature selection is mandatory to make recognition computationally feasible (we can't possibly make *all* measurements!). While it seems intuitive that more measurements will lead to more robust recognition systems, it is important to consider the dimensionality of the feature space. The number of training samples available should influence the choice of features used. High dimensional feature spaces require more populous training sets, which are not always readily available. Even when training sets are plentiful, dimensionality reduction can improve recognition performance. Techniques exist for finding the optimal subset of features [Kittler 86]. In practice, *a priori* knowledge, experience, and experimentation are often used to find a workable set of features.

Classification The final step of analysis is classification. A myriad of pattern recognition techniques are available, including template matching, neural networks, decision trees, and distance measurements of feature vectors. Of these approaches, the latter technique is the most common. One boon of music notation is the variation of symbol size and shape, making these features ideal candidates for preliminary recognition based on decision trees or distance measurements.

Description Complex, structured documents warrant an addition step, sometimes called *description.* Description is, aptly, the process of describing the semantics of

a document. For musical scores, the 2-dimensional geometric layout of the symbols must be analyzed to infer an interpretation of the music.

2.2 Optical Music Recognition

2.2.1 Definition

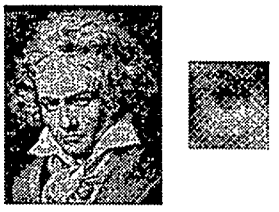
Optical music recognition is a method for music acquisition. The pages of a score are digitially replicated by a scanner. The digitized pages are then analyzed by the OMR system. The OMR system recognizes and interprets the musical symbols present, and produces a symbolic computer representation for the notated music. From this symbolic representation, additional software can be used to print, play, or edit the music.

2.2.2 History

Research in optical music recognition was initiated at MIT by Pruslin [Pruslin 66] and Prerau [Prerau 70, Prerau 75] in the late sixties. These efforts produced marginal results: Pruslin's system recognized only quarter notes and beamed note groups; Prerau's system was significantly more sophisticated, but Prerau's decision to exclude the recognition of chords limited his system to but the simplist monophonic scores. Based on the literature, research in OMR was idle during the seventies and early eighties, but henceforth the field has spawned into an active area of research. Indeed, in [SF94], the author identified 36 research efforts to develop an OMR system.

---

[2] The answer to the puzzle in Figure 2.1 on page 9. The data set represents the gray levels of the eye image taken from a digitized portrait of Ludwig van Beethoven.



2.2.3 Music Notation

The notational language of music does not lend itself to automated understanding. Music notation has evolved over the centuries as composers introduce new styles and methods of notation. This trend will inevitably continue as musicians experiment with new techniques and instruments. Although general rules for notating music are well established (modern music notation has been in use since the 13th century), examples of dated or obscure notation are plentiful. Besides conventional notation, tablature, shape-note, and mensural notation are still in use today to varying degrees[3]. The extreme notational style is graphic notation, which, by definition, is non-standard:

> Graphic notation systems use new symbols ... to avoid the specificity of conventional systems, ...
>
> — from the definition of *graphic notation* in [Morehead 92]

Even conventional notation, however, is not void of ambiguity. Stone writes:

> The example on page 102 from Impromptu fantasque (1973) ... is representative of the large number of compositions that depend chiefly on effects and contain large numbers of new notational signs and symbols [Stone 80].

Figure 2.2 provides a collection of music examples that illustrate the large variation of notation styles and conventions. It is expected that any OMR system will be deficient – music notation is too varied for a system to recognize all forms of notation. The challenge is to successfully recognize a well established subset of music symbols *and* passively ignore unknown symbols, which may be plentiful.

[3]Tablature is used primarily for notating guitar music and is very common. The other styles are considerably less common; Carter, however, adapted his OMR system to read mensural notation [Carter 92b].

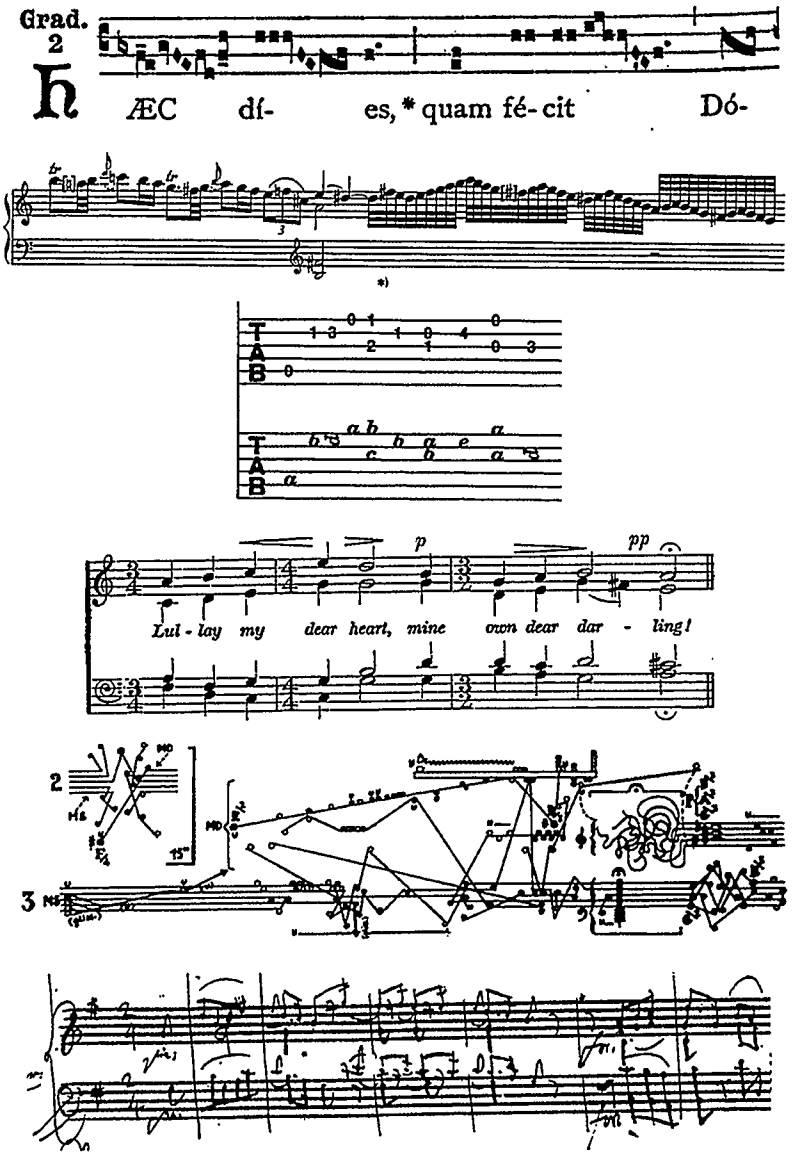


Figure 2.2: Examples of music notation. From top-to-bottom: Gregorian chant notated in 14th c. neume notation (from [Kamien 90]) — A staff taken from a score that has no time signature and no bar lines (from [Mozart 77]) — Guitar tablature notation — A score typeset in a peculiar font. Also note the repeated change of time signature (from [Dearmer 45], pp. 182) — A sample of graphic notation (from [Bussotti 59]) — A sample of handwritten music (from [Nettl 49]); this level of neatness is typical of handwritten scores.

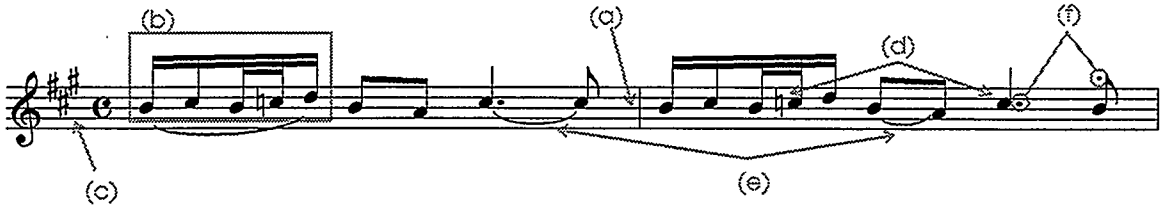


Figure 2.3: The above sample of music notation illustrates some of the difficulties of recognition process. (a) The staff intersects virtually all symbols to be recognized. (b) Compound symbols are formed by aggregating primitive symbols; in this case, note heads, stems, and beams form a note group. (c) The key signature affects the semantics, in particular the pitch, of the notes. (d) The accidental overrides the key signature, and affects all subsequent notes with the same vertical spacing up to the next barline. (e) The left curve is a tie, while the right curve is a slur; the former implies summation of the note durations on either side; the latter is an articulation mark suggesting the notes should be played legato. (f) The semantics of the dots: the left dot is a duration dot, the right dot a staccato dot.

Another aspect of music notation that complicates automated recognition is its intricate syntax and semantics: music symbols are superimposed; symbols are aggregated to form complex compound symbols; physically distant symbols affect local semantics, and identical symbols can have different semantics (see Figure 2.3). By comparison, in OCR, the symbols are disjoint, finite in number, and their correct classification fulfills semantic interpretation requirements. For these reasons, solutions to the OCR problem, which are well researched, are often unsuccessful in the domain of OMR[4].

2.3 Approaches to OMR

This section has two simultaneous goals: introduce methodologies and techniques of computer vision applicable to document analysis, and report on previous work in OMR. Numerous approaches to OMR have been reported in the literature, and a

[4]Although, OCR techniques can certainly be applied to certain aspects of OMR. See Section 4.2 on page 53.

complete survey is beyond the scope of this thesis. Rather, this section will introduce the techniques that are both recent and frequently cited in the literature, or relevant to subsequent work presented herein. For a complete and comprehensive survey see [Blostein 92a] and [SF94].

2.3.1 Projections

The application of projections for segmentation and classification of musical images is extensively investigated in [Fujinaga 88].

A projection is defined as [Pavlidis 82]:

$$p(\rho, \theta) = \int_D I(x, y)\delta(\rho - x\cos\theta - y\sin\theta)\, dx\, dy \tag{2.1}$$

where D is the image domain, δ is the Dirac delta function[5], and I is the image. For binary images it is customary to define I as:

$$I(x, y) = \begin{cases} 0 & \text{if the point } (x, y) \text{ is white} \\ 1 & \text{if the point } (x, y) \text{ is black} \end{cases}$$

The value $p(\rho, \theta)$ is equal to the sum of the pixel values along the line parameterized by $\rho - x\cos\theta - y\sin\theta = 0$; for bilevel images, this is simply the number of black pixels along the line. A series of projections with varying ρ and fixed θ form a parallel projection, which can be graphically presented as a histogram; such histograms are often called projection profiles. Of particular interest are x (or vertical) and y (or horizontal) projection profiles, obtained when $\theta = 0$ and $\theta = \pi/2$, respectively (see Figure 2.4).

[5]The scalar Dirac delta function satisfies the following properties:

$$\delta(x - a) = 0\ \forall\, x \neq a \text{ and } \int_{-\infty}^{+\infty} \delta(x - a)\, dx = 1$$

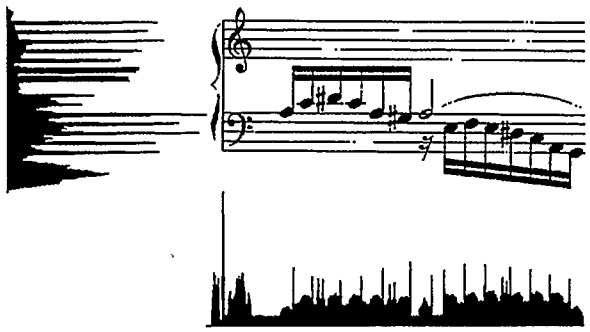


Figure 2.4: x and y projections (bottom and left, respectively) of a section of music (from [Mozart 77]).

To locate the position of staves, Fujinaga suggests taking a y-projection of the entire image. In the absence of excessive rotation, the staff lines can be identified by peaks in the projection profile. Five equally spaced peaks indicate the presence of a staff, and each peak provides the y-coordinates of a constituent staff line. This technique, or more robust variations of it, are used by [Bulis 92, Clarke 88, Martin 91, Randriamah 93, Kato 92, Baumann 92, Wolman 92]. Most variations partition the staves into vertical strips, and then perform y-projections on each strip, thus minimizing the effects of rotation.

Fujinaga uses x-projections of each staff to locate the position of symbols, and even to classify sharps and flats in the key signature; otherwise, symbols are classified by width and height. Symbols with similar dimensions are differentiated by projections. Bulis *et al.* also use projections for classification [Bulis 92]. Both of these efforts to use projections for classification were limited to monophonic music. Recognition rates reported in [Fujinaga 88] range from 64–82%, suggesting projections are insufficient for classifying music symbols. Fujinaga *et al.* has since adopted a k-NN classifier, using a number of geometrical feature measurements [Fujinaga 92].

2.3.2 Line Adjacency Graphs

A line adjacency graph (LAG) [Pavlidis 82] is a representation of a binary image based on run lengths of black (or white) pixels. The image is run length encoded, and a graph is constructed as follows: each run length becomes a node; an edge between two nodes is added if the associated run lengths overlap (see Figure 2.5).

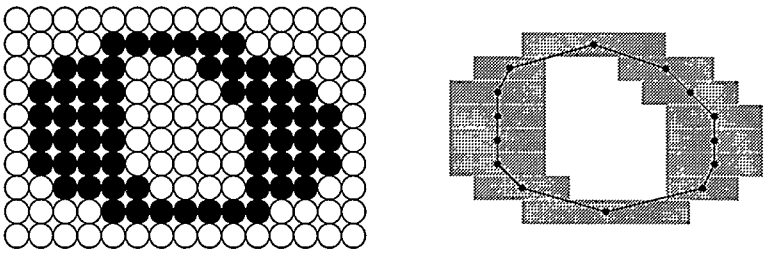


Figure 2.5: Example of a LAG representing an isolated whole note.

Carter uses a LAG based on vertical run lengths for segmentation and recognition [Carter 92c], noting several advantages gained by employing LAGs: pixel based operations are avoided, thus reducing raw data and speeding processing time; and, a LAG provides a first approximation to the structure of symbols. Size, aspect ratio, and LAGs based on horizontal run length encoding are used for symbol recognition. In addition to conventional notation, Carter has adapted his system for the recognition of seventeenth century white mensural notation, providing evidence that the LAG approach is insensitive to poor image quality.

Carter is considered a leading figure in the OMR community. McGee writes:

> In our view an understanding of the work of Prerau and Nicholas Carter [text deleted] is essential to progress in the field [McGee 94].

Leplumey *et al.* also adopt a LAG approach and describe a technique to find music staves that suffer from discontinuities, curvature, and inclination. This work was motivated by the success of Carter's system:

> The method developped [sic] by N.P. Carter seems to be the most robust with regard to these factors [speaking of staff line slope, curvature and discontinuities] [Leplumey 93].

2.3.3 Neural Networks

Artificial neural networks are decision-making machines modeled after biological neural networks. In computer vision, the term *perceptron* is often used to describe a neural network designed for pictorial pattern recognition. A description of how perceptrons work is beyond the scope of this discussion, and it is sufficient of treat them as black-box filters that convert their input (an image) to a meaningful output (a classification). There is a plethora of literature available on neural networks, for example [Schalkoff 92].

In the field of OMR, the use of perceptrons has been reported in [Martin 91, Martin 92, Yadid 92].

Neural nets have been used for staff line removal. Precautions must be taken when removing staff lines so that pixels that belong to both a staff line and at least one other symbol are not removed, otherwise the resulting image is so distorted that subsequent processing is impossible. In [Clarke 88] the authors solve the problem by examining a fixed set of pixels above and below the target staff line pixel and applying a simple algorithm which decides whether the target pixel should be cleared or not. This approach has been adopted by others including [Bainbridge 91] and [Baumann 92]. Martin and Bellissant [Martin 91] suggest a more complex and robust technique. For each target staff line pixel, a histogram of radial chord lengths is constructed. Each chord length measures how many contiguous black pixels radiate at a specific angle from the target pixel. A neural network, previously trained from sample histograms, decides if the target pixel is part of another symbol based on

the input histogram. Martin and Bellissant provide an example demonstrating the superior performance of their technique compared to the method described by Clarke. It is unclear, however, if the problem warrants the complexity of a neural net. As demonstrated later, simpler techniques give workable results.

Martin and Bellissant also use a perceptron for recognition of symbols, but, speaking of perceptrons, concluded:

> their performances in the classification area are less impressive when compared to statistical methods; we noticed, as others before, that a nearest neighbor classifier is usually enough to reach the same recognition rates than the best multi-layer perceptron [Martin 91].

Yadid *et al.* also investigated the use of neural networks for symbol recognition, reporting recognition rates of 90–100% for "simple notes" and 70-80% for "complex symbols."

2.3.4 Grammars

Although the analysis of music notation is a natural application of grammars and language theory, in practice grammars have proved to have limited utility. Fujinaga suggests using a music grammar to assist the classification phase of OMR by restricting the possible symbols that can occur based on context [Fujinaga 88]. Ensuring each bar contains the beats indicated by the time signature is an obvious way in which recognition can be assisted. However, as noted in [SF94]:

> [Notational anomalies] interfere with a grammatical approach to object recognition, since in its graphic presentation the underlying logic of notation is beset on every side with exceptions of an unpredictable nature.

Indeed, Fugjinaga concedes that exceptions to the beats/bar rule exist.

In [Andrònico 82], the authors describe a two-tier music grammar: a high level grammar describing staves, bars, and notes, and a low level grammar with positional operators to describe how primitive symbols can be combined to form notes.

The drawback of using traditional grammars for describing music is that they are inherently serial descriptions, while music notation is bi-dimensional and capable of relating parallel information. To avoid the restriction of traditional string grammars, in [Fahmy 92, Fahmy 93][6] the authors use a graph grammar to semantically interpret a collection of music symbols. Their system assumes the existence of a symbol classifier that provides the location and type of all important primitive music symbols; the system then interprets the symbols and forms a graph revealing the semantics. This thesis further investigates the applicability of graph grammars for music recognition in Chapter 5, using the results in [Fahmy 92, Fahmy 93] as a foundation.

Most recently, in [Couasnon 94, Couasnon 95] the authors use a grammar to control the segmentation and recognition algorithms. The authors claim their approach provides an effective solution to problems caused by touching and broken symbols, and also a formal way to encode musical syntactic rules. Because these papers describe work-in-progress, implementation details are scant. However, the successful and elegant integration of syntactic and contextual information with the low level tasks of segmentation and recognition would be a significant development in OMR research.

In the absence of any formal grammar technique, most OMR systems still apply contextual information and musical knowledge to aid recognition, albeit in an ad hoc manner.

[6] Although [Fahmy 92] precedes [Fahmy 93] in publication dates, the former is the more current work.

2.3.5 Morphology

Modayur *et al.* employ mathematical morphology for segmentation and recognition [Modayur 93] of music scores. Morphology is a technique for analyzing shape or structure. Images are manipulated by morphological operators of the form $I \alpha S$ where I is the image, α is the operator, and S is a structuring element. The structuring element is itself a small image, with one pixel designated as the origin. The structuring element S is superimposed on the input image I, and the pixels in I and S are compared according to some operator α. This is repeated for each pixel in the input image, by shifting the structuring element's origin over each position. A new image is created, representing the results of the α operator. Two fundamental morphological operators are *erosion* and *dilation* which have the effect of shrinking or expanding the foreground pixels of an image when they match the structuring element. Erosion and dilation operators are often chained together with special structuring elements to form more complex operators. Morphological operators can be designed to enhance certain shapes in an image, while removing uninteresting features. Used in this way, morphology is an effect pattern recognition technique.

In [Modayur 93] a variety of morphological operations are performed to isolate distinguishing features of music symbols. Recognition is achieved by the process of elimination. Each set of morphological operations, applied to the original image, is designed to detect a single class of symbols by removing all other symbols. Thus, after a particular set of operations, all remaining symbols are trivially classified. The technique yields impressive results, with recognition rates in excess of 95% reported.

2.3.6 Handwritten Recognition

Efforts to recognize handwritten music notation are limited. Bulis *et al.* reports on the use of projections for recognizing handwritten notes [Bulis 92], but, as previously

argued, it is doubtful such an approach will produce a robust OMR system, especially one designed for handwritten music. In [Wolman 92], the authors indicate the intention to develop an OMR system capable of recognizing handwritten notation; however, no subsequent progress reports have appeared in the literature to date. The most significant attempt at handwritten recognition is detailed in [Roach 88], which reported results that were "quite good." Only simple notes are recognized — clefs, rests, time signatures, etc. are ignored.

2.3.7 Commercial Systems

Several commercial OMR systems are currently available:

MIDISCAN The first commercial OMR package released. A consumer review of MIDISCAN can be found in [Lindstrom 94].

NoteScan A companion program to the *Nightingale* notation software available for Apple Macintosh computers.

SCORSCAN A companion program to *SCORE* for IBM PCs. *SCORSCAN* was written by Nicholas Carter, a long time OMR researcher.

Coda Technology Coda Technology, the publisher of the music notation package *Finale*, announced the intention to release on OMR system. At the time of writing, however, Coda Technology only provides an OMR service.

In Chapter 6, the developed OMR system will be compared to *MIDISCAN* and *NoteScan*.

2.4 State of the Art

Evaluating the current state of the art in optical music recognition is complicated by many circumstances:

- Researchers have yet to establish a common corpus of sheet music images. Such a corpus would facilitate objective comparisons among existing OMR systems. In [Blostein 92a] it was indicated that such a corpus was under construction by Carter. No such corpus is currently available to the OMR research community, but Carter assures, pending permission from copyright holders, that a collection of images will be available.

- It is unclear how to measure the results of recognition since the scope of existing OMR systems vary. Some systems, for example, produce MIDI output for sound reproduction, thus eliminating the necessity to recognize certain symbols. An OMR system directed at desktop publishing of music, on the other hand, is required to interpret all markings, including lyrical underlay. Blostein and Carter also note the need for a precise definition of error rate, and suggest that such a definition be multi-tiered, thus capable of accommodating the diverse scope of OMR systems [Blostein 92b].

- The absence of a standard music representation language also hinders direct comparisons.

The only attempt to quantitatively compare OMR systems is reported in [SF94]. Unfortunately, the lack of participation from invited contributors and the vague responses from those who did participate limit the usefulness of the survey. The author of the survey writes:

> Although some of our questions were intended to facilitate comparison, we learned that it is difficult, and at the present time probably unwise, to

make them [SF94].

Most researchers report recognition rates of over 90%. However, there are many reasons to suspect that the reported results are optimistic:

- Recognitions rates are conventionally calculated as the ratio of correctly identified symbols to the total number of attempted symbols, rather than the total number of actual symbols. For example, a hypothetical system may have a recognition rate of 95%, but not recognize any rest symbols. Such a system would have limited utility in practice, despite it's high reported recognition rate. When evaluating the worth of an OMR system based on recognition rates, it is important to consider the scope of recognition as well.
- Reported recognition rates are based on a limited test suite. The results reported in this thesis are not exempt. The lack of large score databases make extensive testing impractical[7].

The most realistic and meaningful performance measurements of OMR were conducted by Carter [Carter 92a]. Working under the umbrella of the Oxford University Press, Carter's system was employed for the task of producing a new edition of *Façade* by William Walton. The project involved processing 323 images. Although complete recognition rates were not reported, Carter noted that only 73% of the images resulted in passable output, and subsequent manual editing was required. Carter concluded that current OMR technology is useful, but addressed the need for a "more comprehensive and accurate recognition system."

Although all aspects of optical music recognition are candidates for continued researched, the following outstanding problems have been identified as the most prominent:

[7]By comparison, large corpora exist for printed and handwritten textual document analysis and scientific image analysis. It is not uncommon for OCR systems to be tested against a corpus with thousands of samples.

- The graphical nature of music notation results in many symbols being superimposed on other symbols. Techniques are needed that can separate touching symbols, or recognize symbols in the presence of adjoined symbols.
- Formal methods are needed to infer a semantic interpretation of the 2 dimensional arrangement of recognized symbols.
- Techniques adaptable to a variety of notational styles are needed[8].

In Chapter 7, the developed system will be evaluated on how it addresses these issues.

[8]The most significant effort in this area is reported in [Bainbridge 94, Bainbridge 95]; at the time of writing this project was in-progress. The envisioned system will be able to recognize any type of music notation. The particulars of each notation style are encoded in a description language which is interpreted by the OMR system.

Chapter 3

Pre-processing and Segmentation

This chapter is the first of three chapters that detail the implementation of the developed OMR system, as well as experiments that led to the techniques used. The focus of this chapter is on preparing a digitized image of a score for recognition. This involves identifying the staff lines, removing the staff lines, extracting text from the score, and finally isolating regions that contain musical symbols.

3.1 Staff Line Identification

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

The use of y-projections for finding staves is well documented in the literature [Bulis 92, Clarke 88, Martin 91, Randriamah 93, Kato 92, Baumann 92, Wolman 92], and described in the previous chapter. This technique, while simple, is not robust with respect to rotation. Figure 3.1 shows the music sample from Figure 2.4 rotated 5 degrees, with its y-projection. Here, the y-projection provides no clues to the

location of the staff lines. Taking projections at several different angles, and choosing the "best" projection, the one with the most distinct valleys and peaks, is one possible work around. However, this procedure can be formalized by employing the *Hough transform*, a well known technique for finding straight lines at arbitrary angles.

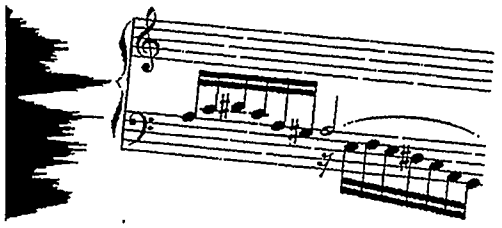


Figure 3.1: The y-projection of a score skewed by 5 degrees. Compare to Figure 2.4.

3.1.1 The Hough Transform

The Hough transform is given by:

$$H(\rho, \theta) = \int_D I(x, y)\delta(\rho - x\cos\theta - y\sin\theta)\, dx\, dy \tag{3.1}$$

Note that this is precisely the definition of a projection given earlier in Equation 2.1. Indeed, the Hough transform is equivalent to calculating a projection at every angle. The use of the term *Hough transform*, however, usually implies the use of a novel technique to calculate Equation 3.1 devised by Hough and outlined later in Figure 3.4.

The intuitive interpretation of the Hough transform is a mapping of lines in the xy-plane to points in the $\theta\rho$-plane, where ρ is the perpendicular (*i.e.* the shortest) distance between a line and the origin, and θ is the orientation of the line (see Figure 3.2). If we restrict θ to $[0, \pi)$ then the mapping is bijective; each possible line corresponds to a single point in the Hough domain (*i.e.* the $\theta\rho$-plane).

The function $\rho_i(\theta) = x_i\cos\theta + y_i\sin\theta$ produces a sinusoidal in the Hough domain, representing all lines which pass through the point (x_i, y_i) in the xy-plane. If ρ_i is

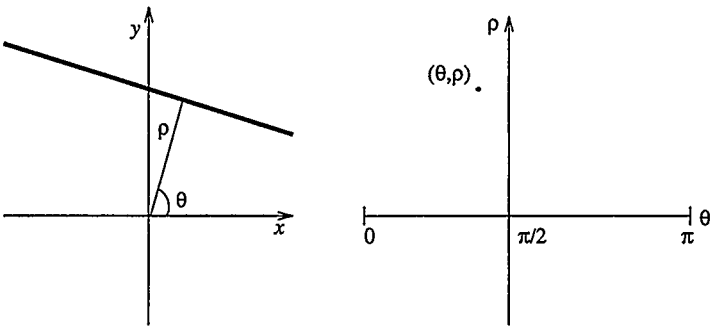


Figure 3.2: Relationship between lines in the xy-plane and points in the $\theta\rho$-plane.

evaluated for all points, then the sinusoidals corresponding to collinear points will intersect (see Figure 3.3).

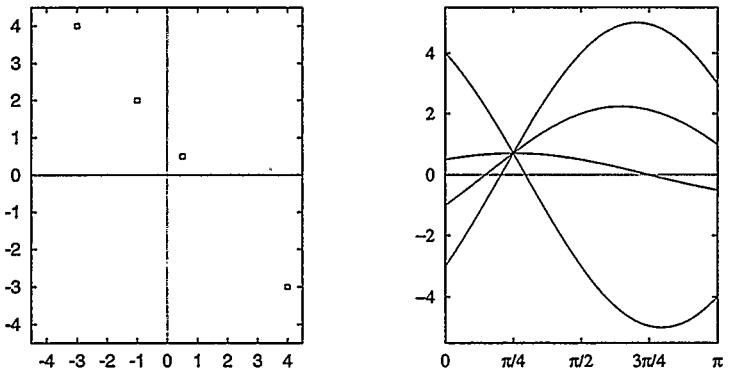


Figure 3.3: Points in the xy-plane mapped to the Hough domain. The intersections of the sinusoidals give the (θ, ρ) parameterization for the line passing through the collinear points in the xy-plane.

Rather than analytically calculating the intersections of sinusoidals, ρ_i is plotted from 0 to π and tabulated in a matrix quantization of the Hough domain. For each matrix cell, $H_{\rho\theta}$, that ρ_i intersects, the cell's value is incremented by one. If ρ_i and ρ_j intersect near (θ, ρ), both will increment $H_{\rho\theta}$ by one. Hence, $H_{\rho\theta}$ indicates how many points lie on the line parameterized by (θ, ρ). The algorithm for calculating the Hough domain is summarized in Figure 3.4.

To calculate the Hough transform H for the input image I:

set $H(\rho, \theta) \leftarrow 0$ for all (ρ, θ)
  for each x in I do
    for each y in I do
      if $I(x, y)$ is black do
        for $\theta \leftarrow 0$ to π do
          set $\rho \leftarrow x \cos\theta + y \sin\theta$
          increment $H(\rho, \theta)$ by one.

Figure 3.4: Algorithm for calculating the Hough transform.

Figure 3.5 shows an image containing a section of music and its corresponding Hough domain. The Hough domain image is a visual representation of the quantization matrix – dark regions indicate low values, and bright regions indicate high values. The five vertically aligned bright spots in the Hough domain provide the (θ, ρ) parameterization for each of the five staff lines. The bottom most staff line is superimposed by a long beam, hence the relatively larger spot in the Hough domain.

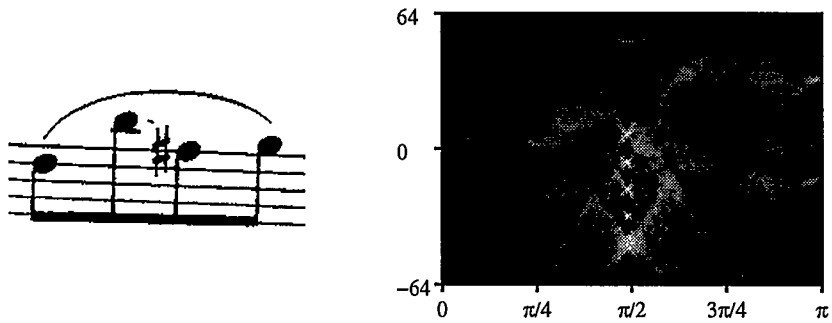


Figure 3.5: (a) A section of a music score (from [Mozart 90], measure 67). (b) The Hough domain of (a).

The Hough transform is a well known technique for finding straight lines, but is

often criticized for its excessive time and space requirements. Indeed, if the Hough domain is quantized by a $m \times n$ matrix and there are b black pixels in the image, then the time requirement is $\Theta(bn)$ and the space requirement is $\Theta(mn)$. Assuming 10% of all pixels are black, a typical page of music scanned at 300 DPI will have $\sim$1 million black pixels. Suppose $n = 180$ (giving 1° precision), then $n \times 10^6 = 180 \times 10^6 = 180$ million evaluations of $\rho(\theta)$. On current hardware, this calculation can exceed several hours; consequently, a direct and naïve implementation of the Hough transform for staff line detection is impractical.

Two significant optimizations of the Hough transform have been developed. Adiv [Adiv 83], Illingworth and Kittler [Illingwort 87, Illingwort 88], and others describe adaptive techniques that are based on an iterative course-to-fine analysis of the Hough domain. The basic idea is to examine the Hough domain for an image at a course resolution, find regions where there is a cluster of high values, and zoom into those regions by reapplying the Hough transformed at a higher resolution. Only a small, fixed-size quantization matrix is required because at higher resolutions, a smaller area of the Hough domain is being considered. Another optimization, described by Kiryati *et al.* [Kiryati 91] and many others, is to transform only a random subset of image points to the Hough domain; this technique is commonly known as the *probabilistic Hough transform.*

3.1.2 Hough-Based Algorithms for Staff Line Identification

Two different Hough based techniques for staff lines detection were developed. Both methods were ultimately rejected for an ad hoc approach that is both faster and more reliable. Nonetheless, the two Hough based methods are worth discussing.

The first Hough method was based on the adaptive technique described in [Illingwort 87]. Rather than transforming all black image pixels, however, a

pre-processing step was implemented to find a subset of black pixels that were potentially part of a staff line[1]. The image was scanned vertically at equally spaced horizontal intervals to find short runlengths (< 5 pixels for a 300 DPI image) of black pixels; the centers of these runlengths provided the y-coordinate for the points to be transformed to the Hough space.

Using the points collected from the pre-processing step, the adaptive Hough transform was applied to find long, approximately horizontal lines. Initial experiments showed promising results, but the technique was never fully explored because of a shortcoming discovered early on: if the staff lines are sufficiently bowed[2], then they cannot be parameterized as a single straight line. Consequently, when the Hough transform is applied to a curved line, a "fuzzy" peak is observed in the Hough domain, rather than a narrow, sharp peak. This behaviour is illustrated in Figure 3.6. The staff in Figure 3.6, at first glance, appears relatively straight; however, such subtle curvature is sufficient to produce unreliable results via the Hough transform.

Experiments with the Hough algorithm described above, herein called the α-Hough algorithm for the purpose of nomenclature, demonstrated a need for a line finding algorithm that would be robust with respect to curvature. The second staff finding Hough algorithm, β-Hough, was designed specifically to handle curved staff lines robustly and efficiently. β-Hough is an adaptive and recursive algorithm that works by applying the Hough transform to various regions along the staff, dividing the staff into approximately linear sections. The algorithm is shown in pseudo-code in Figure 3.7[3]. The technique requires as input a small region, R_{init}, that possibly contains a staff; R_{init} can be found by using y-projections.

[1]This does not qualify as a probabilistic Hough transform because the chosen pixels are not chosen randomly.

[2]A common cause of curved staff lines is when the original score is taken from a book whose binding prevents the entire page from being flush against the photocopier/scanner glass.

[3]For simplicity, the data structures used to represent the staff lines are omitted from the pseudo-code.

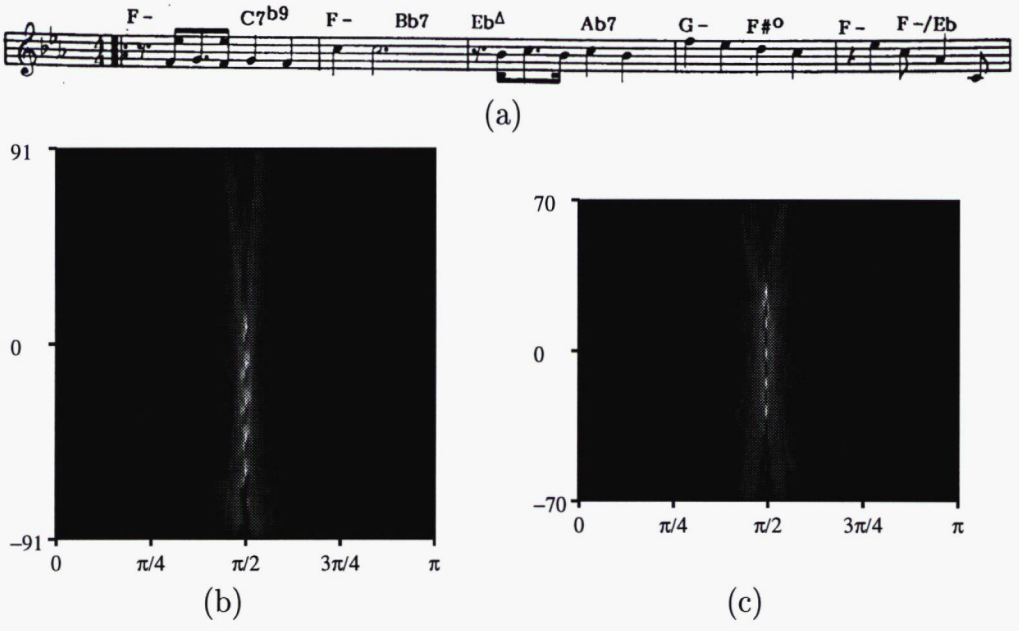


Figure 3.6: (a) A music staff with bowed staff lines. (b) The Hough domain for (a). (c) The Hough domain for a staff with straight lines (not shown). Notice that in (c) the peaks in the Hough domain are very narrow in the horizontal direction, indicating that the angle of the corresponding lines in the xy-plane are known within a small tolerance. By contrast, in (b) the peaks are "fuzzy" since the corresponding lines are actually curves.

The β-Hough algorithm successfully found all staff lines in all experiments conducted. Its major drawback is that, even for images with straight staff lines where no staff subdivision is necessary, the running time was approximately 60–90 seconds per page. Another drawback is the algorithm's complexity. The pseudo-code in Figure 3.7, for brevity, ignores several complications that can arise. Most significant is that for an arbitrary staff region, there is no guarantee the Hough transform can find the staff lines (because of the density of music symbol) or differentiate other line segments (beams, long slurs, and voltas, for example) from the staff lines.

Had a simpler and faster algorithm not been developed, this Hough-based algorithm would have been used.

3.1.3 Staff Line Detection by Vertical Runlengths

The final algorithm developed for detecting staff lines operates by examining vertical columns of pixels and recording all *staff samples* – five equally spaced, equally sized runlengths of black pixels[4]. The entire image is scanned from left to right, every nth[5] column yielding a new set of staff samples (see Figure 3.8). Congruent staff samples are collected into *staff groups*; two samples are congruent if they have similar spacing, runlength thickness, and vertical position.

[4]This assumes a conventional five line staff, in use since the 13th century. Chant notation, however, uses a four line staff, and a grand-staff has ten lines. These formats, although uncommon, are still candidates for optical recognition.

[5]Here, n is dependent on the resolution of the image. n is chosen to be approximately 2.5mm.

FindStaff(R_{init}) ≡
  if *Lines*(R_{init}) = ∅ then
    return false /* no staff found */
  else
    return FindLeftSide(R_{init}) and FindRightSide(R_{init})

FindLeftSide(R_{init}) ≡
  R_{left} = region left of R_{init}
  return *FindStaff*(R_{left}, R_{init})

FindRightSide(R_{init}) ≡
  R_{right} = region right of R_{init}
  return *FindStaff*(R_{init}, R_{right})

FindStaff(R_{left}, R_{right}) ≡
  if *Lines*(R_{left}) $\cong^a$ *Lines*(R_{right}) then
    return true
  else
    R_{mid} = region between R_{left} and R_{right}
    return *FindStaff*(R_{left}, R_{mid}) and *FindStaff*(R_{mid}, R_{right})

Lines(R) ≡
  return lines found by applying the Hough transform to R

[a] ≅ meaning *almost collinear.*

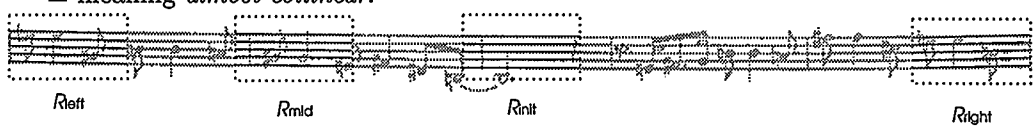


Figure 3.7: The β-Hough staff finding algorithm.

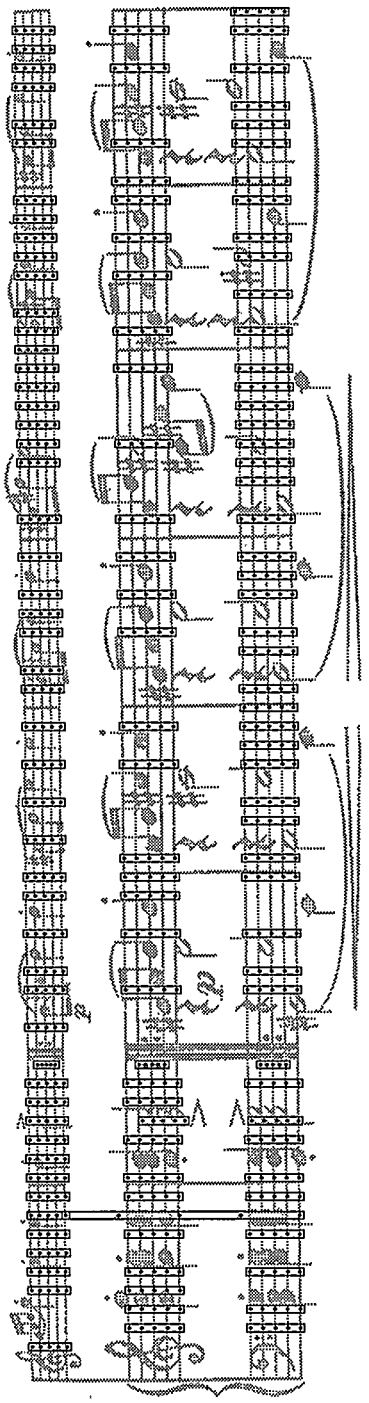


Figure 3.8: Staff samples used for detecting staff lines (music sample from [Dvořák 61]).

Each staff group is a candidate staff. It must be determined which samples in each group form a staff, and which samples are invalid. In Figure 3.8, there are several false staff samples that must be identified. The median of all the angles between staff samples within a group provides a reasonable estimate of the angle of rotation. A modified Hough transform can now be applied to find the most dominate line formed by all the staff samples. The conventional Hough space for lines is a two parameter space, namely θ and ρ. However, the previous calculation of rotation constrains θ, leaving only ρ as a free parameter. Thus, the modified Hough transform is a one dimensional transform that can be used to determine the ρ parameter. Once ρ is determined, staff samples that are not collinear with the dominant line can be removed. To allow for curved staff lines, rather than removing the invalid staff samples, samples that are connected to valid samples are retained. Connectedness can be determined by tracing a line between adjacent samples; if no white pixels are encountered, then the samples are connected[6]. Finally, once all valid samples are determined, redundant samples are removed.

Figure 3.9 shows the result of the staff finding algorithm for a test score. Figure 3.10 illustrates the handling of curved staff lines. When a staff line cannot be represented by a single line, the staff line is piecewise approximated be dividing the staff line until each division is linear, within a small tolerance. The algorithm also works well on images that are rotated. Figure 3.11 shows the original test image rotated by 10 degrees and the identified staff lines. The running time of this algorithm is approximately 15 seconds per page, a four-fold improvement over the β-Hough based method.

[6]This is a simplified approach. In practice, a count of black pixels on the traced line is made. Two samples are connected if the number of black pixels exceeds, say 80%, of the total number of pixels along the line.

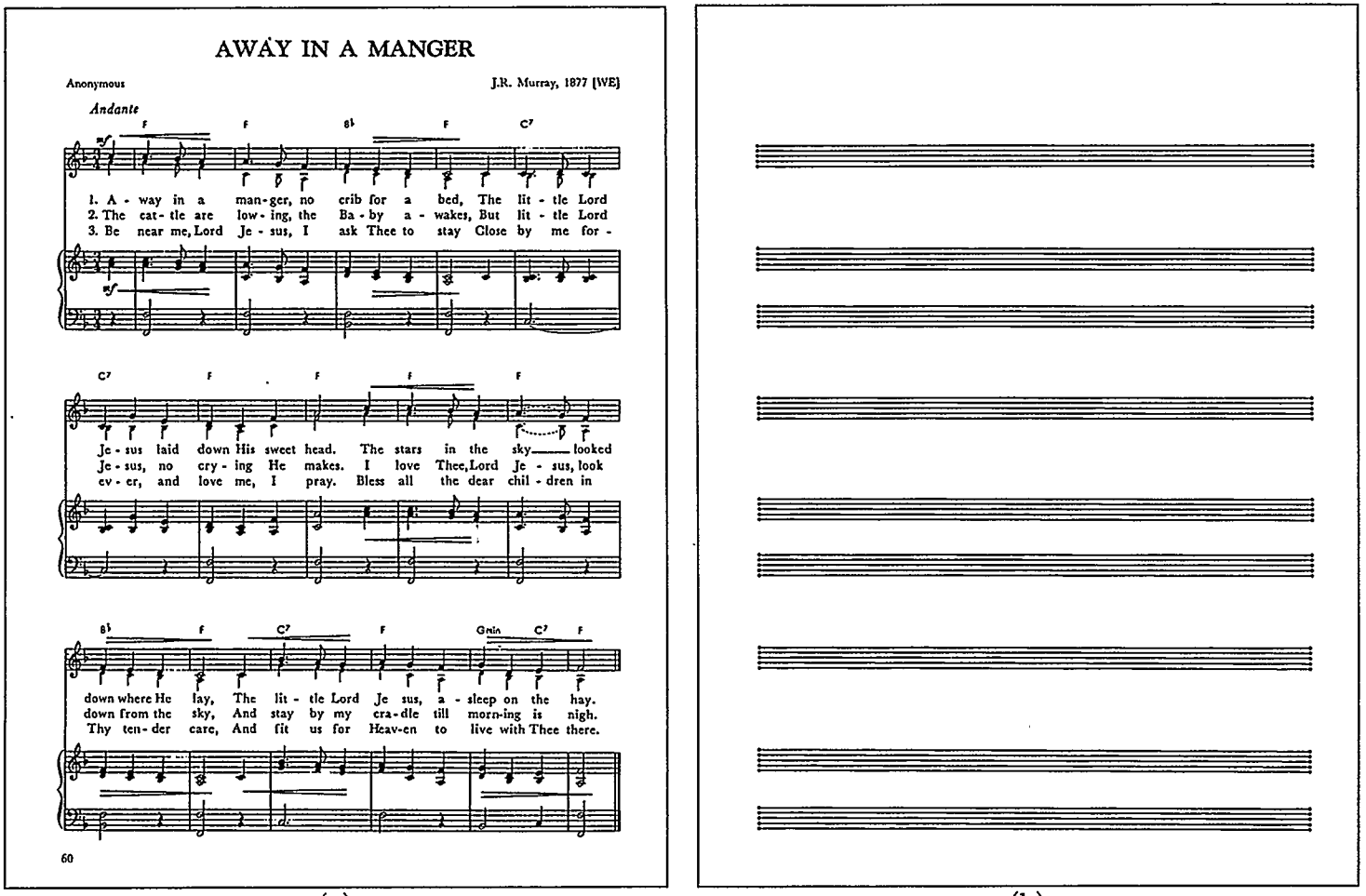


Figure 3.9: (a) A test score (from [Anonymous 80]). (b) Recognized staff lines.

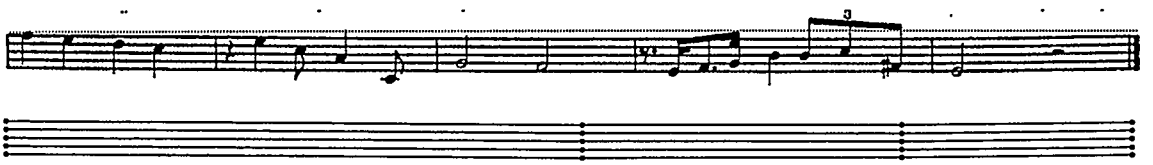
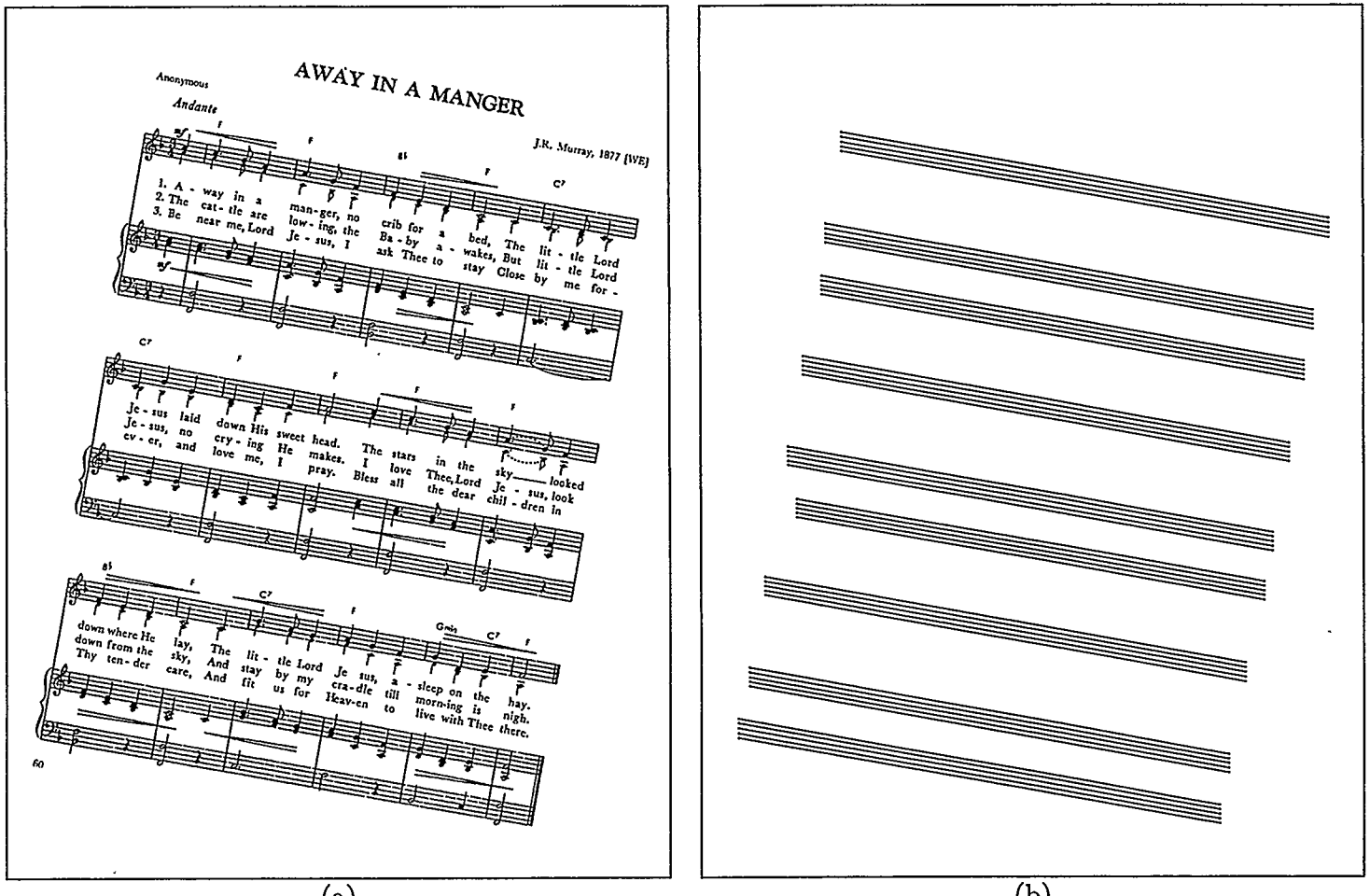


Figure 3.10: A staff with curved lines. When recognized, the curve is approximated by linear segments.



(a) (b)

Figure 3.11: (a) The original test score rotated 10 degree. (b) Recognized staff lines. The mediocre determination of the staff ends is a consequence of the rotation, but is not considered a serious flaw. It is common practice in image document analysis to correct the orientation of a page prior to processing so that all subsequent processing can assume a properly orientated page. Hence, the staff finding algorithm can be applied as pre-processing step to determine the skew, in which case the positions of the staff ends are irrelevant.

3.2 Removing Staff Lines

Because the staff lines intersect almost all other symbols, their presence generally interferes with subsequent symbol recognition. An inevitable consequence of removing staff lines is that some symbols will become fragmented. Although complex techniques have been developed to minimize this undesirable side effect (see Section 2.3.3 on page 19), these efforts have produced only marginally better results than the simplest of alternative techniques. The approach adopted here removes the staff lines, except where their thickness exceeds some threshold, presumably caused by the presence of another symbol. The technique works especially well in the absence of whole notes, half notes, and slurs, as shown in Figure 3.12. However, in the presence of the aforementioned symbols, considerable fragmentation occurs, as shown in Figure 3.13.

Prior to removing the staff lines, the original image is saved so that it is available in subsequent stages. Problems with note fragmentation are completely avoided by working with the original image when detecting note heads.

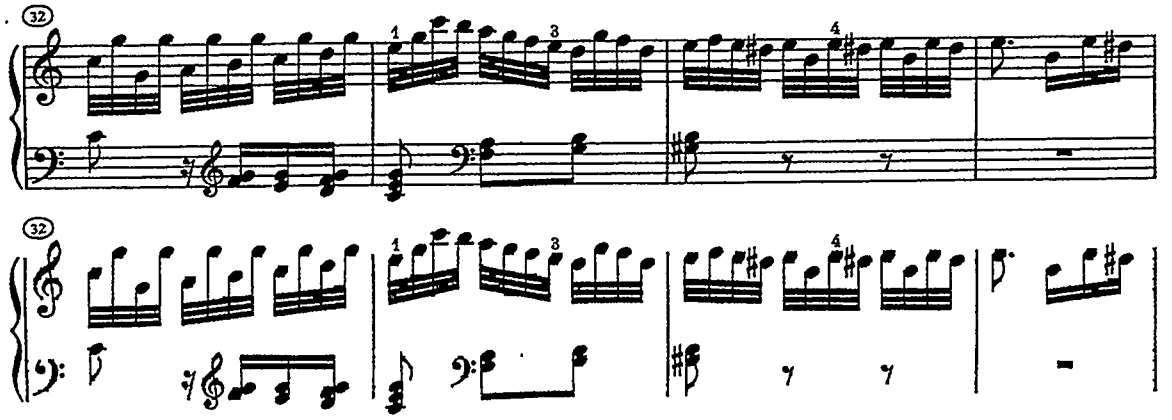


Figure 3.12: Top-to-bottom: Original image (from [vB50]); removal of staff lines using vertical run-lengths to test for the presence of other symbols.

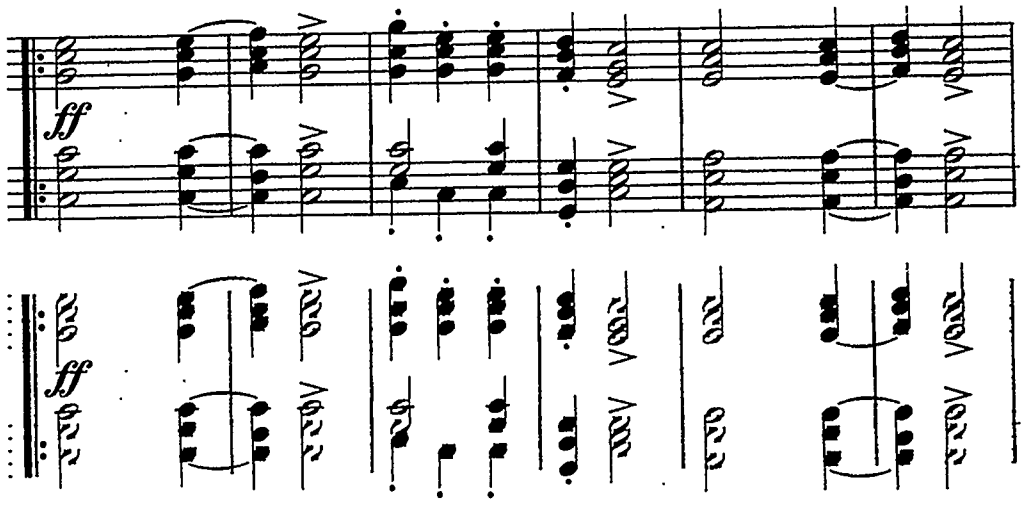


Figure 3.13: Top-to-bottom: Original image (from [Dvořák 61]); after removing staff lines many half notes become fragmented.

3.3 Connected Component Analysis

Before proceeding with subsequent recognition tasks some pre-processing is necessary to find the *connected components*. Two black pixels are connected if they are adjacent. More specifically, two black pixels are 4-connected if they share a common side, and 8-connected if they shared a common corner or are 4-connected. A region R is connected if between any two pixels in R, (x, y) and (x', y'), there exists a sequence of n pixels (x_i, y_i) such that $(x_1, y_1) = (x, y)$, $(x_n, y_n) = (x', y')$, and (x_j, y_j) and (x_{j+1}, y_{j+1}) are connected for $j = 1, \ldots, n-1$. Finally, a connected component is a maximal connected region.

3.3.1 Compressed Line Adjacency Graphs

Line adjacency graphs, briefly introduced previously in Section 2.3.2 on page 18, provide an efficient way to represent connected components. A LAG representation of an image can be constructed by scanning the image along either vertical or horizontal

strips. Each runlength of black pixels becomes a node, and an edge between two nodes is added if the associated runlengths overlap. The connected components can then be determined by any graph traversal algorithm[7], such as depth first search.

LAGs can be transformed into a more compact form. The number of edges attached to a node is called the *degree*. Since LAGs have a geometric interpretation, it makes sense to distinguish edges that are on either side of a node. For a LAG constructed via horizontal runlengths, a node can have edges to nodes above and nodes below, relative to itself. Let (a, b) denote the degree of a LAG node having above degree a and below degree b. A compressed LAG[8] is formed by amalgamating two connected nodes, A and B, if A is above B, the degree of A is $(a, 1)$ and the degree of B is $(1, b)$ (a and b are unconstrained). The additional constraints that the runlengths for node A and B be of similar lengths, and the pixels represented by the resulting amalgamated node can be approximated by a straight line segment are employed (see Figure 3.14). Figure 3.15 shows a compressed LAG equivalent of the LAG presented earlier for an isolated whole note. The construction of a compressed LAG can be done on-the-fly, without prior construction of a LAG.

The results of connected component analysis using LAGs is illustrated in Figure 3.16 for the score in Figure 3.9.

A useful interpretation of a compressed LAG is a line segment approximation of the original image. If each node stores, in addition to the end points, the thickness of the line it represents, the original image can be approximately reconstructed from the LAG. Consequently, the compressed LAG can be used instead of the original image for recognition. This will prove especially convenient when recognizing line segments,

[7]Incidently, the term *connected component* is also used in graph theory. The connected components (in the graph theory sense) of a LAG are precisely the connected components (in the computer graphic sense) being sought. This relation only holds if the graph is undirected.

[8]The abbreviation c-LAG has been used in [Mori 95] for "compressed-LAG". However, C-LAG is used to abbreviate "contour-LAG" in [Pavlidis 82], so abbreviations are abstained from in this case.

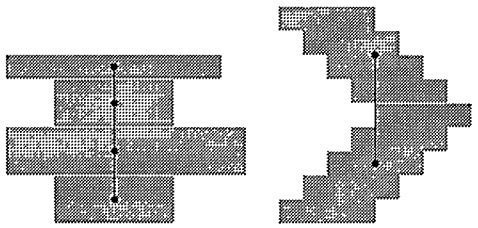


Figure 3.14: Two compressed LAGs. (a) Different sized runlengths are not amalgamated. (b) Two nodes that have identical runlength sizes cannot be amalgamated if they form nonlinear segments in the image. A single node cannot represent a curved component, or a component with a corner.

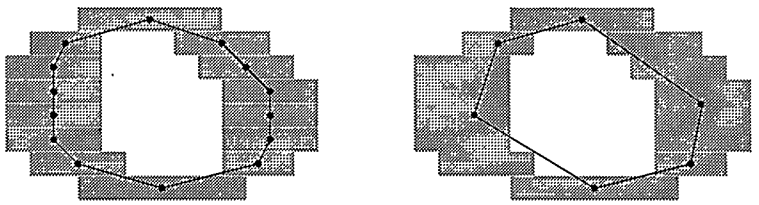


Figure 3.15: A LAG and a compressed LAG representing an isolated whole note.

as discussed in Chapter 4.

3.4 Segmentation

Aside from music notation, a score often contains textual information as well: the title, the composer, and, of course, lyrics. Many scores, especially older scores, also contain decorative artwork. Segmentation is the process of dividing a score image into regions of music, text, and artwork.

The segmentation of artwork is not investigated in this thesis, and the remainder of this section will focus on text segmentation. Even though no attempt is made at recognizing textual information, separating textual markings from non-textual markings in a score is critical so subsequent processing can operate exclusively on music symbols. Text segmentation removes a significant source of noise in a music score,

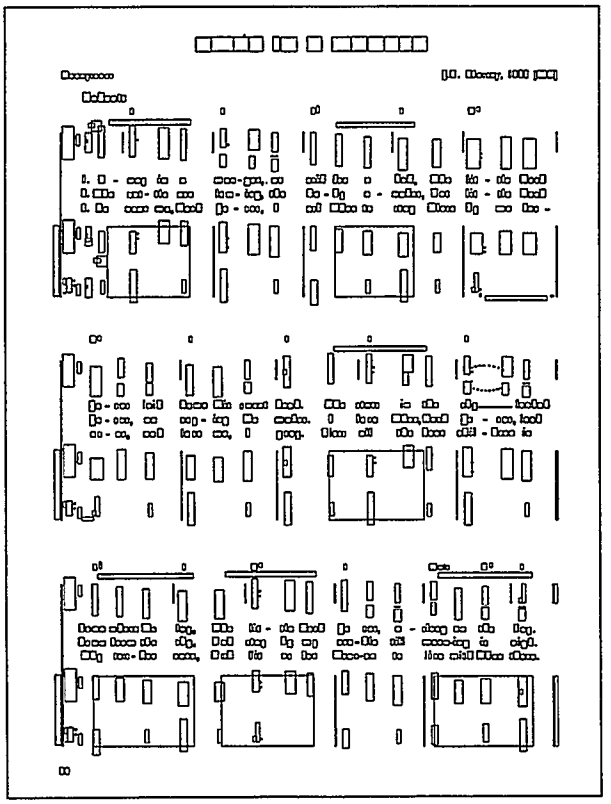


Figure 3.16: Bounding boxes for the connected components of Figure 3.9(a) (after staff lines have been removed).

thus improving both speed and accuracy.

Segmentation methods can be classified as either top-down or bottom-up. The classic top-down segmentation technique is known as recursive X-Y cut. Horizontal and vertical projection profiles are taken, and the image is split, horizontally or vertically, at the widest valley in either of the projection profiles. The algorithm is applied recursively to each subregion formed by the split until the projection profiles are uniform and exhibit no wide valleys. Recursive X-Y cut assumes the document is predominantly textual and can be recursively subdivided into halves; these assumptions severely restrict its application for general segmentation. More advanced techniques use a declarative language or grammar that describes the document layout. These

techniques are appropriate for well structured documents with fixed layouts, such a journals.

Bottom-up segmentation techniques work iteratively by building high level descriptions from low level evidence. Typically, pixels are grouped into characters, which are grouped into words, which are grouped into lines, and so on. Because bottom-up techniques make fewer *a priori* assumptions about document layout, they are typically more robust than top-down techniques.

In practice, most segmentation algorithms are a hybrid of top-down and bottom-up segmentation, employing different techniques at different stages of segmentation[9].

The algorithm implemented to extract textual markings from a score is a simplification of the algorithm developed in [Fletcher 95]. This hybrid approach uses the Hough transform to group collinear marks, and subsequently uses bottom-up techniques to group words and phrases. This technique was chosen over others for its effectiveness in extracting small strings in predominantly graphical documents. The following sections detail the implementation of this algorithm.

A more detailed overview of segmentation techniques with additional references can be found in [Witten 94].

3.4.1 Text Segmentation

Once the connected components for a score have been determined, text segmentation can proceed. Before applying the text segmentation algorithm proper, several preprocessing steps can be applied to filter out components that are obviously non-textual. The location of the staves serves as an immediate aid to segmentation:

[9] Although "top-down" and "bottom-up" are popular labels for segmentation techniques in the literature, their usefulness is questionable. The run-length smoothing algorithm developed by Wong [Wong 82] has been labeled both top-down (in [Jain 95], for example) and bottom-up (in [Witten 94], for example). The simplicity of run-length smoothing forbids it from being a hybrid approach. The author favours the "bottom-up" label for run-length smoothing.

symbols that intersect a staff are almost certainly musical symbols. Components whose bounding boxes have extreme aspect ratios or a height greater than some threshold can also be filtered.

The text segmentation algorithm described in [Fletcher 95] uses the Hough transform to find collinear components. It is assumed that collinear components with regular spacing form words. In a music score, it is expected that all text is printed parallel to the direction of the staff lines. Since the staff lines have already been identified, the orientation of the staff lines can again be used to reduce the Hough space to a one dimensional parameter space[10]. Aside from this modification, the algorithm described in [Fletcher 95] was implemented almost to the letter. The Hough transform is used to group components that are collinear; the centroids of the components are used in the transformation. Regions in the Hough domain with a high concentration of components are isolated. These components are then sorted into reading order to examine the inter-character and inter-word spacing. Characters are grouped into words, and then words are grouped into phrases. For a word to be positively identified as text, it must have at least 3 characters, or be part of a phrase containing words with more than 3 characters. The algorithm is summarized in Figure 3.17.

Figure 3.18 shows the results of text segmentation for the the score image in Figure 3.9. Note that several one and two letter words were not identified as text, even though they occur within a sentence. This is a consequence of the wide inter-word spacing prevalent in lyrical underlay, since the lyrics must be temporally aligned with the notes, producing larger than normal inter-word spaces. Increasing the inter-word threshold would erroneously group adjacent non-text symbols together to form words. One solution can be implemented as a post-processing step. After textual symbols have been identified, reapply the 1D-Hough transform to those symbols only.

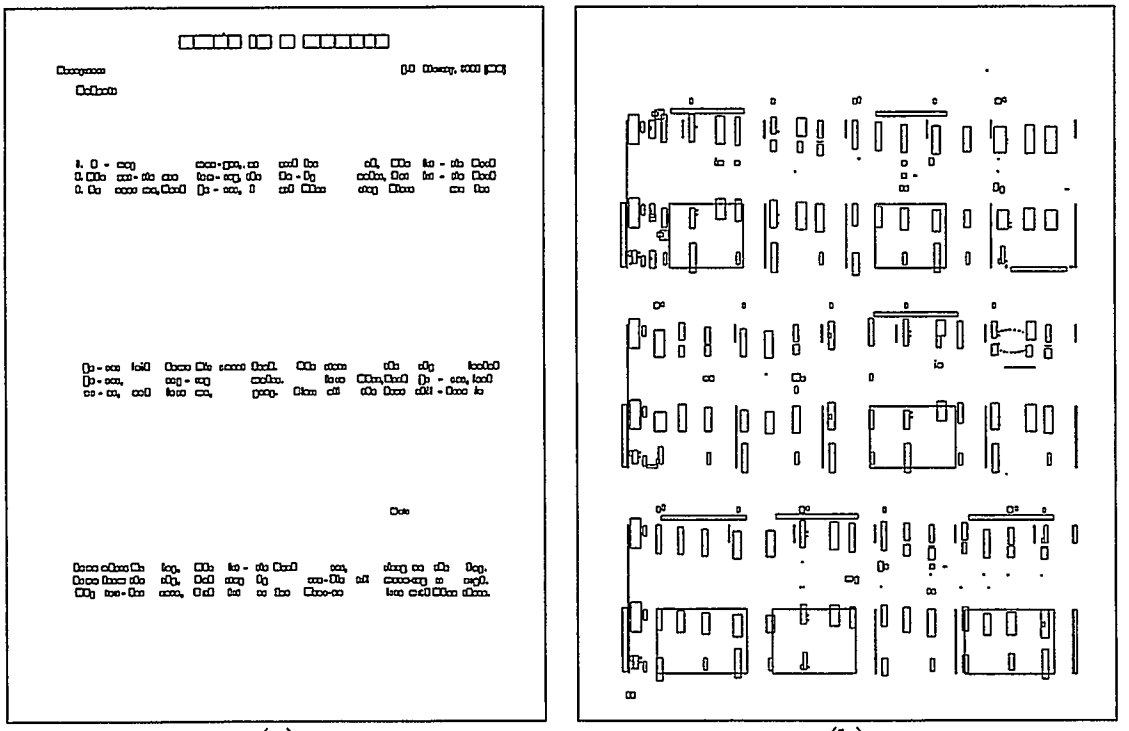
[10]Recall that a similar one dimensional Hough transform was used in the staff finding algorithm itself (see Section 3.1.3 on page 34.

To identify connected components containing textual symbols:

- Calculate the average height, H, of all components.
- Set the Hough domain ρ resolution such that each cell maps onto $R = 0.2 \times H$ pixels. Each Hough cells represents a height several times smaller than the typical character height.
- Calculate the modified 1D Hough transform.
- For each cell ρ having a count greater than 3 do
  - Calculate the local average height, H_{avg} of all components in cells $\rho - 5$, $\rho - 4, \ldots, \rho, \ldots, \rho + 4, \rho + 5$.
  - Calculate a local clustering factor, $f = H_{\text{avg}}/R$.
  - Cluster the components in the cells $\rho - f$, $\rho - f + 1, \ldots, \rho + f - 1$, $\rho + f$.
    / This clustering is performed to account for different font sizes in the image, and so characters with ascenders and descenders are not omitted. */*
  - Perform string segmentation on the cluster.
  - Remove components that were identified as characters from the Hough domain.

Figure 3.17: Algorithm for text segmentation.

Then apply the 1D-Hough transform to the other symbols, but without updating the Hough space. Instead, use the Hough space to decide if the symbol is likely part of a sentence. For example, suppose symbol α transforms to cell m with value v. If $v > t$ (where t is some threshold, say 20), then label α as a text symbol. Figure 3.19 shows the results of implementing this post-processing step.



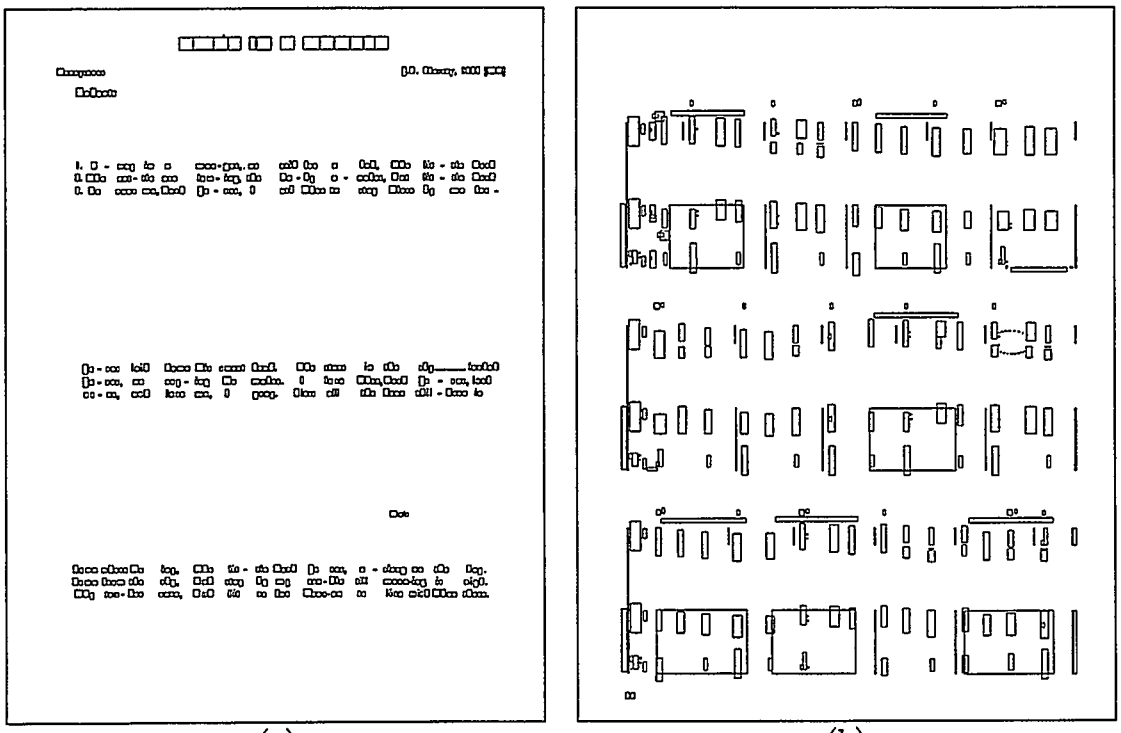
(a) (b)

Figure 3.18: Results from text segmentation for Figure 3.9(a). (a) Bounding boxes for text components. (b) Bounding boxes for non-text components.

There are several deficiencies with the implemented text segmentation algorithm:

- It only works for good quality, printed text. Handwritten, or even script type, would not be located since characters in the same word would be connected and thus could not be isolated.

- The algorithm is naïve in assuming *any* collinear sequence of symbols form



(a) (b)

Figure 3.19: Text segmentation results after post-processing for Figure 3.9(a). (a) Bounding boxes for text components. (b) Bounding boxes for non-text components.

text. Dashed lines, for example, would be identified as a word. This could be circumvented by applying a post-processing step to filter out words with repeated characters with certain characteristics.

- Many expression marks, even when valid graphic symbols exist for them, are conveyed by written text. For example, crescendos and decrescendos are often substituted by *cresc.* and *decresc.* or *dim.* respectively. Many written instructions have no graphic equivalents. In Figure 3.18 and Figure 3.19 the expression mark *Andante* was extracted, even though it is part of the music notation[11]. Such symbols are significant if the goal of the OMR system is automated audio rendition of the score.

[11] *Andante* means the piece should be played at a moderately slow tempo.

Without actually recognizing the text, these problems are inevitable with any text segmentation algorithm.

Chapter 4

Symbol Recognition

This chapter discusses methods for classifying the music symbols. A minimalist approach to recognition is adopted. Only the simplest symbols, from which all other symbols can be constructed, are recognized; these include lines, note heads, and character-like symbols. The next chapter will describe how more complex symbols are recognized by analyzing the geometric layout of the primitive symbols.

4.1 Lines and Curves

The primary reason for adopting the compressed LAG approach for connected component analysis was to aid the recognition of line segments. A consequence of compressing a LAG is that every node represents a line segment. Some nodes, of course, represent degenerate lines because the node was not compressible. Line segments are easily located by searching the compressed LAG for nodes representing non-degenerate lines; optionally, additional constraints on line thickness, length, and orientation can restrict what lines are found. After extracting line segments from the LAG, neighbouring collinear segments are merged. The same technique can be used to find curves

by relaxing the collinearity constraint in the final merging step.

Because compressed LAGs are constructed by traversing the image along consecutive scan lines, the LAG nodes always represent lines that are in the orthogonal direction to the scan direction. For example, a compressed LAG constructed from horizontal scans will have nodes representing non-horizontal lines segments. Therefore, such a LAG cannot be used for extracting horizontal lines; a perfectly horizontal line would be interpreted as a very thick, very short vertical line. To find all lines, regardless of orientation, it is necessary to construct both horizontal and vertical scan based compressed LAGs.

Figure 4.1 and Figure 4.2 show the results of extracting vertical and horizontal lines respectively.

4.2 Character Recognition

Many components of music notation are character symbols – small letter-like symbols. These include, for example, clefs, rests, accidentals, and note heads. This section will examine the recognition of all these symbols, except note heads, which will be discussed in the next section.

Little effort was expended investigating the problem of character recognition since it is already so well researched. Indeed, literature on the topic abounds, and time constraints excluded a comprehensive investigation. Rather a simple solution to character recognition was sought that would be adequate and could be implemented in a short time.

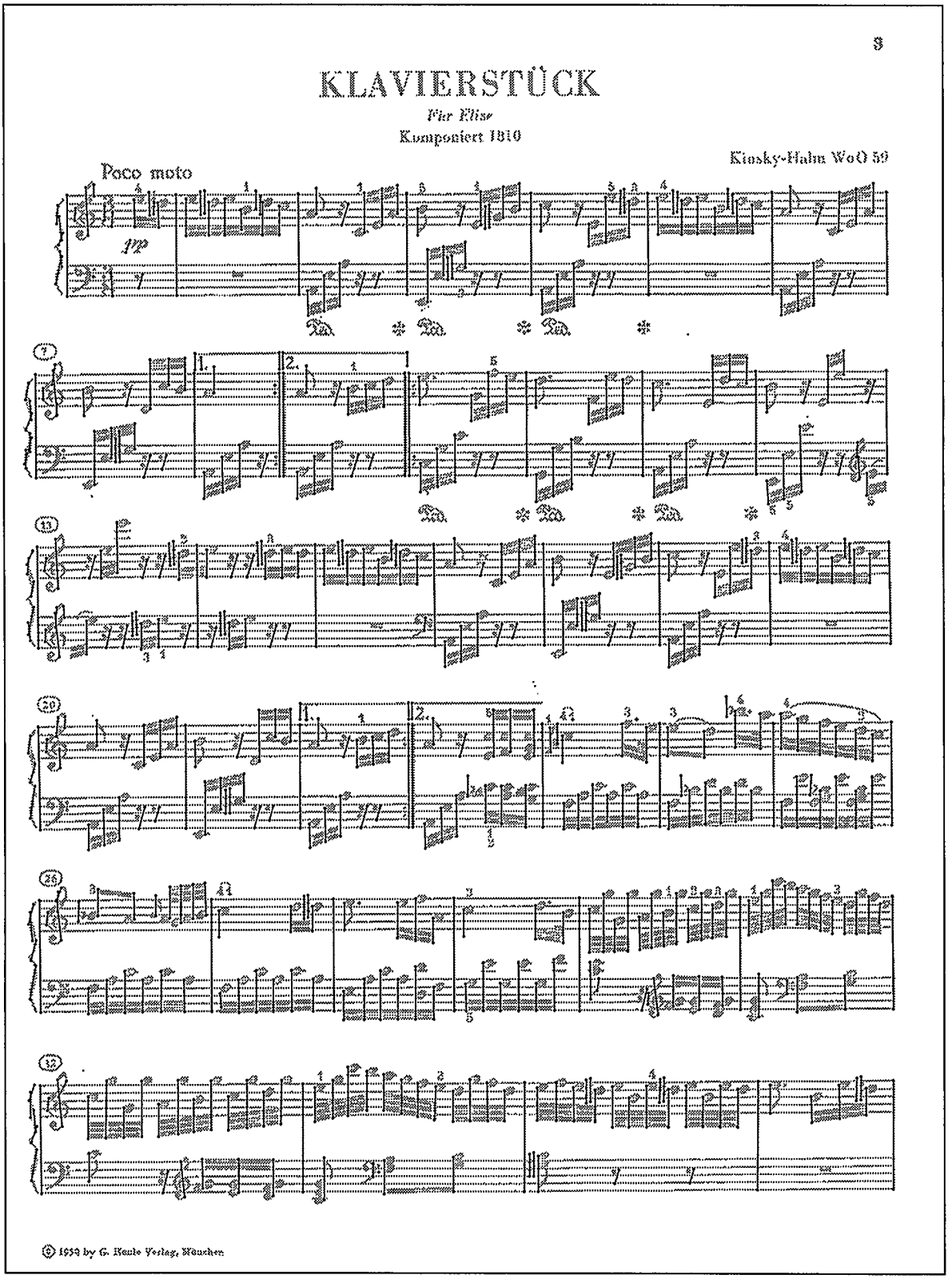


Figure 4.1: Vertical line detection. Original image shown in gray (also shown in Figure B.11); extracted vertical lines shown in black.

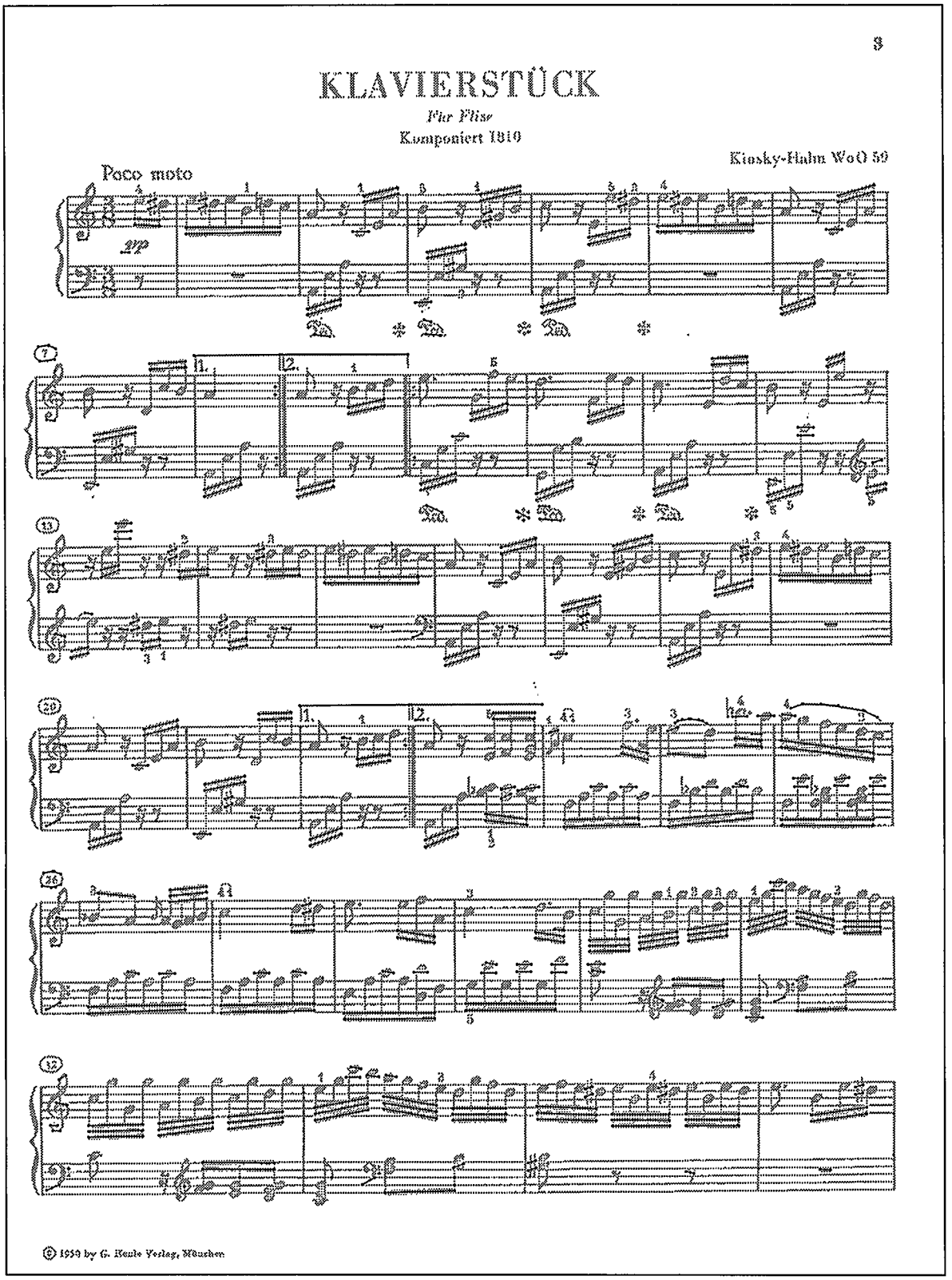


Figure 4.2: Horizontal line detection. Original image shown in gray (also shown in Figure B.11); extracted horizontal lines shown in black.

4.2.1 Character Profiles

The technique adopted is based on *character profiles*, functions that measure the perpendicular distance of the character's outer contour from a reference axis. The left and bottom edge of the character's bounding box serve as the reference axis, yielding left/right profiles (L and R) and bottom/top profiles (B and T) respectively. Width and height profiles (H and W) are obtained from $R - L$ and $T - B$. Figure 4.3 illustrates all the profiles taken from a natural character.

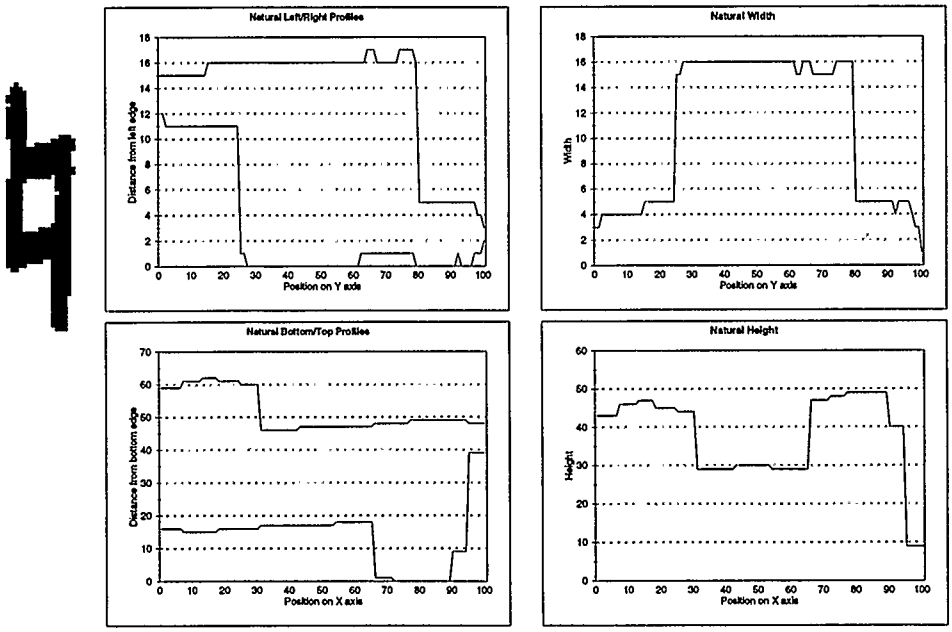


Figure 4.3: Sample profiles for a natural.

A rule based classifier, using measurements obtained from the profiles, is used to recognize symbols. Several useful measurements include:

- $\text{maxima}_{[a,b]}P$ = the set of maxima peaks found in profile P in the range $[a, b]$[1], denoted $\{l_1, \ldots, l_n\}$, where $l_i \epsilon [a, b]$ gives the location of the ith maxima[2].

[1]Profiles are normalized to the interval $[0, 100]$.

[2]Special care must be taken so that single pixels caused by noise do not constitute a maxima.

- $\text{minima}_{[a,b]}P$ = the set of minima peaks found in profile P.
- $\max_{[a,b]} P =$, the location of the absolute maximum of profile P in the interval $[a, b]$.
- $\min_{[a,b]} P =$, the location of the absolute minimum of profile P in the interval $[a, b]$.

For a symbol to be classified as a natural, for example, the following conditions must be satisfied:

- $\text{minima}_{[5,95]}H = \{a\}$, $a\epsilon[30, 70]$.
- $\text{maxima}_{[5,95]}W = \{a\}$, $a\epsilon[30, 70]$.
- $\text{minima}_{[5,95]}L = \emptyset$.
- $\text{maxima}_{[5,95]}R = \emptyset$.

Similar rule sets are used to identify treble clefs, all rests[3], sharps, and flats. The only apparent omission here is bass clefs, which are actually recognized by locating the two dots left of the main clef structure.

Although other recognition techniques have not been implemented here, better recognition rates can be obtained by applying a multi-algorithm paradigm. If several independent recognition techniques are available, each technique can participate in a voting scheme to "elect" the best classification for a symbol. If a majority vote is not obtained, then the symbol is rejected. Such approaches have been implemented in [Kimura 91] and [Parker 94b], where character profiles were used. One of the other techniques use in [Parker 94b] is *vector templates*; Parker has successfully used vector templates to recognize musical characters [Parker 95].

[3]Currently only 1/1 through 1/16 rests are recognized. The relative rarity of smaller rests prevented sufficient samples to be collected from the corpus used.

4.3 Note Head Recognition

Because note heads are usually attached to stems, and in the case of chords, attached to other notes heads, it is not possible to isolate note heads by connected component analysis. This precludes using most stock algorithms for character recognition since they assume the character is already isolated. Rather, a searching algorithm is required that can locate a character symbol that exists as a subimage of the input image.

Two approaches to note head recognition were examined: the Hough transform for circles, and template matching. Initial experiments with the Hough transform produced marginal, but unsatisfactory results, leading to the adoption of template matching. Two simple refinements to the standard template matching technique were developed, one improving accuracy, the other run-time performance.

4.3.1 The General Hough Transform

The discussion of the Hough transform in the previous chapter was simplified. Though first used to identify straight lines, the Hough transform technique has been generalized, and can be used to identify any parameterized path.

The general Hough transform is given by:

$$H(p_0, \ldots, p_n) = \int_D I(x,y)\delta(P(p_0, \ldots, p_n))\, dx\, dy \tag{4.1}$$

where P is a path defined as the set of points $\{(x,y) : P(p_0, \ldots, p_n) = 0\}$. For straight lines, we had used $P = \rho - x\cos\theta - y\sin\theta$[4].

The effectiveness of the Hough transform for finding note heads was investigated. Note heads are generally elliptical, but can be approximated by a circle, or a family

[4]The standard equation for a line, $y = mx + b$, can also be used, but has the disadvantage that vertical lines cannot be represented.

of concentric circles (see Figure 4.4). Circles can be parameterized by $r^2 - (x-a)^2 - (y-b)^2 = 0$, where (a, b) is the circle's center, and r its radius. Hence, the Hough transform for circles is:

$$H(a, b, r) = \int_D I(x, y)\delta(r^2 - (x-a)^2 - (y-b)^2)\, dx\, dy \tag{4.2}$$

Note that the Hough domain is now a 3 dimensional space, parameterized by r, a and b. However, since note heads have a known size[5], r can be fixed to approximately half of the staff line spacing, thus reducing the Hough space to 2 dimensions, and simplifying the calculations. To allow for a degree of tolerance, and to approximate all note head types by circles, the calculation is repeated for several radii, centered around the ideal radius.

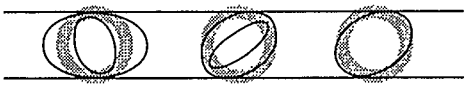


Figure 4.4: Note head shapes. The gray circle represents the circles that are sought by applying the Hough transform for circles.

The circular Hough transform was applied to three small sample images, each containing samples of one type of note head. In each case, the transformation was applied to both the original images and the images after staff lines have been removed. The removal of staff lines is both beneficial and detrimental for finding note heads. In one case, a staff line may bisect a whole or half note making the note more similar to a filled note; removing the staff line corrects this. In the other case, a staff line may be tangent to a whole or half note; removing the staff lines usually fragments the note (see Figure 3.13). The samples also contain other symbols that might be mistaken for a note head; all accidentals, for example, contain an approximately circular subset of points.

[5]This is a naïve assumption, since a score may have grace notes. Also, see Figure 3.9(a) – the vocal staff contains note heads of two distinct sizes.

Figure 4.5 shows the results from the above experiment. Bright spots in the Hough space correspond to the origin of circles found in the input image. Visual inspection of the Hough domain provides sufficient evidence that the circular Hough transform is inadequate for finding note heads. Although the Hough transform would likely work for finding filled note heads, the lack of distinct bright spots for half and whole notes suggest the technique is not robust and would likely produce many false matches for these notes. Better results may be obtained by searching for ellipses, since all notes are generally elliptical; again, the Hough transform can be applied. Nonetheless, the Hough approach was abandoned in favour of template matching.

4.3.2 Template Matching

Template matching is a recognition technique that measures the similarity between an unknown pattern and several known template patterns. Typically, an unknown pattern, P, is matched against all templates, T_i $i = 1 \ldots n$ (where n is the number of classes), and then P is classified to be a member of class j if $S(P, T_j) > S(P, T_i)$ $\forall i \neq j$ (where S is some similarity metric).

Several similarity metrics have been devised for comparing binary images [Parker 94a, Schalkoff 92, Witten 94]. The metric adopted [Parker 94a] is presented below, and illustrated in Figure 4.6:

$$m = \frac{M^+ - M^-}{M^+ + M^-}$$

where

$m \epsilon [-1, 1]$ is the normalized match index.

M^+ is the number of matching pixels ($T(i,j) = P(i,j) = 1$).

M^- is the number of mismatching pixels ($T(i,j) \neq P(i,j)$).

This metric was chosen for several reasons:

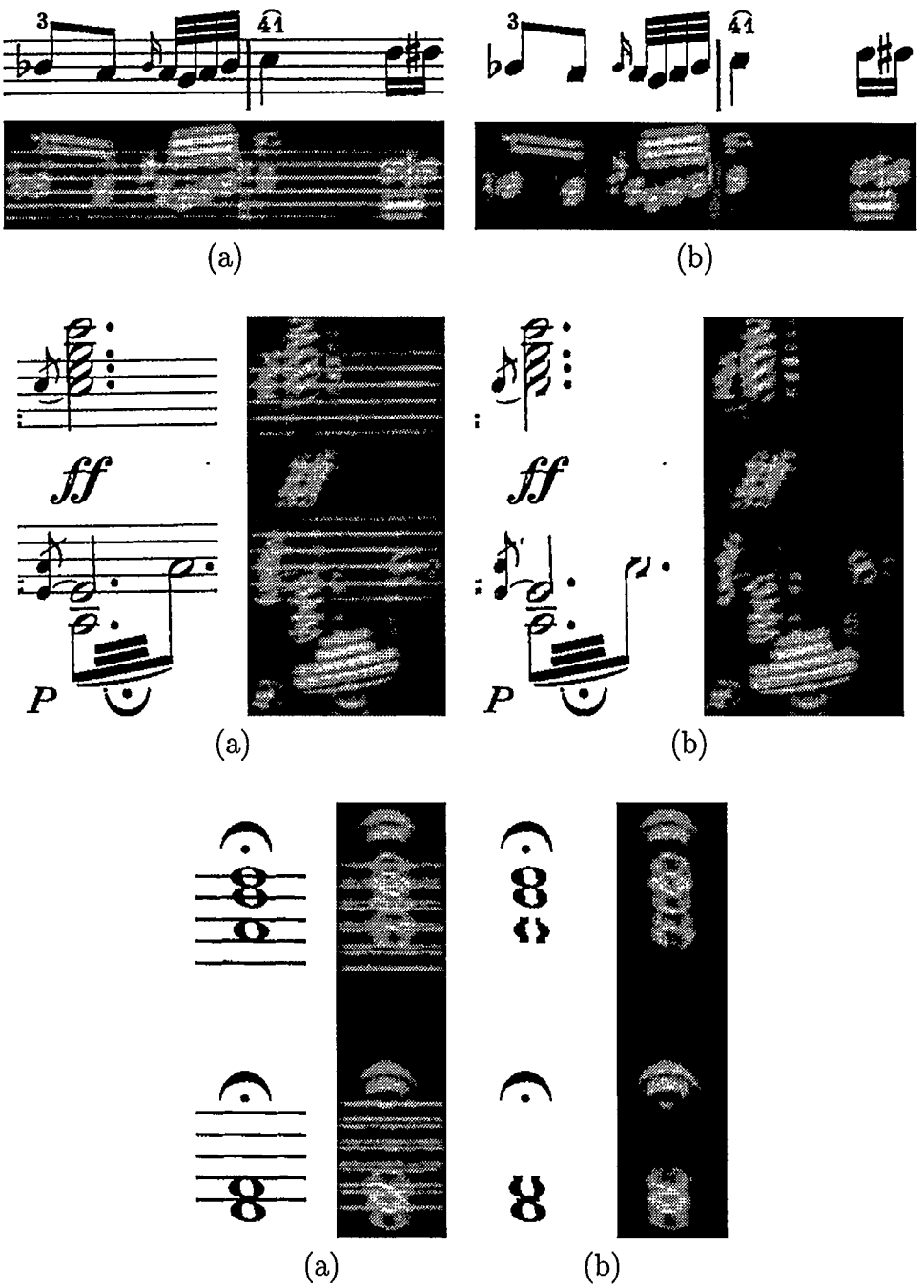
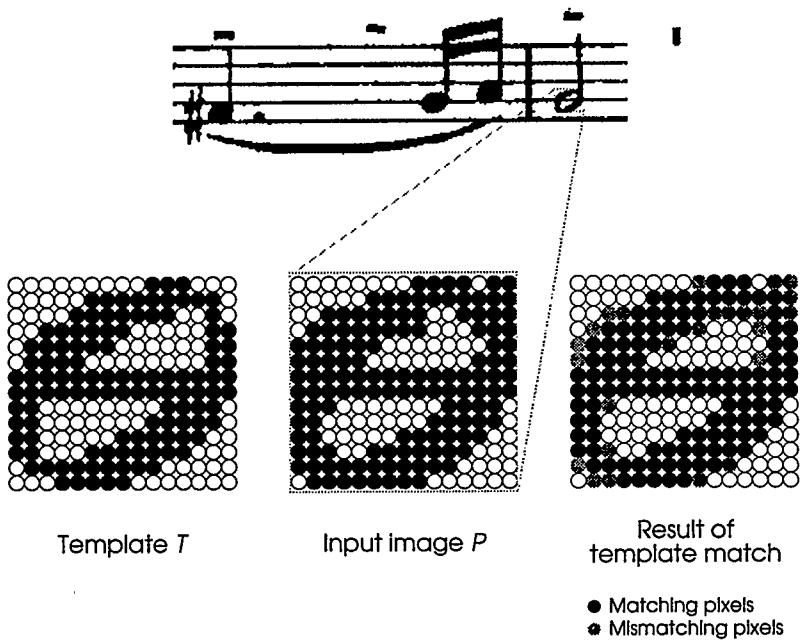


Figure 4.5: Using the circular Hough transform to find note heads. Top-to-bottom: filled notes (music sample from [vB50]), half notes (music sample from [Dvořák 61]), and whole notes (music sample from [Pachelbel 77]). In each sample, the Hough space is the result applying the circular Hough space for circles with radius in $[s/2 - 2, s/2 + 2]$, where s is the staff spacing. (a) With staff lines present. (b) With staff lines removed.



$$m = \frac{M^+ - M^-}{M^+ + M^-} = \frac{113 - 25}{113 + 25} = 0.64$$

Figure 4.6: Calculating the similarity metric for template matching.

- Because note heads cannot be resolved into isolated characters, the template must be shifted over all possible positions in the input image, whilst computing the similarity metric at each position. Some techniques, such as compression based template matching [Witten 94], require that the input image be an isolated symbol.

- The metric is normalized such that $m \epsilon [-1, 1]$ (with $m = 1$ indicating a perfect match). Many other techniques have no convenient normalization and consequently the match index is not independent of the template size.

Note that in many instances, template matching is mathematically equivalent to computing the Hough transform. However, template matching has at least one significant advantage: templates can represent arbitrarily complex shapes that cannot be

practically represented by parametric equations[6], required for the Hough transform. For example, the Hough transform cannot directly distinguish between a circle and a filled circle.

The previous experiment with circular Hough transforms was repeated using template matching. To minimize the error caused by the staff lines intersecting the note heads, template symbols that include the staff line are considered. Consequently, each note head type requires two templates corresponding to the two possible positions of the note relative to its surrounding the staff lines[7]. In total, three templates are tested: (1) the note head in isolation, and (2,3) the note head with staff lines.

Figure 4.7, Figure 4.8, and Figure 4.9 show the results of performing template matching on the filled, half, and whole note head test images respectively. For each figure, the test image (a), and the results for the three templates considered are shown (b,c,d). For each template, the first three false matches, and all positive matches with indices[8] higher than the smallest false match are labelled. For (c) and (d), two distinct positive matches are possible: a true positive match, and a *semi-positive* match, obtained when the template matches a note head and the note head is in a different position than the template, relative to the staff lines.

There are several factors to considered when evaluating the effectiveness of template matching based on the results shown in Figure 4.7, Figure 4.8, and Figure 4.9:

- It is essential that the number of positive matches equals the actual number of

[6] One drawback with templates, however, is that a distinct template must be constructed for each possible font size. Alternatively, the templates can be created dynamically using a vector font that describes mathematically the outline contour for the font.

[7] Two templates are actually not sufficient, as illustrated below:

An additional bottom ledger line is needed to give the effect of being between staff lines. Therefore, to handle every case, four templates would be needed. In practice, using only two templates does marginally decrease recognition rates as discussed in Section 6.2.

[8] When is it clear from context, the abbreviated *index* will be used for *match index*.

Note Type	Template Confidence Index		
	(b)	(c)	(d)
Filled Notes (Figure 4.7)	0.223	0.280	0.388
Half Notes (Figure 4.8)	0.209	0.227	0.222
Whole Notes (Figure 4.9)	0.200	0.031	0.193

Table 4.1: Template confidence indices for Figure 4.7, Figure 4.8, and Figure 4.9.

note heads present in the test image. If this is not the case, then note heads exist that have smaller indices than a false match. Consequently, it is impossible to reliably select the note heads without also selecting false matches.

- If the above condition is satisfied, then the effectiveness of the template matching procedure can be quantitatively measured. Let m_f^i, $i = 1, 2, 3$ be the indices of the three false matches selected. Let m_p^j, $j = 1 \ldots n$, where n equals the number of positive matches[9]. Then define the *template confidence index* as:

$$C = \min(m_p^1, \ldots, m_p^n) - \max(m_f^1, m_f^2, m_f^3)$$

C is simply the difference between the smallest positive match index, and the largest false match index. The larger C is, the better the template procedure. Table 4.1 summarizes the template confidence indices for all the template experiments.

According to Table 4.1, the best recognition rates for note heads would be achieved by employing the following methods:

[9]Excluding semi-positive matches.

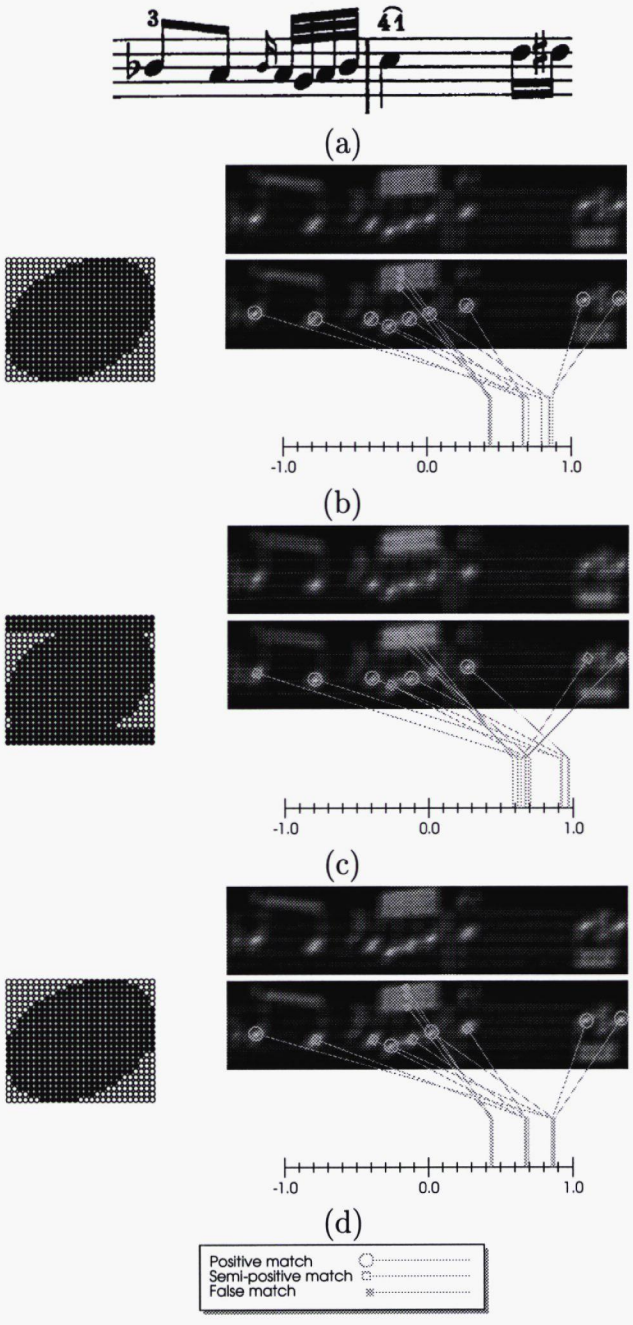


Figure 4.7: Filled note head template matching.

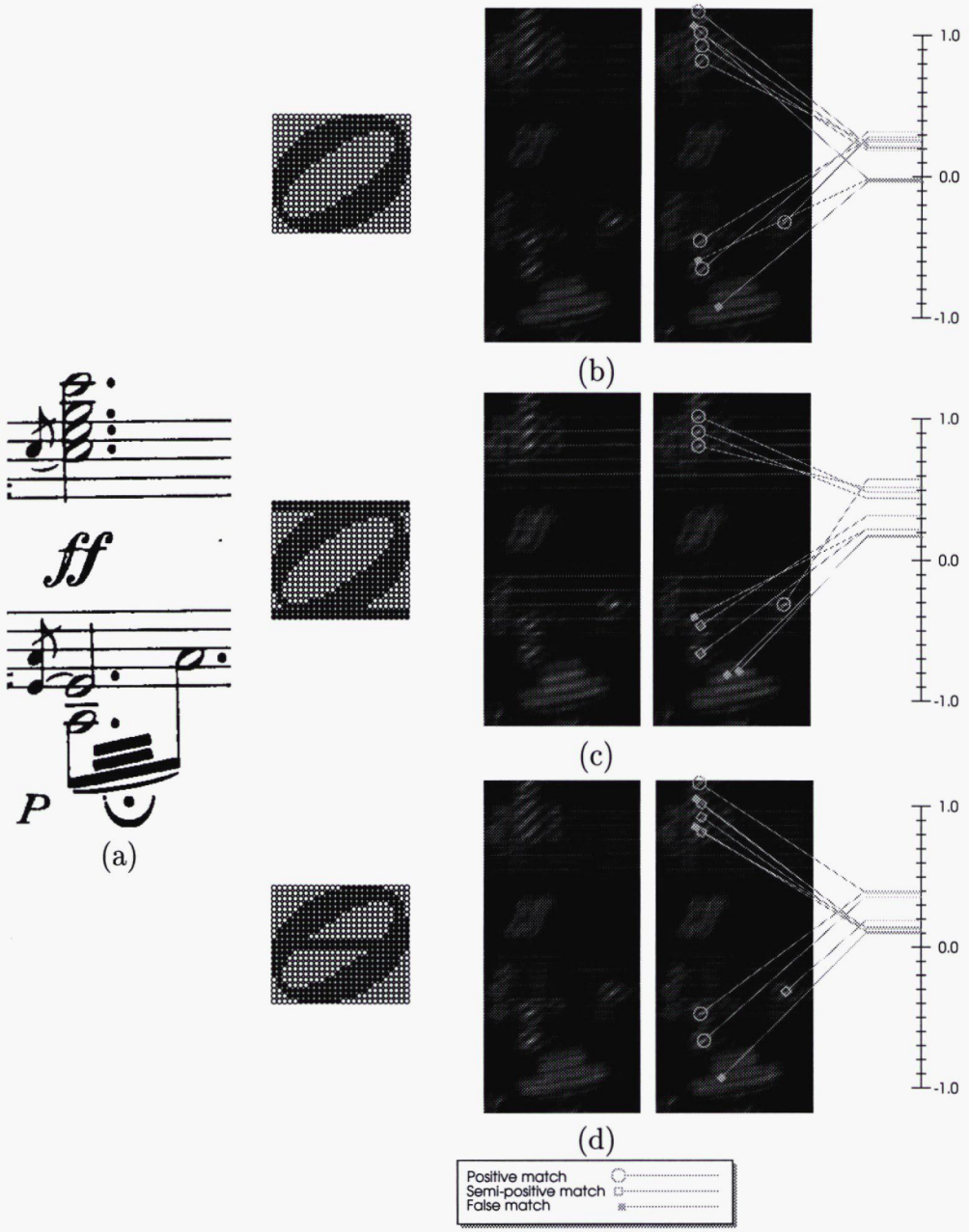


Figure 4.8: Half note template matching.

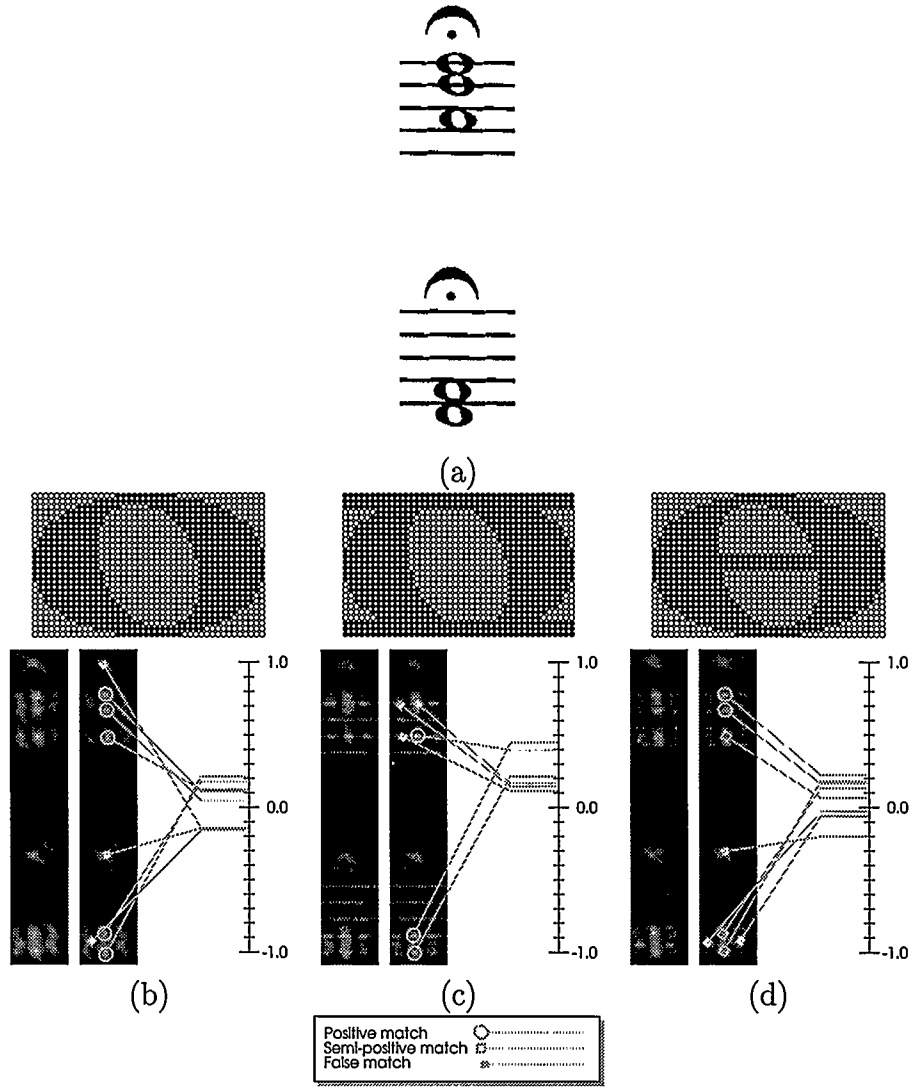


Figure 4.9: Whole note template matching.

Note Type	Template Type	Template Confidence Index [a]
Filled notes	With staff lines	0.334
Half notes	With staff lines	0.225
Whole notes	Without staff lines	0.200

[a]When the template type is *With staff lines*, two templates are used; the template confidence index is the average for the two templates.

The results for matching half and whole note heads are significantly worse than the result for filled note heads. There are many reasons for this discrepancy:

- When matching a template with an object of the same shape, pixel mismatches are randomly distributed about the boundary of the template. Half and whole notes have almost twice the boundary length of a quarter note, hence a larger error.

- Font shape and size for half and whole notes can vary significantly. Shape variations increase the number of mismatching pixels along the boundary of a template. Staff spacing is used to select appropriate font size. In some cases, especially with half notes, the note head height is greater than the staff spacing, resulting in less than ideal template selection[10].

- Half and whole notes have fewer foreground pixels than filled note heads. The ratio of boundary pixels to total pixels is a good indicator for the amount of error expected. This ratio is high for whole and half notes (0.30 and 0.42 respectively), and low for filled note heads (0.15).

[10]Indeed, better results could have been posted for the half note and whole template experiments, if the templates were chosen manually. The template sizes used were chosen automatically by the software, based on the staff spacing and staff line thickness, and were slightly smaller than the ideal templates.

4.3.3 Improving Template Matching Accuracy

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, template matching performance degrades significantly, as demonstrated by the previous results.

The proposed solution is to distinguish *exterior* and *interior* background pixels in the template (see Figure 4.10). The interior background pixels are matched against background pixels in the test image; exterior background pixels in the template are ignored as before. The precise metric is:

$$m = \begin{cases} \frac{M_O^+ - M_O^- + M_I^+}{N} & \text{if } M_O^+ - M_O^- > 0 \\ \frac{M_O^+ - M_O^- - M_I^+}{N} & \text{if } M_O^+ - M_O^- \leq 0 \end{cases} \tag{4.3}$$

where

$m \epsilon [-1, 1]$ is the normalized match index.

M_O^+ is the number of object pixels that matched.

M_O^- is the number of object pixels that didn't match.

M_I^+ is the number of interior background pixels that matched.

N equals $M_O^+ + M_O^- + M_I^+$.

The justification for this metric is an intuitive one. If more object pixels match than mismatch ($M_O^+ - M_O^- > 0$), than the matching interior pixels have a likelihood of actually matching interior pixels of some unknown object, and make a positive contribution. On the other hand, if more object pixels mismatch than match ($M_O^+ - M_O^- \leq 0$), then the matching interior pixels are likely matching background pixels, thus contributing negatively.

Figure 4.11 and Figure 4.12 show the results using Equation 4.3. A visual comparison between Figure 4.8 and Figure 4.11, and Figure 4.9 and Figure 4.12 reveals

markedly better results by using the modified template matching technique in most cases. Table 4.2 summarizes the results. Now, the best recognition rates for note heads would be achieved by employing the following methods:

Note Type	Template Type	Template Confidence Index
Filled notes	With staff lines	0.334
Half notes	Without staff lines	0.515
Whole notes	Without staff lines	0.562

Note Type	Template Confidence Index		
	(b)	(c)	(d)
Filled Notes (Figure 4.7)	0.223	0.280	0.388
Half Notes (Figure 4.8)	0.209	0.227	0.222
Half Notes (Figure 4.11)	**0.515**	**0.178**	**0.228**
Whole Notes (Figure 4.9)	0.200	0.031	0.193
Whole Notes (Figure 4.12)	**0.562**	NA[a]	**0.585**

[a] A false match index exceeded the smallest positive match index.

Table 4.2: Template confidence indices for all template matching experiments. Bold entries report results obtained from the modified template matching technique (Equation 4.3); other entries are duplicated from Table 4.1.

4.3.4 Improving Template Matching Run-time Performance

The most significant drawback of template matching is its poor run-time performance, especially when the templates are large. Indeed, template matching consumes more than 90% of the total running time of the complete OMR system, as illustrated in Figure 4.13[11]. Clearly, template matching is an obvious target for optimizations.

[11] The times indicated in Figure 4.13 are not consistent with those reported in Section 6.1.3. The improved times in Section 6.1.3 reflect the performance when all debugging code has been removed and compiler optimizations switched on.

An effective technique for optimizing template matching is as follows. The original template is partitioned into multiple templates such that each partition samples the original (see Figure 4.14). When testing for a match, each partition participates in one pass over the test image. After each pass, an interim match index is computed. The interim match index is an approximation of the real match index obtained by comparing every pixel. A sufficiently low interim match index[12] aborts the template match, thus avoiding the cost of comparing every pixel in both the template and the image. The effect on overall performance is dramatic, as shown in Figure 4.15.

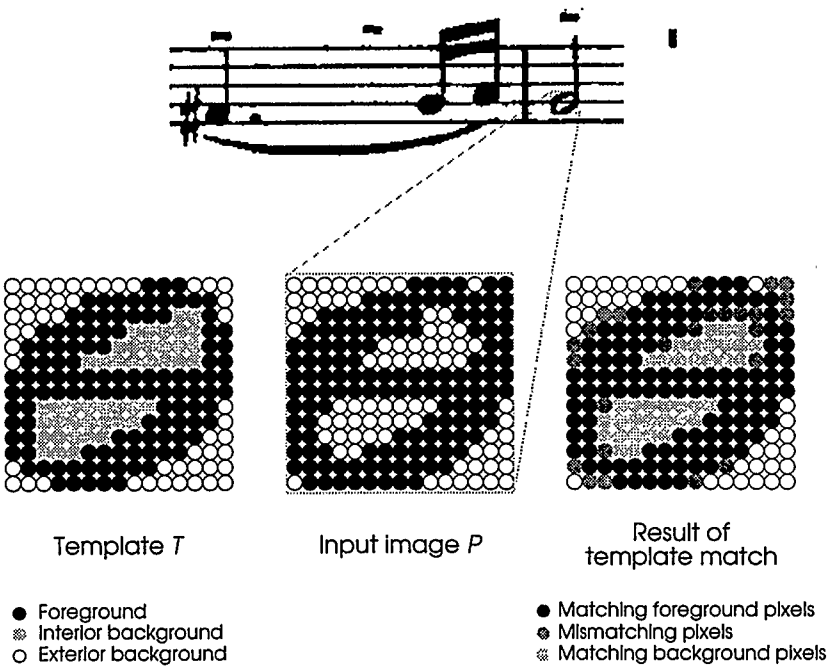
This paradigm has several possible implementations. The original template can be partitioned into any number of partial templates. In Figure 4.14, the original template is partitioned into four partial templates, such that each pass will sample a quarter of the pixels. Other partition schemes are, of course, possible. Furthermore, given any particular partition scheme, not every partial template will necessarily be used. For large templates, comparing every pixel may be excessive, and comparing only half the pixels may be sufficient to confirm a match. This is precisely the scheme used in the current implementation for finding note heads; the partitions in Figure 4.14 are used, but a maximum of two passes are made. Other applications would likely warrant a different configuration.

Another possible variation that may provide an additional, albeit marginal, performance gain is to abort the template match when the interim match index is sufficiently high. This simply mirrors the behaviour when a very low interim match index is obtained.

This template matching optimization was developed independently. Subsequent to its implementation, it was discovered that the a similar technique had already been developed. *Two-stage template matching* uses subtemplates (*i.e.* partial templates)

[12]If the threshold match index is T, a *sufficiently low interim match index* should be $T - \epsilon$, $\epsilon > 0$. In the experiments conduct, $\epsilon = 0.15$ was used.

and performs a two-pass application of the templates [Vanderbrug 77]. Other optimizations developed include sequential methods and hierarchical methods [Li 85].



$$\begin{aligned} M_O^+ &= 113 \\ M_O^- &= 25 \\ M_I^+ &= 37 \end{aligned}$$

Since $M_O^+ > M_O^-$,

$$m = \frac{M_O^+ - M_O^- + M_I^+}{N} = \frac{113 - 25 + 37}{113 + 25 + 37} = 0.71$$

Figure 4.10: Modified template similarity; background pixels in the interior of the template symbol are now significant.

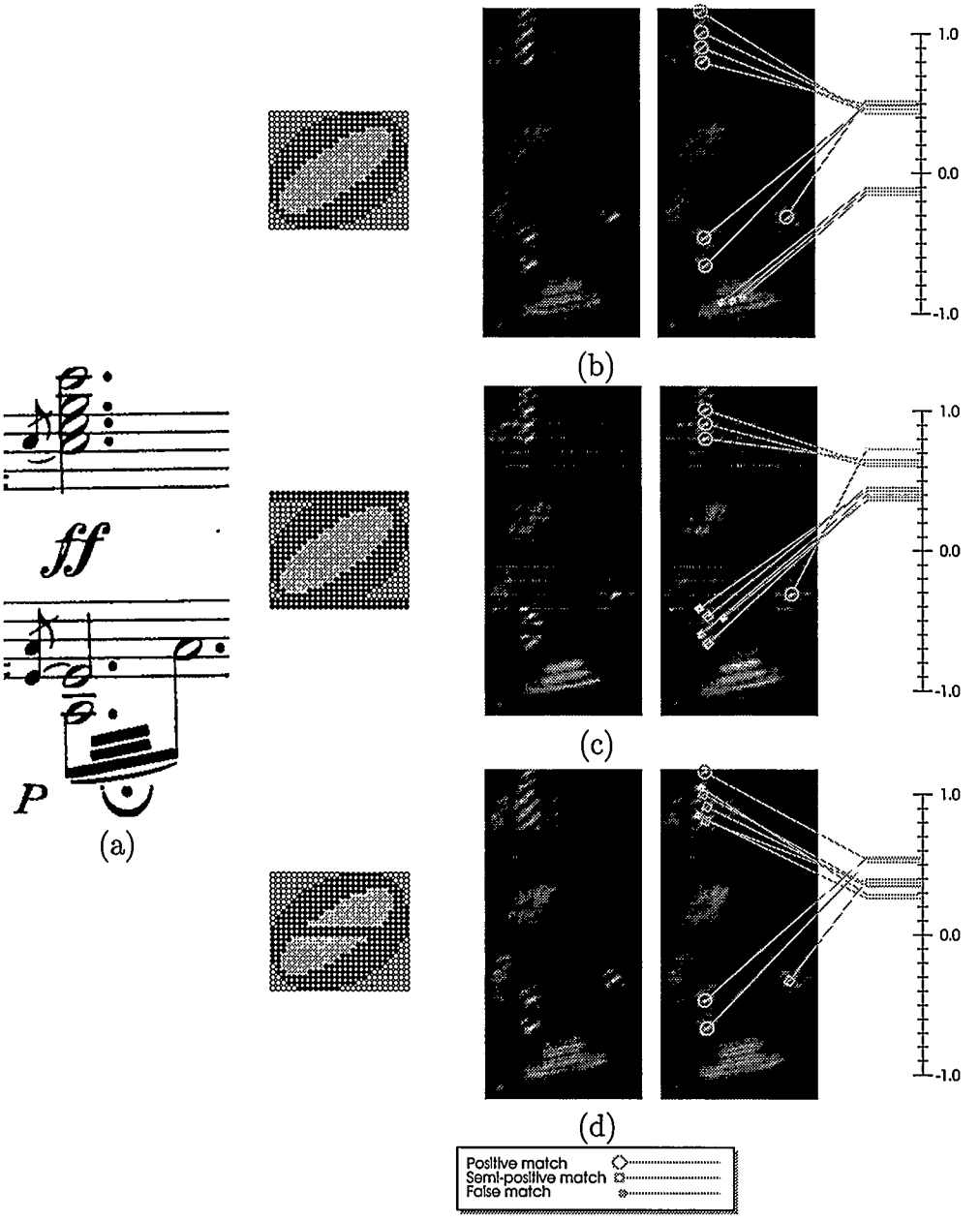


Figure 4.11: Half note template matching using the modified similarity metric.

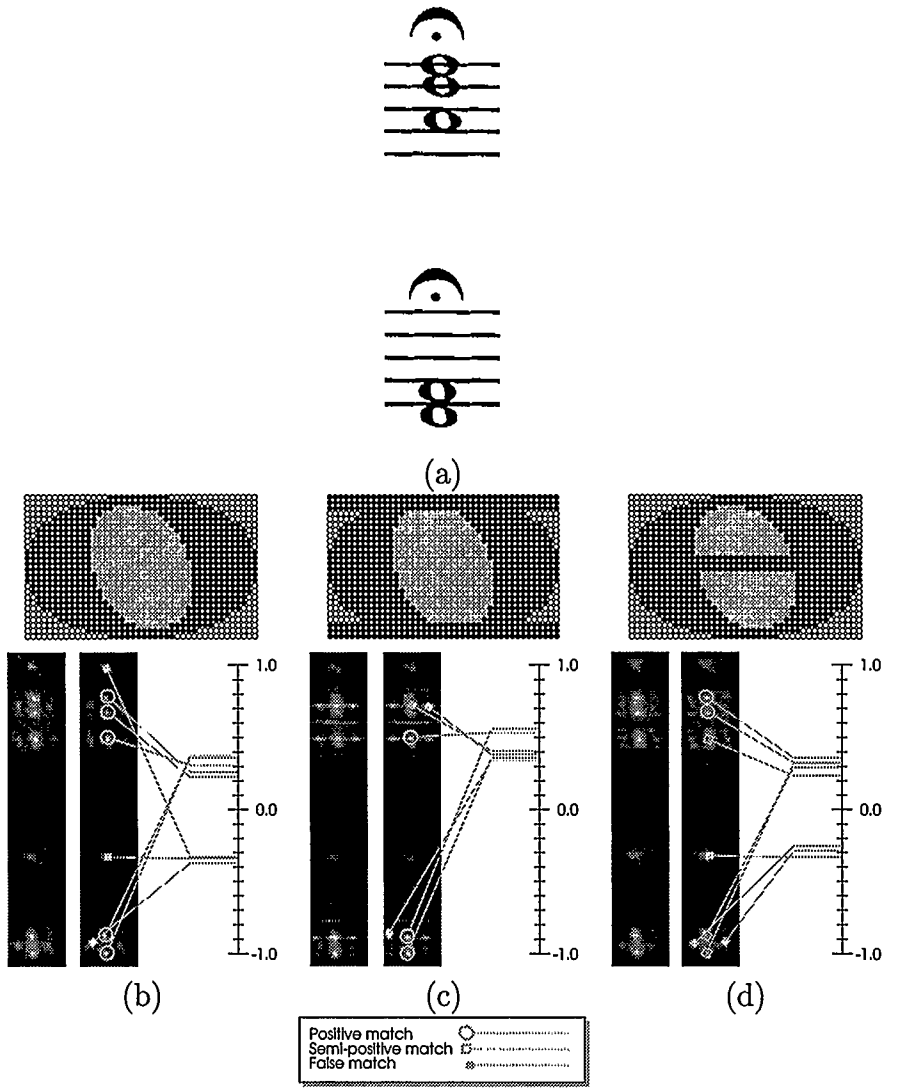


Figure 4.12: Whole note template matching using the modified similarity metric.

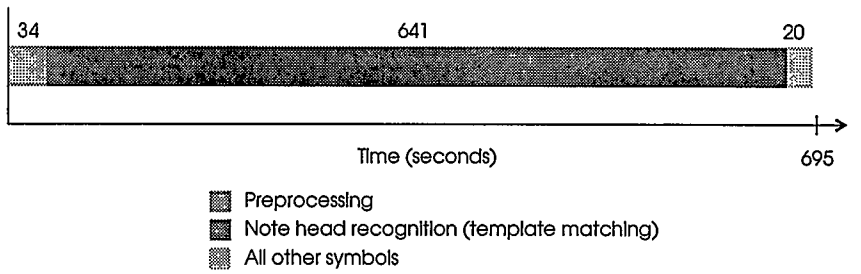


Figure 4.13: Run-time statistics obtained from running the OMR system on the last system of Figure B.11.

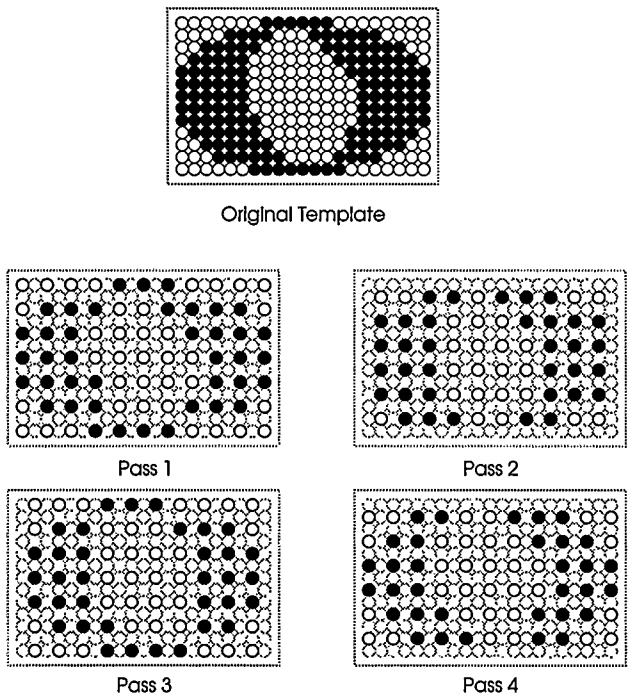


Figure 4.14: Optimized template matching.

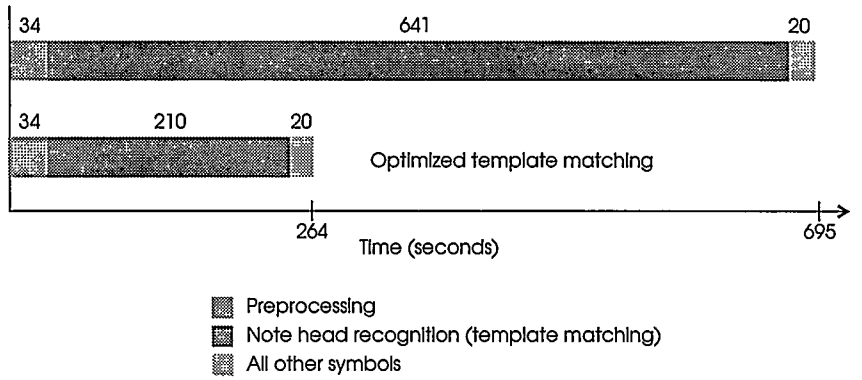


Figure 4.15: Run-time statistics obtained from running the OMR system on the last system of Figure B.11 with and without optimized template matching.

Chapter 5

Contextual Recognition

The previous two chapters showed how to pre-process, segment, and classify primitive symbols from a digitized image of a music score. This chapter will examine the final recognition task of description. Description is necessary to extract semantic meaning from the set of symbols obtained by previous methods. Semantics are determined from the context of symbols – the geometric and spatial relationships between known symbols – hence the chapter title "Contextual Recognition."

Few OMR researchers have discussed in the literature how they implement semantic interpretation. The adoption of ad hoc, rather than formal, techniques is a likely explanation for this void in the literature[1]. To the author's knowledge, there have been only three significant efforts to formalize the semantic interpretation phase of OMR. These efforts are summarized in Section 2.3.4 on page 20.

The method of graph grammars described in [Fahmy 92, Fahmy 93] will be examined, critiqued, and extended in this chapter. This work was adopted as a foundation over the others because it is the most developed and general methodology.

[1]The other explanation is that few existing OMR systems do semantic interpretation – a conclusion supported by the many number of papers and thesis that show recognition results as a textual list of symbols rather than a music description language, or better yet, an automatic reconstruction of the original score.

5.1 Graph Grammars

String language theory has proved to be a powerful and practical tool for building parsers that "understand" languages. However, the sequential, one-dimensional structure of strings preclude using traditional string grammars for parsing pictorial data since no succinct linear ordering of terminals (*i.e.* primitive elements) exists. Graph grammars are an extension of traditional string grammars that manipulate graphs, and provide a logical methodology for "understanding" pictorial data. An extensive collection of graph grammar literature can be found in [Claus 78, Ehrig 82, Ehrig 86, Ehrig 90]; also, the introduction from [Fahmy 93] includes an overview of graph grammars. The following section provides a brief introduction.

5.1.1 Definitions

Graphs

A *graph* is a set of vertices, possibly connected by edges. Both vertices and edges may have additional attributes that convey semantic meanings. In the scope of recognition, graph nodes represent the primitive symbols, and edges represent relationships between these symbols. Figure 5.1 shows an equivalent graph representation for a sample of music.

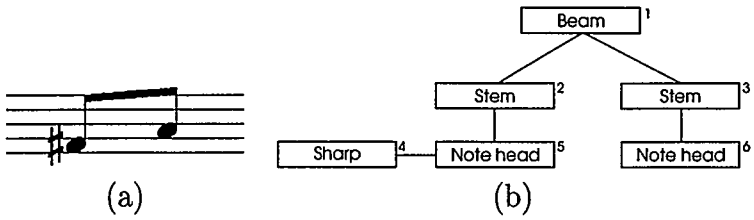


Figure 5.1: A graph representation for sample of music. (a) Sample music. (b) Graph representation of (a)..

Graph Grammars

In the most general case, a (sequential) graph grammar[2] is a set of productions. Each production, P, is a triple (G_l, G_r, E), where G_l and G_r are respectively, the left and right sides of the production, and E is an *embedding transformation.* Borrowing from string grammar syntax, one might write:

$$G_l \xrightarrow{E} G_r$$

An application of a production is a substitution of G_l with G_r inside the input host graph G^{host}. If G_l^{host} is a subgraph of G^{host} that is *isomorphic*[3] to G_l, then G_l^{host} is replaced by G_r. The embedding transformation, E, describes how G_r is embedded into the *restgraph*, the graph obtained by removing G_l^{host} from G^{host}. The application of a graph grammar production is illustrated in Figure 5.2. This example is typical of the productions used: the initial graph is disconnected, and the production simply adds edges between certain nodes.

The astute reader may have noticed that in Figure 5.2 there are four subgraphs in (b) isomorphic with the left hand side of the production. The geometric layout of the graph nodes, however, only make two of these subgraphs immediately obvious[4]. *Applicability predicates* provide a mechanism to constrain which isomorphic subgraphs are eligible for production applications. Each production is augmented by an applicability predicate, a boolean condition that must be satisfied for the production to be applied. The predicates usually use additional attributes attached to the nodes.

[2]Graph grammars are commonly categorized as *sequential* or *parallel.* Only sequential graph grammars are considered here.

[3]Though the definition carries more meaning, for our purposes *isomorphic* is synonymous with *equivalent.* Testing for graph isomorphism is computationally inefficient – no (known) polynomial time algorithm exists for determining whether two graphs are isomorphic (*i.e.* the problem is NP-complete). Consequently, graph grammars can be extremely expensive.

[4]The four subgraphs in (b) isomorphic to (1,2) in (a) are: (2,5), (2,6), (3,5), and (3,6).

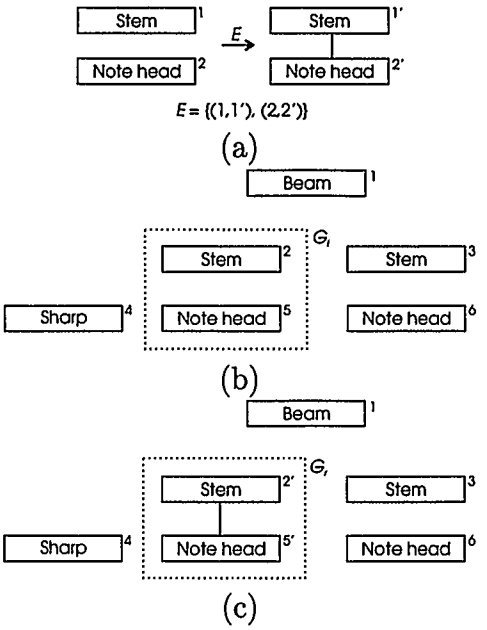


Figure 5.2: A graph grammar production. (a) The production. (b) The host graph before applying the production (the isomorphic subgraph is boxed). (c) The host graph after applying the production.
Note that the production could be applied a second time since another isomorphic subgraph exists in (c).

A reasonable applicability predicate for the production in Figure 5.2 would be:

$$P : \mathbf{staff}(1) = \mathbf{staff}(2) \textbf{ and } |\mathbf{x}(1) - \mathbf{x}(2)| < S \textbf{ and } \mathbf{y}(2) \epsilon \mathbf{yrange}(1)$$

where S is the spacing of the staff lines. This predicate has the effect of only applying the production if: nodes 1 and 2 are on the same staff; the x coordinates of nodes 1 and 2 are within one staff unit; and the y coordinate of node 2 (the note head) is within the range of y values occupied by node 1 (the stem).

When parsing string data, there is a well defined ordering, namely reading order, of the terminals. During the parsing process, the current state of the parser (*i.e.* the history of productions already applied) and the next input terminal are used to decide the next production to apply. Hence, the string grammar controls the parser. Achieving the same level of automation in graph grammars is only possible with special classes of graph grammars. Often, these grammars lack the expressive power to solve the problem at hand, thus an alternative parsing method is required.

Programmed grammars are grammars that include additional instructions to control the order of production applications. In a sense, the term *graph grammars* is a misnomer, because of the strong connotation between *grammars* and *(automatic) parsing*. Some authors prefer *graph rewriting system* over *graph grammar* to emphasize that the technique is usually used as a formalism for specifying a sequence of graph transformations.

Programmed grammars are typically expressed graphically by *control diagrams*; control diagrams are essentially flow charts, indicating what production is applied when.

5.1.2 Previous Work – MUBEEN

In [Fahmy 93] the authors use a programmed graph grammar for high-level recognition of music notation. Their system, called MUBEEN, assumes the existence of a

primitive symbol recognizer; the authors simulated a perfect symbol recognizer for the development of MUBEEN.

MUBEEN uses a three-phase paradigm the authors term *Build-Weed-Incorporate.* The input graph is a completely disconnected graph with each node representing a primitive symbol. Attributes associated with each node include the symbol type, the $(x; y)$ coordinates of the symbol, what staff the symbol belongs to, and the vertical position, called the *space code*[5], on the staff. The *Build-Weed-Incorporate* graph grammar programming paradigm is illustrated by an extended example below; the example demonstrates how accidentals are associated with note heads. Figure 5.3 shows the original music sample and corresponding graph used in the example.

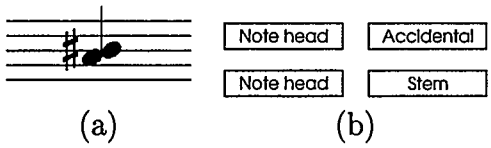


Figure 5.3: Original input graph to be parsed by graph grammar. (a) The original music notation. (b) The corresponding graph representation.

Build The MUBEEN graph grammar introduces edges between nodes A and B if there is a potential semantic relationship between A and B. Edges are not labeled. Rather, the precise meaning of the edge relationship is determined by the types of symbols that are connected. See Figure 5.4.

Weed Filter out conflicting or uninteresting associations built in the previous step by removing edges. See Figure 5.5.

Incorporate Semantics conveyed by edge connections are incorporated into node attributes. See Figure 5.6

[5]The term *space code* comes from DARMS, the music description language adopted in this thesis. Appendix A provides an introduction to DARMS.

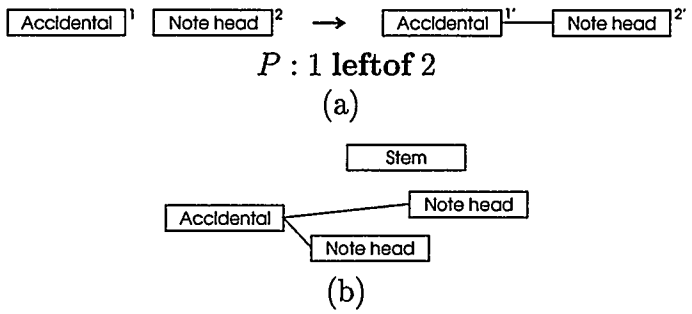


Figure 5.4: MUBEEN Build production. An edge between a note head and an accidental is added if the accidental is positioned to the left of the note head. (a) The graph production. (b) The result of applying the production repeatedly (as often as possible).

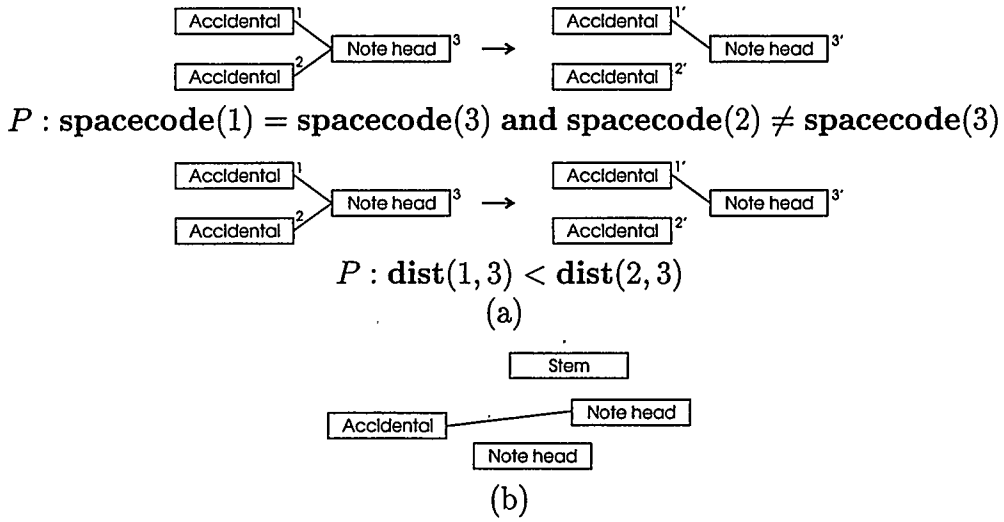


Figure 5.5: MUBEEN Weed production. A note head may have edges to two or more accidentals. Productions are applied to select the best accidental by deleting edges to all but one accidental. Here, two productions are used: the first is used to select accidental/note pairs that have identical vertical positions; if no such pairs exist, the second production will select nearest neighbour accidental/note pairs. (a) The graph productions. (b) The result of applying the productions.

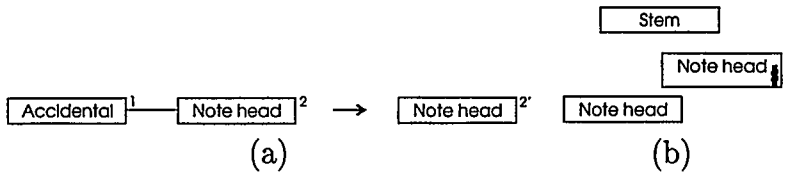


Figure 5.6: MUBEEN Incorporate production. A note head connected to an accidental will be transformed to just a note head; the accidental becomes encoded as an attribute of the note head. Fahmy and Blostein call the accidental the *donor* node, and the note head the *acceptor* node. (a) The graph productions. (b) The result of applying the productions.

Accounting for Uncertainty

In [Fahmy 92][6] the MUBEEN authors extend their work to account for uncertainty in the primitive symbol recognizer.

It is assumed that the symbol recognizer produces a list of probabilistic tokens, tokens that represent a set of probable symbols. For example, a token might be $\{\text{WholeNote}_{0.7}, \text{HalfNote}_{0.3}\}$, meaning with probability 0.7 the token is a whole note, and with probability 0.3 the token is a half note. Tokens always represent sets of syntactically equivalent symbols, such as note heads or accidentals; a token cannot contain a sharp and a whole note because they belong to different syntactic classes. This restriction on tokens is circumvented by allowing multiple, mutually exclusive tokens. So, if the symbol recognizer thinks a symbol is a either a bass clef or a half note, two tokens would be generated, but only one token can participate in a successful interpretation by the parser.

Each token is represented by a single graph node. In the case of multiple tokens just described, the corresponding nodes are connected by an *exclusion edge*. With the exception of exclusion edges, the input graph is disconnected, as before.

The input graph is processed in a similar manner as before. Fahmy and Blostein

[6] Although [Fahmy 92] precedes [Fahmy 93] in publication dates, the former is the more current work.

dub the modified scheme the *Build-Constrain-Incorporate* methodology. During the *Build* phase, edges are added to convey semantic relationships; these edges are labelled as *potential semantic associations* to distinguish them from exclusion edges. Unlike before, edges carry an *affinity* attribute, used to measure the strength of the relationship represented by the edge. The *Constrain* phase, formerly the *Weed* phase, deduces the "best" interpretation of the graph by removing conflicting nodes and edges, or *anchoring* nodes. Anchoring a node means to definitely label a node's symbol type. Knowledge of music syntax and the uncertainty values encoded in the graph are used to design graph productions that result in the "best" output graph. Finally, the *Incorporate* phase amalgamates related nodes, encoding necessary attributes in the *acceptor* node.

5.1.3 Critique of MUBEEN

In [Fahmy 93, Fahmy 92], the authors dealt exclusively with the problem of high level recognition, and treated primitive symbol extraction as an independent problem. Furthermore, during the development of their system, MUBEEN, they simulated the output of a hypothetical recognition system that identifies primitive music symbols. An almost inevitable consequence of relying on simulated, rather that actual, data is that practical problems are naïvely avoided[7]. With the experience gained by implementing an OMR system, several problems with MUBEEN have been identified. These are summarized below:

- MUBEEN places too much responsibility on the symbol recognizer. Stems and barlines, for example, are considered primitive symbols. For the symbol recognizer to make this distinction, it is necessary to use additional information. The decision whether a vertical line is a stem or barline can be made by con-

[7]In [Fahmy 92], the authors indicated the intention to use real data in subsequent work.

text (stems have notes attached to them), thus are candidates for contextual recognition via graph grammars.

- Although separating primitive symbol recognition from high level interpretation is appealing from an organizational view point, several advantages can be realized by incorporating domain knowledge during symbol recognition. For example, filled and half note heads are (almost) always attached to stems. This information can be used to isolate where to look for note heads. The integration of symbol recognition and parsing is a major goal sought in [Couasnon 94, Couasnon 95].

- The graph productions employed by MUBEEN, especially the latter version that considers uncertainty, are based on heuristics and expectations of what the symbol recognizer produces. Hence, the graph productions are not completely independent from the symbol recognizer. For example, a symbol recognizer might commonly, but incorrectly, find filled note heads within a beam[8]. If the designers of the graph grammar did not anticipate this error, no productions will exists to compensate.

To its merit, the technique of graph grammars and the paradigm adopted by MUBEEN are easily adapted to overcome the aforementioned problems. The next section provides an overview of the author's graph grammar implementation which addresses the problems just discussed.

[8]Indeed, this problem is common in the author's implementation, as is the converse situation – recognizing a beam where there are actually two notes. See Figure 5.3(a) for an example of when this might happen.

5.1.4 Implemented Graph Grammar

This section provides an overview of how graph grammars are applied for contextual recognition. The description is admittedly lacking in precise details. This is so for brevity, and because the graph grammar is tightly coupled with the primitive recognition system, thus limiting its generality and potential to be used by others. Indeed, the implemented graph grammar borrows heavily from MUBEEN, but also makes significant modifications, noted below, for practical reasons. The intent is to summarize, and provide evidence that graph grammars are an effective tool for interpretating semantics from pictorial data.

The following sections describe various aspect of the implemented graph grammar system.

The Input Graph

The input graph is completely disconnected. Probabilistic tokens, and hence exclusions edges, are not supported. However, uncertainty is accounted for by allowing certain tokens to occupy the same space in the xy-plane. Graph productions are specifically designed to disambiguate conflicting tokens.

The list of primitive symbols currently supported are: dots, lines, curves, filled note heads, half note heads, whole note heads, treble clefs, bass clefs, rests (1/1 to 1/16), the three primary accidentals, and stem flags. Note that, unlike in MUBEEN, stems, barlines, beams, and ledger lines are not primitives. These symbols are identified by graph productions based on context. Repeats are similarly identified.

Production Application

Unlike MUBEEN, graph productions are not executed as a post-recognition process. Rather, graph productions are executed intermittently during primitive symbol recog-

nition, when convenient. One advantage of this approach, as previously mentioned, is that primitive symbol recognition can potentially be aided by using contextual knowledge obtained by applying graph productions to previously recognized symbols. The best example of this in the current implementation is for stem flags. Stem flags are, of course, always attached to stems, and searching for stem flags elsewhere would be futile. Limiting the search space for stem flags is only possible if stems have been identified by the prior execution of the appropriate graph productions.

In the current implementation, there are no graph productions that correspond to MUBEEN's *Incorporate* productions. MUBEEN *Incorporate* productions condense the final graph, but do not change the semantics of the graph. In the developed OMR system then, semantics are implied by what connections exist between graph nodes *and* the nodes' attributes, rather than just the nodes' attributes. The difference is inconsequential except for implementation convenience.

Table 5.1 summarizes the primitive symbol and contextual recognition performed.

Goal Graph

The ultimate purpose of applying graph productions is to transform the input graph into the *goal graph* – the most convenient representation from which the music semantics can be extracted. What is convenient depends on the target music representation, of which there are several possibilities (see page 2). The DARMS music representation language [Erickson 83] was adopted as the output format for the developed OMR system. Hence, the goal graph is such that conversion to DARMS is straightforward. This is quite different than the goal graph sought in the MUBEEN system. For example, in MUBEEN, the pitch of each note head is determined based on the active clef and active accidentals in the note's measure. This process is avoided with DARMS, because DARMS describes notes by spaces codes (*i.e.* vertical positions on the staff)

Primitive Symbol Recognition[a]	**Contextual Recognition**
Find dots (any mark sufficiently small).	
Find treble clefs (profiles).	
	Locate bass clefs.
	Locate repeats.
Find whole rests (profiles).	
Find half rests (profiles).	
Find quarter rests (profiles).	
Find eighth rests (profiles).	
Find sixteenth rests (profiles).	
Find flats (profiles).	
Find sharps (profiles).	
Find naturals (profiles).	
Find vertical lines (horizontal runlength based LAG)	
Find filled note heads (template matching).	
Find half note heads (template matching).	
	Label stems by connecting notes to vertical lines.
	Label barlines.
Find curves (vertical runlength based LAG)	
Find horizontal lines (vertical runlength based LAG)	
	Separate curves and horizontal lines.
	Label beams by connecting thick horizontal lines to stems.
Find flags for beamless stems.	
Find whole notes (template matching).	
	Group wholes notes into chord, connecting them to an imaginary stem.
	Associated accidentals, ties, slurs, and duration dots with note heads.

[a]Bracketed text indicates technique used for recognition.

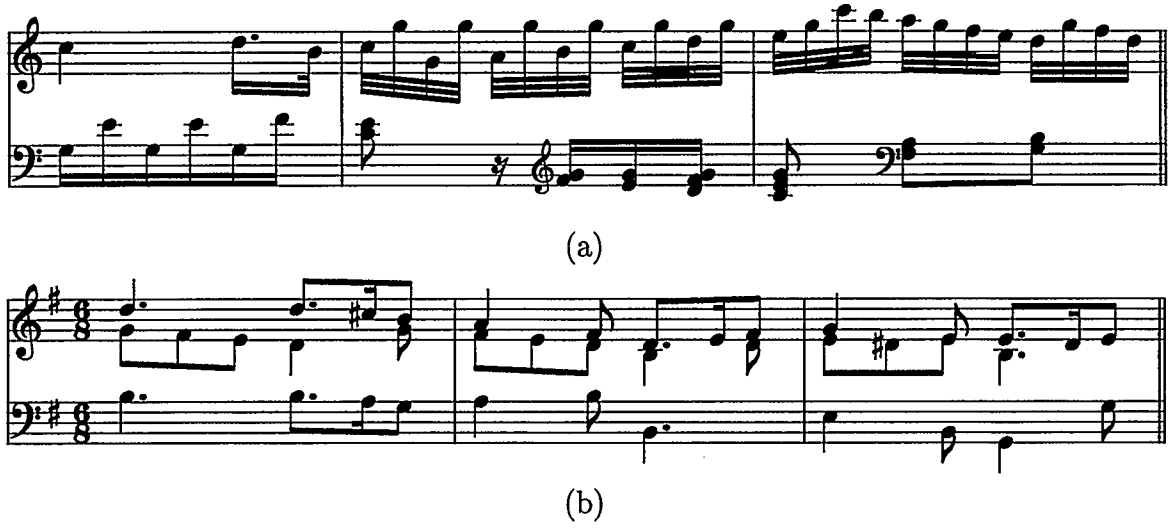
Table 5.1: Summary of primitive symbol and contextual recognition.

rather than pitch. Spaces codes are independent of clefs and accidentals. The actual pitch of the note is deduced by the DARMS interpreter.

Because the goal graph is dependent on DARMS, and a complete description of DARMS is beyond the scope of this discussion, the details of the goal graph format, and the conversion process to DARMS are omitted.

Limitations

The technique used to convert the goal graph to DARMS relies on a linear ordering of specific graph nodes, which are subsequently parsed by a string grammar. Consequently, this approach is limited to interpreting homophonic music only, or more specifically, music with one voice per staff. This restriction is illustrated in Figure 5.7.



(a)

(b)

Figure 5.7: Homophony versus polyphony. (a) A sample of a homophonic score. (b) A sample of a polyphonic score. Within the top staff there are two voices with independent rhythms. Note that some note heads participate in multiple voices. Scores similar to (a) are supported by the current system.

To support polyphony, a linear decomposition algorithm is needed that can ex-

tract each voice in a multi-voice musical phrase. It is not currently known if graph grammars are sufficiently powerful to perform linear decomposition, and this remains a topic for continued research.

Chapter 6

Evaluation

This chapter evaluates the developed OMR system, herein called *Lemon*[1]. A small collection of selected score samples were selected for recognition experiments. From these experiments, recognition rates and running times are reported. Some of the common errors exhibited are identified and possible improvements are discussed. The recognition results are then compared with existing commercial OMR systems to provide a benchmark for comparison. The chapter concludes with a critique of *Lemon*, including suggestions for future research.

6.1 Experimental Results

This section summarizes the experiments conducted to test *Lemon*. Nine typeset score samples and one handwritten sample were selected for testing.

[1]I adopted the name *Lemon* during the initial development stages. At the time, I thought the output produced by my system would be *Tilia*, the music representation language used by the notation editor *Lime* – hence the name *Lemon*. Later, DARMS supplanted *Tilia*, and *The NoteProcessor* supplanted *Lime*. Until this chapter, I planned to keep the system nameless, but after writing the "the developed OMR system" repeatedly, the need for a name was apparent. I reluctantly (for obvious reasons) retained the original name *Lemon*.

6.1.1 Score Selection

Score selection was subject to the following criteria:

- As described in Section 5.1.4, the process of converting the goal graph to DARMS does not support multiple voices on a staff. Thus, polyphonic scores were not considered.

- Scores of varying complexity were selected. This criterion was somewhat compromised by the previous criterion since the most complex music is polyphonic.

- Scores were selected to provide testing of all symbols recognized by *Lemon*. Unfortunately, because some symbols appear infrequently in most music, recognition rates for these symbols are based on a comparatively smaller sample. Whole notes, for example, are much less frequent than quarter notes.

Table 6.1 lists the selected score samples. The samples can be found in Appendix B.

6.1.2 Measuring Recognition Performance

Recognition performance is conventionally measured by determining the error exhibited by the recognition system on a test corpus. There are two commonly used error metrics:

- A type I error, or *error of omission*, measures how many symbols of class c_i were missed:

$$E_I^i = 1 - \frac{\text{number of class } c_i \text{ symbols correctly recognized}}{\text{actual number of class } c_i \text{ symbols}} \tag{6.1}$$

  Type I error are usually reported as a recognition rate rather than an error rate; a type I error of 0.05 would be reported as a recognition rate of 0.95. The

Score	Image of Score
Magnificat (Vom Himmel hoch)[a]	Figure B.1
Canon in D[b]	Figure B.3
Concerto No. 1[c]	Figure B.5
Polovetzian Dance[c]	Figure B.7
The Entertainer[c]	Figure B.9
Klavierstück (Für Elise)[d]	Figure B.11
Moonlight Sonata[c]	Figure B.13
Sonata in C Major[e]	Figure B.15
The Four Seasons (Winter)[f]	Figure B.17
Antiphonae Marianae[g]	Figure B.19

[a]Selected from [Bach 79].
[b]Selected from [Pachelbel 77].
[c]Selected from [Sco82].
[d]Selected from [vB50].
[e]Provided by San Andreas Press.
[f]Selected from [Vivaldi 61].
[g]Selected from [Gruber 90].

Table 6.1: Test scores used in recognition experiments.

recognition rate is simply $1 - E_I^i$, or

$$R^i = \frac{\text{number of class } c_i \text{ symbols correctly recognized}}{\text{actual number of class } c_i \text{ symbols}} \tag{6.2}$$

- A type II error, or *error of commission*, measures how many times class c_k symbols were classified as a class c_i symbol ($k \neq i$):

$$E_{II}^i = \frac{\text{number of class } c_k \text{ symbols classified as } c_i \text{ symbols}}{\text{actual number of symbols not in class } c_i} \tag{6.3}$$

It is not immediately clear how to apply these metrics to music recognition. Music notation is sufficiently complex to make tabulating recognition results a non-trivial task. The most fundamental problem is defining precisely what constitutes a symbol. One possible definition defines symbols in terms of the primitives used during recognition: beams, stems, note heads, flags, dots, etc. Alternatively, symbols may be defined in terms of musical constructs: half notes, quarter notes, eighth notes,

rests, etc. The difference can be surprisingly significant when reporting recognition results. Figure 6.1 illustrates how different accounting methods can result in a recognition rate discrepancy of over 80%, in favour of symbols defined as those used during recognition.

The symbol definition adopted defines a symbol as either:

- a character symbol. These include clefs, accidentals, rests, duration dots, and repeats.

- a "minimal musical construct" – a symbol which has meaning in isolation, and cannot be broken down into smaller symbols that still have meaning. By this definition, a note is a symbol. A thirty-second note, though composed of a stem, a note head, and three flags, is only one symbol because none of its composite parts are significant independently. A triad is three symbols. Other symbols in this category include slurs/ties and barlines. Also, for note durations smaller than a quarter note, flagged and beamed notes are distinct.

This definition more or less subscribes to the view that a symbol should be defined in terms musical constructs. The primary motivation for selecting this definition is that it allows comparison with other OMR systems. Had the primitives used during recognition been used to define a symbol, an objective comparison with other systems would be impossible, since the primitives used by each system will likely differ.

Aside from how symbols are defined, it is another matter to even determine if a symbol has been properly recognized. For example:

1. An accidental in the key signature is correctly identified, but incorrectly associated with a note.

2. A half note is correctly identified, but it's pitch is incorrectly determined.

3. A slur is identified, but it's endpoints are associated with the wrong notes.

These issues were resolved almost arbitrarily. Key signature accidentals and note accidentals are considered distinct symbols. Hence, situation 1 above would contribute to both type I and II errors. Recognition rates for notes do not account for pitch – pitch determination is reported separately. For example, if 95 of 100 notes are located, the recognition rate will be reported as 95%. A secondary recognition value reports the fraction of correctly determined pitches. If 93 of the 95 identified notes have the correct pitch, this will be reported as 93/95. Recognition rates are similarly reported for slurs and ties. In this case, the primary recognition rate is augmented by the fraction of slur/tie endpoints correctly associated with notes.

(a) Original score

(b) Reconstructed score

Symbol	Occurences	Recognized (%)	False Identification (%)
Stem	8	8 (100%)	0
Beam	5	4 (80%)	0
Filled note head	8	8 (100%)	0
Slur	1	1 (100%)	0
Total	22	21 (**95%**)	-

(c) Recognition Results

Symbol	Occurences	Recognized (%)	False Identification (%)
Sixty-fourth Note	8	0	0
Thirty-second Note	0	-	8 (89%)
Slur	1	1 (100%)	0
Total	9	1 (**11%**)	-

(d) Recognition Results

Figure 6.1: Reporting recognition results. This example illustrates how the definition of a symbol can significantly influence the perceived accuracy of recognition. (a) The original score. (b) The score showing the results of recognition – one beam was not located. (c),(d) Recognition results using difference definitions for a symbol. In practice, such dramatic differences in recognitions rates are unlikely to occur. Nonetheless, the example demonstrates that reported recognition errors must be accompanied by a detailed description of how errors were calculated.

6.1.3 Results

Table 6.2 provides recognition results and errors for all the test scores listed in Table 6.1. A more detailed, per score summary can be found in Appendix B, where DARMS output and reconstructed scores are also provided. The overall recognition rate is 93%, with type II errors less than 1%. A more meaningful indicator for performance is the per score recognition results, shown in Table 6.3. Most tests produced recognition rates over 95%. The overall recognition rate is skewed downward by the results for *The Four Seasons (Winter)* and *Antiphonae Marianae* whose recognition rates are 78% and 79% respectively. These relatively poor results were expected – *The Four Seasons (Winter)* is a "mini-score" (approximately 30% the size of a regular score) and *Antiphonae Marianae* is a handwritten score.

Table 6.4 summarizes the execution times for each test score. Running times range from 1:21[2] to 8:17; the average running is 5:05.

[2]1 minute, 21 seconds. The alternative, 1 hour, 21 minutes is not unheard of – in [Kato 92] the authors report processing time is "about 90 minutes for one page."

Symbol	Occurences	Recognition Rate (%)[a]	False Identification (%)[b]
Duration Dot	30	27 (90%)	8 (0.1630%)
Treble Clef[c]	25	20 (80%)	0
Bass Clef[c]	15	13 (87%)	0
Repeat	10	10 (100%)	0
Thin Barline	386	385 (100%)	2 (0.0439%)
Thick Barline	14	12 (86%)	0
Whole Rest	21	17 (81%)	3 (0.0610%)
Half Rest	11	7 (64%)	0
Quarter Rest	42	39 (93%)	1 (0.0204%)
Eighth Rest	57	57 (100%)	1 (0.0205%)
Sixteenth Rest	47	46 (98%)	0
Flat (in key signature)[c]	24	8 (33%)	0
Flat	19	14 (74%)	1 (0.0203%)
Sharp (in key signature)[c]	22	16 (73%)	0
Sharp	50	46 (92%)	0
Natural	30	23 (77%)	0
Whole Note	11	11 (100%)	5 (0.1015%)
Half Note	134	125 (93%)	5 (0.1041%)
Quarter Note	364	349 (96%)	16 (0.3499%)
Flagged Eighth Note	122	117 (96%)	3 (0.0623%)
Beamed Eighth Note	691	658 (95%)	1 (0.0236%)
Flagged Sixteenth Note	0	0	1 (0.0203%)
Beamed Sixteenth Note	455	451 (99%)	14 (0.3124%)
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	99	82 (83%)	0
Slur/Tie	182	132 (73%)	0
Space Code (Pitch)[d]		1786/1793	
Slur/Tie Association[e]		261/264	
Total[f]	2861	2665 (93%)	61

[a]The first number is the count of correct matches. The parenthesized number is the recognition rate, evaluated from Equation 6.2, and expressed as a percent.

[b]The first number is the count of false identifications. The parenthesized number is an upper bound on the type II error, expressed as a percent. This upper bound is calculated by,

$$\tilde{E}_{II}^{i} = \frac{\text{number of class } c_k \text{ symbols classified as } c_i \text{ symbols}}{\text{number of connected components-number of class } c_i \text{ symbols}}$$

This is the same as Equation 6.3, except the denominator of the fraction has been replaced by a conservative estimate for the actual number of symbols not in class c_i. The connected components are determined after the staff lines have been removed (see Figure 3.16). The reason for this approach it that it's too tedious to manually determine the number of symbols not in class c_i, while the number of connected components can be automatically counted.

[c]Only the first system is considered for clefs and key signatures.

[d]Indicates how many notes had their vertical position correctly identified. Statistics are only for correctly identified notes.

[e]Indicates how many slur/tie end points were associated with the correct note. Statistics are only for correctly identified slurs/ties.

[f]Total excludes "space code" and "slur/tie association" statistics.

Table 6.2: Overall recognition results for all test samples.

Score	Symbols	Recognized	False Identifications
Magnificat (Vom Himmel hoch)	254	250 (98%)	4
Canon in D	232	231 (100%)	1
Concerto No. 1	262	246 (94%)	9
Polovetzian Dance	299	287 (96%)	6
The Entertainer	247	224 (91%)	3
Klavierstück (Für Elise)	548	542(99%)	5
Moonlight Sonata	179	172 (96%)	0
Sonata in C Major	270	267 (99%)	2
The Four Seasons (Winter)	344	267 (78%)	24
Antiphonae Marianae	224	177 (79%)	11

Table 6.3: Per score recognition results summary.

Score	Running Time[a]
Magnificat (Vom Himmel hoch)	1:56
Canon in D	3:52
Concerto No. 1	7:28
Polovetzian Dance	8:17
The Entertainer	1:54
Klavierstück (Für Elise)	5:04
Moonlight Sonata	5:38
Sonata in C Major	3:41
The Four Seasons (Winter)	1:21
Antiphonae Marianae	1:35

[a]Times are given in minutes and seconds. Experiments were conducted on a 486 33MHz PC with 16MB RAM running OS/2.

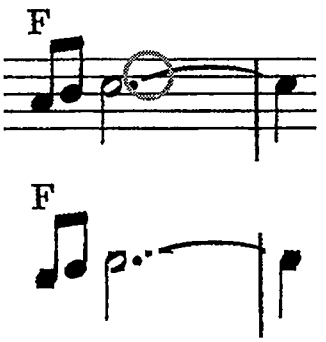
Table 6.4: Running times.

6.2 Analysis

While evaluating *Lemon*, several common errors were identified. This section discusses the most frequently encountered errors, and provides suggestions on how to reduce them.

6.2.1 Problems with Staff Line Removal

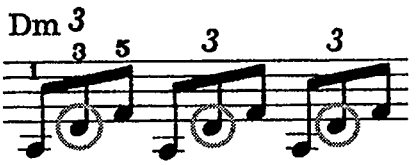
The removal of staff lines often causes superfluous dots to be created, if the staff lines bisect another symbol. This happens most frequently with the tips of slurs and ties. Indeed, every false identification of a duration dot in the conducted experiments was a consequence of this staff line removal side effect.



One possible solution, which is not entirely adequate, is to employ graph grammars. Normally, dots are associated with only note heads. If dots are also associated with the tips of slurs/ties, a graph grammar production can be used to decide whether the dot is a genuine dot, or a bogus dot that really is an extension of the slur/tie. This decision would be based on a how well the dot is aligned with an extrapolated slur/tie. Such a technique can easily be fooled.

Another side effect caused by the removal of staff lines
is the accidental removal of slurs/ties that are tangential with the staff lines. The effect is a splitting of a slur/tie into two. Usually the residuals of the slur/tie are too small to be considered significant, and are discarded as noise. This type of error accounts for the low recognition results obtained for slurs and ties.

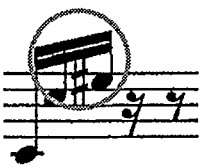
6.2.2 Template Matching



Recall that two distinct templates are used for detecting note heads: one with a staff line through the center of the note head, the other with the note sandwiched between two staff lines. These templates correspond to the possible vertical positions of a note relative the staff. There are two cases, however, that are unaccounted-for – a note may have only *one* staff line (or ledger line) above or below it if the note is below or above the staff. Consequently, the match index computed for such notes will be relatively low and may produce a type I error.

Without boycotting the use of templates, two solutions exist: two additional templates can be used, or the system can draw in ledger lines in the original image[3]. The latter solution would be rendered useless by certain scores that have different vertical spacing for staff lines and ledger lines. In Figure B.3 for example, the spacing between ledger lines is visibly greater than the spacing between staff lines.

6.2.3 Character Symbols



The technique adopted for recognizing character symbols assumes that the characters can be isolated. This is frequently not the case, especially for accidentals which often touch beams, notes, or other accidentals. Recognizing symbols that intersect other symbols remains an area for future research. Several ad hoc solutions exist for special cases. For example, because all accidentals have at least one straight line component, the presence of a line can indicate the possible presence of an accidental. A technique such as template matching can then be used to confirm the existence or non-existence of an accidental.

[3]Before modifying the original image, it would be prudent to make a copy of the affected region, in case no notes are found.

Another solution, discussed in [Couasnon 94], is to delete symbols in the original image once they have be identified, and then re-segment the image. In the above example, the removal of the beams would result in an isolated sharp, which could then be identified. This is the most general and promising solution known to date. However, it is not without deficiencies. Most notably, touching symbols of the same class still cannot be segmented. For example, if two accidentals intersect, removing all other symbols will not separate them.

6.3 Comparison with Commercial OMR Systems

Many other OMR researchers report recognition rates between 90%-100%. However, as demonstrated earlier in this chapter, recognition results are extremely sensitive to the adopted method for tabulating errors. Hence, comparisons with figures reported in the literature are of limited value.

To provide a concrete foundation for comparison with other OMR systems, the experiments conducted to test *Lemon* were repeated with two existing commercial OMR systems, *MIDISCAN 2.0*[4] and *NoteScan 1.04*. Although these experiments were conducted in an identical fashion as the initial experiments, the results are necessarily more complex because of differences between the systems. The following list summarizes known difference between the OMR systems:

- Neither *MIDISCAN* nor *NoteScan* recognize slurs, although *MIDISCAN* does recognize ties.
- *Lemon* does not recognize alto clefs, while both *MIDISCAN* and *NoteScan* do.
- *NoteScan* does not recognize repeats, or thick barlines[5].

[4]The demo version of *MIDISCAN 2.0*, obtained from http://www.musitek.com was used for the experiments.

[5]Or *NoteScan* failed to recognize all repeats and thick barlines in the experiments.

Because of these discrepancies, most results reported below are based on the subset of symbols recognized by all systems. A myriad of comparisons are presented below in table format. Table 6.5 and Table 6.6 summarizes the recognition rates and type II errors respectively, on a symbol basis for all systems. Table 6.7 and Table 6.8 summarize recognition rates and type II errors on a per score basis. Finally, Table 6.9 summarizes the running times for each system. A synopsis of these results:

Recognition Rate *Lemon* (95%) marginally bested *MIDISCAN* (92%). *NoteScan* (82%) placed a distant third.

Type II Error In all experiments, *Lemon* had a total of 61 false identifications (type II errors) – less than half the errors produced by *MIDISCAN* (130), and less than one third the errors produces by *NoteScan* (201).

Run-time Because each system was tested on a different hardware/software configuration, definitive run-time results are not available. Nonetheless, the results suggest *MIDISCAN* is the fastest of the systems; *Lemon* and *NoteScan* posted similar running times.

Symbol	Occurences	Recognition Rate (%)		
		Lemon	*MIDISCAN*	*NoteScan*
Duration Dot	30	27 (90%)	**28 (93%)**	5 (17%)
Treble Clef	25	20 (80%)	**25 (100%)**	20 (80%)
Bass Clef	15	13 (87%)	**14 (93%)**	9 (60%)
Repeat†	10	**10 (100%)**	**10 (100%)**	0 (0%)
Thin Barline	386	**385 (100%)**	376 (97%)	375 (97%)
Thick Barline†	14	12 (86%)	**14 (100%)**	0 (0%)
Whole Rest	21	17 (81%)	20 (95%)	**21 (100%)**
Half Rest	11	7 (64%)	**8 (73%)**	7 (64%)
Quarter Rest	42	**39 (93%)**	38 (90%)	13 (31%)
Eighth Rest	57	**57 (100%)**	32 (56%)	26 (46%)
Sixteenth Rest	47	**46 (98%)**	43 (91%)	41 (87%)
Flat (in key signature)	24	8 (33%)	**22 (92%)**	9 (38%)
Flat	19	14 (74%)	**19 (100%)**	11 (58%)
Sharp (in key signature)	22	**16 (73%)**	14 (64%)	11 (50%)
Sharp	50	**46 (92%)**	40 (80%)	24 (48%)
Natural	30	**23 (77%)**	**23 (77%)**	16 (53%)
Whole Note	11	**11 (100%)**	2 (18%)	8 (73%)
Half Note	134	**125 (93%)**	**125 (93%)**	103 (77%)
Quarter Note	364	349 (96%)	350 (96%)	**356 (98%)**
Flagged Eighth Note	122	**117 (96%)**	78 (64%)	91 (75%)
Beamed Eighth Note	691	658 (95%)	**660 (96%)**	589 (85%)
Flagged Sixteenth Note	0	0	0	0
Beamed Sixteenth Note	455	**451 (99%)**	439 (96%)	370 (81%)
Flagged Thirty-second Note	0	0	0	0
Beamed Thirty-second Note	99	82 (83%)	**83 (84%)**	78 (79%)
Slur/Tie†	182	**132 (73%)**	24 (13%)	0 (0%)
Space Code (Pitch)		1786/1793	1735/1737	1593/1595
Slur/Tie Association		261/264	48/48	NA
Total[a]	2861	**2665 (93%)**	2487 (87%)	2183 (76%)
Total[b]	2655	**2511 (95%)**	2439 (92%)	2183 (82%)

[a]For all symbols listed in table.

[b]Only for symbols recognized by all systems. (Excluded symbols are marked with †).

Table 6.5: Comparison of overall recognition rates for all test samples. The best result in each row is printed in **bold**. See footnotes from Table 6.2 for a description of how recognition rates are calculated.

Symbol	False Identification (%)		
	Lemon	*MIDISCAN*	*NoteScan*
Duration Dot	8 (0.1630%)	**3 (0.0611%)**	**3 (0.0611%)**
Treble Clef	**0 (0.0000%)**	1 (0.0204%)	4 (0.0814%)
Bass Clef	**0 (0.0000%)**	1 (0.0203%)	1 (0.0203%)
Repeat	**0 (0.0000%)**	**0 (0.0000%)**	NA
Thin Barline	2 (0.0439%)	2 (0.0439%)	**0 (0.0000%)**
Thick Barline	**0 (0.0000%)**	**0 (0.0000%)**	NA
Whole Rest	3 (0.0610%)	4 (0.0814%)	**1 (0.0203%)**
Half Rest	**0 (0.0000%)**	1 (0.0203%)	**0 (0.0000%)**
Quarter Rest	**1 (0.0204%)**	7 (0.1430%)	7 (0.1430%)
Eighth Rest	**1 (0.0205%)**	5 (0.1025%)	6 (0.1230%)
Sixteenth Rest	**0 (0.0000%)**	2 (0.0409%)	2 (0.0409%)
Flat (in key signature)	**0 (0.0000%)**	**0 (0.0000%)**	2 (0.0407%)
Flat	**1 (0.0203%)**	4 (0.0813%)	16 (0.3253%)
Sharp (in key signature)	**0 (0.0000%)**	**0 (0.0000%)**	**0 (0.0000%)**
Sharp	**0 (0.0000%)**	**0 (0.0000%)**	**0 (0.0000%)**
Natural	**0 (0.0000%)**	2 (0.0408%)	**0 (0.0000%)**
Whole Note	5 (0.1015%)	**4 (0.0812%)**	10 (0.2030%)
Half Note	**5 (0.1041%)**	9 (0.1874%)	7 (0.1457%)
Quarter Note	**16 (0.3499%)**	34 (0.7435%)	18 (0.3936%)
Flagged Eighth Note	**3 (0.0623%)**	7 (0.1454%)	32 (0.6646%)
Beamed Eighth Note	**1 (0.0236%)**	28 (0.6594%)	14 (0.3297%)
Flagged Sixteenth Note	**1 (0.0203%)**	4 (0.0810%)	55 (1.1140%)
Beamed Sixteenth Note	14 (0.3124%)	**10 (0.2231%)**	15 (0.3347%)
Flagged Thirty-second Note	**0 (0.0000%)**	**0 (0.0000%)**	4 (0.0810%)
Beamed Thirty-second Note	**0 (0.0000%)**	**0 (0.0000%)**	4 (0.0827%)
Slur/Tie	**0 (0.0000%)**	2 (0.0421%)	NA
Total	**61**	130	201

Table 6.6: Comparison of overall type I errors for all test samples. The best result in each row is printed in **bold**. See footnotes from Table 6.2 for a description of how type I errors are calculated.

Score	Symbols[a]	Recognition Rate		
		Lemon	*MIDISCAN*	*NoteScan*
Magnificat (Vom Himmel hoch)	239	**238 (100%)**	201 (84%)	211 (88%)
Canon in D	206	**205 (100%)**	182 (88%)	151 (73%)
Concerto No. 1	247	235 (95%)	**242 (98%)**	231 (94%)
Polovetzian Dance	281	**274 (98%)**	261 (93%)	243 (86%)
The Entertainer	233	218 (94%)	**230 (99%)**	149 (64%)
Klavierstück (Für Elise)	534	**528 (99%)**	510 (96%)	443 (83%)
Moonlight Sonata	179	**172 (96%)**	**172 (96%)**	133 (74%)
Sonata in C Major	256	**254 (99%)**	243 (95%)	226 (88%)
The Four Seasons (Winter)	264	213 (81%)	**221 (84%)**	208 (79%)
Antiphonae Marianae	214	172 (80%)	187 (87%)	**192 (90%)**

[a]Only symbols recognized by all systems are considered.

Table 6.7: Comparison of per score recognition rates. The best result in each row is printed in **bold**.

Score	Symbols[a]	False Identifications		
		Lemon	*MIDISCAN*	*NoteScan*
Magnificat (Vom Himmel hoch)	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
Klavierstück (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**

[a]Only symbols recognized by all systems are considered.

Table 6.8: Comparison of per score false identifications. The best result in each row is printed in **bold**.

Score	Running Time[a]		
	Lemon[b]	*MIDISCAN*[c]	*NoteScan*[d]
Magnificat (Vom Himmel hoch)	**1:56**	2:31	3:43
Canon in D	3:52	**2:01**	4:16
Concerto No. 1	7:28	**2:18**	4:18
Polovetzian Dance	8:17	**2:57**	4:29
The Entertainer	**1:54**	1:55	3:31
Klavierstück (Für Elise)	5:04	**3:58**	5:27
Moonlight Sonata	5:38	**2:31**	3:54
Sonata in C Major	3:41	**2:35**	3:28
The Four Seasons (Winter)	**1:21**	2:06	2:31
Antiphonae Marianae	**1:35**	1:43	2:34

[a] All times are given in minutes and seconds.

[b] Experiments were conducted on a 486 33MHz PC with 16MB RAM running OS/2. These times are repeated from Table 6.4.

[c] Experiments were conducted on a 486 33 MHz PC with 16MB RAM running Win-OS/2, a Windows 3.1 emulation running under OS/2.

[d] Experiments were conducted on a Power Macintosh 6100/60 AV (60 MHz PowerPC) with 16MB RAM running NoteScan in 68040 emulation mode.

Table 6.9: A comparison of running times for the different OMR systems. The best result in each row is printed in **bold**. Because each OMR system was run under a different hardware/software configuration these figures should not be used to evaluate the relative performance of algorithms used. Rather, the comparison is intended to show that *Lemon* is competitive in run-time performance compared to the commercial systems.

6.4 Summary

This chapter presented empirically measured recognition rates and errors exhibited by *Lemon* when tested against a small test corpus of scores. *Lemon* performed quite well, demonstrated by the high recognition rates (~93%) and low errors reported, and by its favourable comparison to currently available commercial OMR systems.

Chapter 7

Conclusion

7.1 Summary

The primary goal of this thesis was to develop an optical music recognition system, capable of printing a recognized score. In Chapter 6, experimental results demonstrated that this goal has been satisfactorily met. An average recognition rate of 93% was obtained, with low false identifications. Furthermore, the developed OMR system performed measurably better than currently available commercial OMR systems.

As this research was preceded by similar research efforts in the past, the contributions made by this thesis are refinements and extensions of previous work. The major contributions made in this thesis are summarized below:

- The algorithms developed for detecting staff lines are new. The β-Hough algorithm, described on page 32, though not adopted here, can be applied to more general line-finding problems, where it is known that the lines may be slightly curved.

- The integration of text segmentation as a pre-processing step has not been previously employed. The removal of text, especially lyrical underlay, is a sig-

nificant contributing factor to the low false identifications achieved. Indeed, many false identifications made by the commercial systems occurred when letters were mistaken for music symbols (an "o", for example, being classified as a whole note).

- The discussion of template matching techniques in Chapter 4 is the most detailed investigation of template matching for recognizing note heads. Two separate refinements were developed, improving both the accuracy and speed of conventional template matching. Though developed independently here, the technique of partial templates to improve the speed of template matching was developed as early as 1964 [Vanderbrug 77]. The technique has unjustly received limited attention, and hopefully its application here demonstrates its viability.

- In [Fahmy 92, Fahmy 93], the authors develop a graph grammar for high level music recognition. This thesis extended this work. In Chapter 5 a critique of [Fahmy 92, Fahmy 93] showed that designing a graph grammar for contextual recognition independently from the low level symbol recognizer will inevitably avoid practical problems encountered during recognition. The foundation developed in [Fahmy 92, Fahmy 93] was extended and incorporated into the developed OMR system – the first known complete OMR implementation to exploit graph grammars.

The successful implementation of a complete OMR system was demonstrated in the previous chapter. The developed OMR system, *Lemon*, achieved recognition rates as high as 99% on good quality scores. Nonetheless, experiments also demonstrated the need for continued research. Significantly lower recognition rates were posted for scores of poor quality, reduced size, or that were handwritten.

7.2 Analysis and Future Work

In Chapter 2, several problems were identified as being prominent obstacles in the development of a robust OMR system. These problems are recited and addressed below:

- *The graphic nature of music notation results in many symbols being superimposed on other symbols. Techniques are needed that can separate touching symbols, or recognize symbols in the presence of adjoined symbols.*

  This problem is only partially solved in this thesis. The techniques used for recognizing lines and note heads are robust. However, for character symbols, such as clefs, rests, and accidentals, intersecting symbols remain a problem. In Section 6.2.3 on page 103, possible solutions to this problem are discussed.

- *Formal methods are needed to infer a semantic interpretation of the 2 dimensional arrangement of recognized symbols.*

  The use of graph grammars has proven to be a powerful and flexible tool for contextual recognition of symbols, as demonstrated by the results obtained in this thesis. The major shortcoming of the developed OMR system is its limitation to homophony. It remains an open area of research to develop a graph grammar capable of identifying multi-voice staves, and associating each recognized music symbol with the correct voice.

  Although graph grammars have proven effective, research continues in alternative approaches. In [Couasnon 94, Couasnon 95], the authors describe work-in-progress in the use of modified string grammars and Prolog for parsing and segmenting music. The effectiveness of this approach is yet to be determined.

- *Techniques adaptable to a variety of notational styles are needed.*

This thesis does not address this issue. The only known work in this area is described in [Bainbridge 94, Bainbridge 95].

7.3 Conclusion

Optical music recognition is a tool for music acquisition. As computer applications in music continue to mature and become more commonplace, the need for efficient music acquisition methods continues to grow as well. Though OMR systems have been researched since the late sixties, the technology is, ironically, still in its infancy and just starting to displace manual methods in practice. The slow adoption of OMR attests to the immaturity of the technology. The development of a truly robust, comprehensive, and practical OMR system remains a challenge for OMR researchers.

Bibliography

[Adiv 83] Gilad Adiv. Recovering motion parameters in scenes containing multiple moving objects. In *IEEE Computer Vision and Pattern Recognition Conference Proceedings*, pages 399–400, Washington, 1983.

[Andronico 82] Alfio Andronico and Alberto Ciampa. On automatic pattern recognition and acqusition of printed music. In *Proceedings of the International Computer Music Conference*, pages 245–278, 1982.

[Anonymous 80] Anonymous. Away in a manager. In Walter Ehret, George K. Evans, et al., editors, *International Book of Christmas Carols*, page 60. Stephen Greene Press: Brattleboro, Vt., 1980. Music score.

[Bach 79] Johann Sebastian Bach. *Magnificat.* Peters Leipzig, 1979. Music score.

[Bainbridge 91] David Bainbridge. Preliminary experiments in musical score recognition. From ftp://maggia.ethz.ch/pub/roth/omrpapers/-omr.D.Bainbridge.ps.Z, May 1991.

[Bainbridge 94] David Bainbridge. A complete optical music recognition system: Looking to the future. From ftp://maggia.ethz.ch/pub/roth/-omrpapers/dbain_complete.ps.gz, November 1994.

[Bainbridge 95] David Bainbridge. Optical music recognition: Progress report 2. From ftp://maggia.ethz.ch/pub/roth/omrpapers/dbain_prog2.ps.gz, January 1995.

[Baird 92] H.S. Baird, H. Bunke, and K. Yamamoto, editors. *Structured Document Image Analysis.* Springer-Verlag, 1992.

[Baumann 92] Stephan Baumann and Andreas Dengel. Transforming printed piano music into MIDI. In Horst Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition: Proceedings of the International Workshop on Structural and Syntactic Pattern Recognition*, pages 363–372, Bern, Switzerland, August 1992.

[Blostein 92a] Dorothea Blostein and Henry S. Baird. A critical survery of music image analysis. In Baird et al. [Baird 92], pages 405–434.

[Blostein 92b] Dorothea Blostein and Nicholas P. Carter. *Structured Document Image Analysis*, chapter *Recognition of Music Notation: SSPR'90 Working Group Report*, pages 573–574. Springer Verlag, 1992.

[Bulis 92] Alex Bulis, Roy Almog, Moti Gerner, and Uri Shimony. Computerized recognition of hand-written music notes. In *Proceedings of the International Computer Music Conference*, pages 110–112, 1992.

[Bussotti 59] Sylvano Bussotti. Five piano pieces for David Tudor — No. 1. From page 105 of [Stone 80], 1959. Music score.

[Carter 88] N.P. Carter, R.A. Bacon, and T. Messenger. The acquisition, representation and reconstruction of printed music by computer: A review. *Computers and the Humanities*, 22:117–136, 1988.

[Carter 92a] Nicholas P. Carter. A new edition of Walton's Façade using automatic score recognition. In Horst Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition: Proceedings of the International Workshop on Structural and Syntactic Pattern Recognition*, pages 352–362, Bern, Switzerland, August 1992.

[Carter 92b] Nicholas P. Carter. Segmentation and preliminary recognition of madrigals notated in white mensural notation. *Machine Vision and Application*, 5:223–230, 1992.

[Carter 92c] Nicholas P. Carter and Richard A. Bacon. Automatic recognition of printed music. In Baird et al. [Baird 92], pages 456–465.

[Clarke 88] Alastair Clarke, Malcolm Brown, and Mike Thorne. Using a micro to automate data acquisition in music publishing. *Microprocessing and Microprogramming*, 24:549–554, 1988.

[Claus 78] Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors. *Graph Grammars and Their Application to Computer Science and Biology. International Workshop.*, number 73 in Lecture Notes in Computer Science. Springer-Verlag, 1978.

[Couasnon 94] Bertrand Couasnon and Jean Camillerapp. Using grammars to segment and recognize music score. In *International Association for Pattern Recognition Workshop on Document Analysis Systems*, pages 15–27, Kaiserslautern, Germany, October 1994.

[Couasnon 95] Bertrand Couasnon, Pascal Brisset, and Igor Stephan. Using logic programming languages for optical music recognition. In *Third Interna-*

tional Conference on the Practical Application of Prolog, Paris, France, April 1995.

[Dearmer 45] Percy Dearmer, R. Vaughan Williams, and Martin Shaw. *The Oxford Book of Carols.* Oxford University Press, 1945.

[Duggan 92] Mary Kay Duggan. Electronic information and applications in musicology and music theory. *Library Trends*, 40(4):756–780, Spring 1992.

[Dvořák 61] Antonín Dvořák. *Slovanské Tance Op. 46, No. 1–4.* Statni Hudebni Vydavatelstvi, Prague, 1961. Music score.

[Ehrig 82] Hartmut Ehrig, Manfred Nagl, and Grzegorz Rozenberg, editors. *Graph Grammars and Their Application to Computer Science. 2nd International Workshop.*, number 153 in Lecture Notes in Computer Science. Springer-Verlag, 1982.

[Ehrig 86] H. Ehrig, M Nagl, G. Rozenberg, and A. Rosenfeld, editors. *Graph Grammars and Their Application to Computer Science. 3rd International Workshop*, number 291 in Lecture Notes in Computer Science, Warrenton, Virginia, USA, December 1986. Springer-Verlag.

[Ehrig 90] H. Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Graph Grammars and Their Application to Computer Science. 4th International Workshop.*, number 532 in Lecture Notes in Computer Science, Bremen, Germany, March 1990. Springer-Verlag.

[Erickson 83] Raymond Erickson and Anthony 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.

[Fahmy 92] Hoda Fahmy and Dorothea Blostein. Graph grammar processing of uncertain data. In Horst 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.

[Fahmy 93] Hoda Fahmy and Dorothea Blostein. A graph grammar programming style for recognition of music notation. *Machine Vision and Application*, 6:83–99, 1993.

[Fletcher 95] Lloyd Alan Fletcher and Rangachar Kasturi. A robust algorithm for text string separation from mixed text/graphics images. In Lawrence O'Gorman and Rangachar 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.

[Fujinaga 88] Ichiro Fujinaga. Optical music recognition using projections. Master's thesis, McGill University, Montreal, Canada, 1988.

[Fujinaga 92] Ichiro Fujinaga. Optical music recognition system which learns. In *Enabling Technologies for High-Bandwidth Applications*, volume 1785 of *SPIE*, pages 210–217, 1992.

[Gruber 90] Benno Gruber. *Antiphonae Marianae.* Musikverlag Alfred Coppenrath, Altötting, 1990. Music score.

[Illingwort 87] J. Illingworth and J. Kittler. The adaptive Hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):690–698, September 1987.

[Illingwort 88] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer Vision, Graphics, and Image Processing*, 44:87–116, 1988.

[Jain 95] Anil K. Jain and Sushil Bhattacharjee. Text segmentation using Gabor filters for automatic document processing. In Lawrence O'Gorman and Rangachar Kasturi, editors, *Document Image Analysis*, pages 182–197. IEEE Computer Society Press, 1995. Reprinted from *Machine Vision and Applications*, Vol. 15, 1992, pp. 169–185.

[Kamien 90] Roger Kamien, editor. *The Norton Scores. An Anthology for Listening.*, volume 1. W.W. Norton & Company, Inc., 5th edition edition, 1990.

[Kassler 70] Michael Kassler. An essay toward specification of a music-reading machine. In B.S. Brook, editor, *Musicology and the Computer.* CUNY Press, New York, 1970.

[Kassler 72] Michael Kassler. Optical character recognition of printed music: A review of two dissertations. *Perspectives of New Music*, 11(1?):250–254, Fall-Winter 1972.

[Kato 92] Hirokazu Kato and Seiji Inokuchi. A recognition system for printed piano music using musical knowledge and constraints. In Baird et al. [Baird 92], pages 435–455.

[Kimura 91] F. Kimura and M. Shridhar. Handwritten numerical recognition based on multiple algorithms. *Pattern Recognition*, 24(10):969–983, 1991.

[Kiryati 91] N. Kiryati, Y. Eldar, and A.M. Bruckstein. A probabilistic Hough transform. *Pattern Recognition*, 24(4):303–316, 1991.

[Kittler 86] J. Kittler. *Handbook of Pattern Recognition and Image Processing*, chapter *Feature Selection and Extraction*, pages 59–83. Academic Press, 1986.

[Leplumey 93] I. Leplumey, J. Camillerapp, and G. Lorette. A robust detector for music staves. In *International Conference on Document Analysis and Recognition*, pages 902–905, Tsukuba Science City, Japan, 1993.

[Li 85] Xiaobo Li and Richard C. Dubes. The first stage in two-stage template matching. *IEEE Transactins on Pattern Analysis and Machine Intelligence*, 7(6):700–707, November 1985.

[Lindstrom 94] Bob Lindstrom. Musitek MIDISCAN: Optical music recognition is finally a reality. *Electronic Musician*, pages 151–154, February 1994.

[Martin 91] Philippe Martin and Camille Bellissant. Neural networks at differenct levels of a musical score image analysis system. In *Proceedigns of the 7th Scandinavian Conference on Image Analysis*, pages 1102–1109, Aalborg, Denmark, 1991. Pattern Recognition Society of Denmark.

[Martin 92] Philippe Martin and Camille Bellissant. Neural networks for the recognition of engraved musical scores. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(1):193–208, 1992.

[McGee 94] William F. McGee. MusicReader: An interactive optical music recognition system. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Computing in Musicology*, volume 9, pages 146–151. Center for Computer Assisted Research in the Humanities, 1994.

[Modayur 93] Bharath R. Modayur, Visvanathan Ramesh, Robert M. Haralick, and Linda G. Shapiro. MUSER: A prototype musical score recognition system using mathematical morphology. *Machine Vision and Application*, 6:140–150, 1993.

[Morehead 92] Philip D. Morehead. *The New International Dictionary of Music.* Meridian, 1992.

[Mori 95] Shunji Mori, Ching Y. Suen, and Kazuhiko Yamamoto. Historical review of OCR research and development. In Lawrence O'Gorman and Rangachar Kasturi, editors, *Document Image Analysis*, pages 244–273. IEEE Computer Society Press, 1995. Reprinted from *Proc. IEEE*, Vol. 80, No. 7, July 1992, pp. 1029–1058.

[Mozart 77] W.A. Mozart. *Praeludium.* Editio Musica, 1977. Music score.

[Mozart 90] W.A. Mozart. Violin sonata in E-flat major, K.302, second movement. In Roger Kamien, editor, *The Norton Scores: An Anthology for Listening*, volume I: Gregorian Chant to Beethoven, pages 422–426. W.W. Norton & Company, 5 edition, 1990. Music score.

[Nettl 49] Paul Nettl. *Das Veilchen, The Violet.* Storm Publishers: New York, 1949.

[Pachelbel 77] Johann Pachelbel. *Canon in D.* Concordia Publishing House, 1977. Music score, arranged for organ.

[Parker 94a] J.R. Parker. *Practical Computer Vision Using C.* John Wiley & Sons, Inc., 1994.

[Parker 94b] J.R. Parker. Recognition of hand printed digits using multiple parallel methods. In *Golden West International AI Conference*, Las Vegas, June 1994.

[Parker 95] J.R. Parker. Vector templates for handprinted symbol recognition. Technical Report 95/559/11, The University of Calgary, April 1995.

[Pavlidis 82] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press: Rockvill, MD, 1982.

[Prerau 70] David Stewart Prerau. *Computer Pattern Recognition of Standard Engraved Music Notation.* PhD thesis, Massachusetts Institute of Technology, September 1970.

[Prerau 75] David S. Prerau. DO-RE-MI: A program that recognizes music notation. *Computers and the Humanities*, 9:25–29, 1975.

[Pruslin 66] D. Pruslin. *Automatic Recognition of Sheet Music.* PhD thesis, Massachusettes Institute of Technology, June 1966.

[Randriamah 93] R. Randriamahefa, J.P. Cocquerez, C. Fluhr, F. Pépin, and S. Philipp. Printed music recognition. In *International Conference on Document Analysis and Recognition*, pages 898–901, Tsukuba Science City, Japan, 1993.

[Roach 88] J.W. Roach and J.E. Tatem. Using domain knowledge in low-level visual processing to interpret handwritten music: An experiment. *Pattern Recognition*, 21(1):33–44, 1988.

[Schalkoff 92] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* John Wiley & Sons, Inc., 1992.

[Sco82] Hooked on Easy Piano Classics. Joe Goldfeder Music Enterprises & Music Box Dancer Publications Ltd., 1982.

[SF94] Eleanor Selfridge-Field. Optical recognition of musical notation. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Computing in Musicology*, volume 9, pages 109–145. Center for Computer Assisted Research in the Humanities, 1994.

[Stone 80] Kurt Stone. *Music Notation in the Twentieth Century.* W.W. Norton & Company, 1980.

[Tho91] Thoughtprocessors, 584 Bergen Street, Brooklyn, New York 11238. *The NoteProcessor, User's Guide*, 1991.

[Vanderbrug 77] Gordon J. Vanderbrug and Azriel Rosenfeld. Two-stage template matching. *IEEE Transactions on Computers*, 26(4):384–393, April 1977.

[vB50] Ludwig van Beethoven. *Klavierstück Für Elise.* G. Henle Verlag, 1950. Music score.

[vB82] Ludwig van Beethoven. Minuet in g. In *Hooked on Easy Piano Classics*, pages 28–29. Joe Goldfeder Music Enterprises & Music Box Dancer Publications Ltd., 1982. Music score, arranged for piano.

[Vivaldi 61] Antonio Vivaldi. *Le Quattro Stagioni — L'Inverno (The Four Seasons — Winter).* Heugel, 1961. Music score, orchestral arrangement.

[Witten 94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 1994.

[Wolman 92] Amnon Wolman, James Choi, Shahab Asgharzadeh, and Jason Kahana. Recognition of handwritten music notation. In *Proceedings of the International Computer Music Conference*, pages 125–127, 1992.

[Wong 82] K.Y. Wong, R.G. Casey, and F.M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–656, 1982.

[Yadid 92] Orly Yadid, Eliyahu Brutman, Lior Dvir, Moti Gerner, and Uri Shimony. RAMIT: Neural network for recognition of musical notes. In *Proceedings of the International Computer Music Conference*, pages 128–131, 1992.

Appendix A

DARMS Tutorial

DARMS[1] [Erickson 83] (Digital Alternate Representation of Musical Score) is a textual language to describe music notation. This appendix is intended to provide the reader with sufficient knowledge of DARMS to be able to understand the DARMS examples given in this thesis[2].

A.1 Space Codes

DARMS uses *space codes* to describe vertical positions on the staff.

-1 0 1 2 3 4 5 6 7 8 9 10 11

Note that space codes do not uniquely determine pitch. Pitch is determined by the active space code, clef, and the presence of accidentals. Also, space codes that

[1]Formerly the *Ford-Columbia Music Representation*, named after the two principal institutions that sponsored its development, the Ford Foundation and Columbia University.

[2]The description of DARMS here is consistent with the description in [Tho91], which is slightly different than that in [Erickson 83]. However, [Erickson 83] is more complete; [Tho91] only describes the subset of DARMS used by *The NoteProcessor*.

are multiples of 50 signify transposition; *eg.* -50 corresponds to the staff below the current staff (transposition only makes sense when there is more than one staff per system). The instruction !-50 make the staff below the current staff the active staff for all subsequent instructions.

A.2 Duration Codes

Duration codes are mnemonic abbreviations that represented common time durations. These are summarized in Table A.1.

W	Whole
H	Half
Q	Quarter
E	Eighth
S	Sixteenth
T	Thirty-second
X	Sixty-fourth
Y	One-hundred-twenty-eighth
Z	Two-hundred-fifty-sixth

Table A.1: DARMS duration codes.

Duration codes, like notes and rests, can be dotted; *eg.* Q.. = doubly dotted quarter duration.

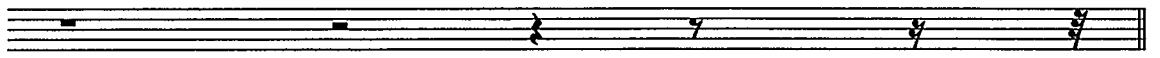
A.3 Notes, Rests, and Non-Printing Rests

Notes are indicated by giving either a space code, a duration code, or both. If either the space code or duration code is omitted, the space code or duration code of the previous note is used. Chords are written as a sequence of notes, separated by commas.

```
1W          2H          3Q     4E 5S 7T   \Q   3Q,5,7     \Q     2H.
```

In this example, *The NoteProcessor* automatically beams notes together when appropriate[3].

Rests are identical to notes, except an R precedes the duration code[4], and rests aren't sequenced together by commas.



```
RW                RH              RQ          RE      \E      RS    \S    RT
```

Non-printing rests have the same effect as a rest, but do not print. Non-printing rests are indicated by \ instead of R. Non-printing rests are sometimes needed in complex polyphonic music[5].

A.4 Symbols

Table A.2 summarizes the remaining DARMS symbols used within this thesis.

A.5 Example

[3]Beams can also be specified explicitly, as is done by the OMR system when generating DARMS output.

[4]Although rests can be positioned vertically by specifying a space code, for single voice music the DARMS interpreter assigns a space code of 5 for rests missing a space code.

[5]In this thesis, non-printing rests are used to make DARMS files acceptable for *The NoteProcessor*'s DARMS interpreter, since recognition errors produce inconsistent DARMS files where the beats per measure disagrees with the declared meter.

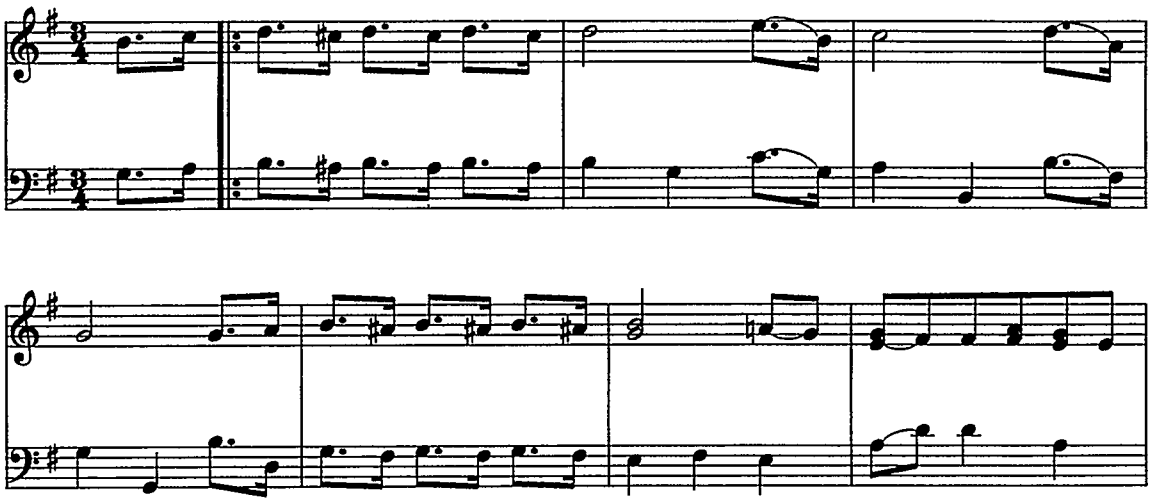
K ... $	Comment
!I*n*	Instrument number
!G	Treble clef
!F	Bass clef
!T	Null clef
!M*n*:*m*	Time signature
!M**n*:*m*	Non-printing time signature
!K*n*-	Key signature (*n* flats)
!K*n*#	Key signature (*n* sharps)
/	Regular (thin) barline
//	Double (thin) barline
/|	Thin and thick barline
/:	Begin repeat
:/	End repeat
:/:	Begin/end repeat
*	Natural
#	Sharp
-	Flat
(^{a}	Start beam
)^a	End beam
Ln^b	Start/end slur
Jn^b	Start/end tie
!R*n*:*m* ... $R^c	Change rhythm

[a] Parentheses can be repeated to indicate several beams. When a group of parentheses is preceded by a semicolon, the beams are short (*i.e.* attached to only one stem)

[b] When n is odd, L starts a slur, and when n is even, L ends the slur started by L$n-1$. Similarly with ties.

[c] Rhythm changes are necessary to encode tuplets. A triplet, for example, is encoded as `!R3:2 5E( 5 5) $R`; each eighth note has a duration of $\frac{2}{3} \times \frac{1}{8}$ beats. See Figure B.13 for an example of a score with triplets. In this thesis, rhythm changes are used to notate ill-formed scores caused by recognition errors, in addition to notating tuplets.

Table A.2: DARMS symbols.



```
!I1 !G,!F
!K1# !M3:4 5E.( 6S);) /:
7E.( 6#S);) 7E.( 6S);) 7E.( 6S);) /
7H 8E.(L1 5S);)L2 /
6H 7E.(L3 4S);)L4 /
!-50
!K1# !M3:4 8E.( 9S);) /:
10E.( 9#S);) 10E.( 9S);) 10E.( 9S);) /
10Q 8 11E.(L5 8S);)L6 /
9Q 3 10E.(L7 7S);)L8 /
!+50
3H 3E.( 4S);) /
5E.( 4#S);) 5E.( 4#S);) 5E.( 4#S);) /
3H,5 4*E(L9 3)L10 /
1E(,3L11 2L12 2 2,4 1,3 1) /
!-50
8Q 1 10E.( 5S);) /
8E.( 7S);) 8E.( 7S);) 8E.( 7S);) /
6Q 7 6 /
9E(L13 12)L14 Q 9 /
```

Figure A.1: DARMS example – a couple bars from *Minuet in G*.

Appendix B

Experimental Results

This appendix provides a detailed account of the experimental results discussed in Section 6.1. For each test score, the following is shown:

- The original bitmap image of the score.
- The reconstructed score, generated from the DARMS output.
- The DARMS output generated by the OMR system.
- A tabular summary of recognition results.

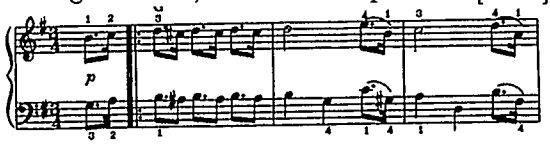
The reconstructed scores were produced by *The NoteProcessor* [Tho91], a commercial music notation software package that interprets DARMS[1]. For *The NoteProcessor* to correctly typeset a score, it expects a reasonably consistent DARMS representation. Specifically, it expects the beats per measure to be consistent with the declared

[1]The availability of *The NoteProcessor* was the primary reason for adopting DARMS as the output format of the OMR system. Although numerous other commercial notation packages are available, *The NoteProcessor* is one of few that uses a published, and public music representation language. Another package, *Lime*, also uses a freely available music language called *Tilia*. This package was rejected in favour of *The NoteProcessor* because *Tilia* is a binary representation, while a DARMS file is a plain text file. Because any OMR will inevitably make mistakes, the ability to easily edit the output is desirable.

meter, and for corresponding measures in a multi-staff system to have the same time count; otherwise, *The NoteProcessor* is unable to correctly generate a score. Because of this, it was usually necessary to manually modify the DARMS files generated by the OMR system, to compensate for recognition errors[2]. These modifications are marked by distinctively typesetting ~~deleted text~~ and inserted text. They are not intended to correct recognition errors, but to produce a score that best represents the results of recognition.

A note about comparing the reconstructed score with the original is warranted. The following aspects of the reconstructed score are determined automatically by *The NoteProcessor*, and not necessarily preserved from the original score: horizontal spacing, stem direction, tie/slur placement, beaming[3], and accidental placement. The number of bars per staff is controlled by user, and was selected to produce a layout similar to the original score when possible. Also, the OMR system does not recognize time signatures or determine the number of staves per system; this information must be provided by the user. The time signature is intentionally not printed in the score.

[2]Or possibly errors in the original score, as in this sample from [vB82]:



[3]A beamed note in the original score will be encoded as a beamed note in the generated DARMS output. Flagged notes in the original score, however, may appear as either flagged or beamed notes in the reconstructed score.

B.1 Test Scores

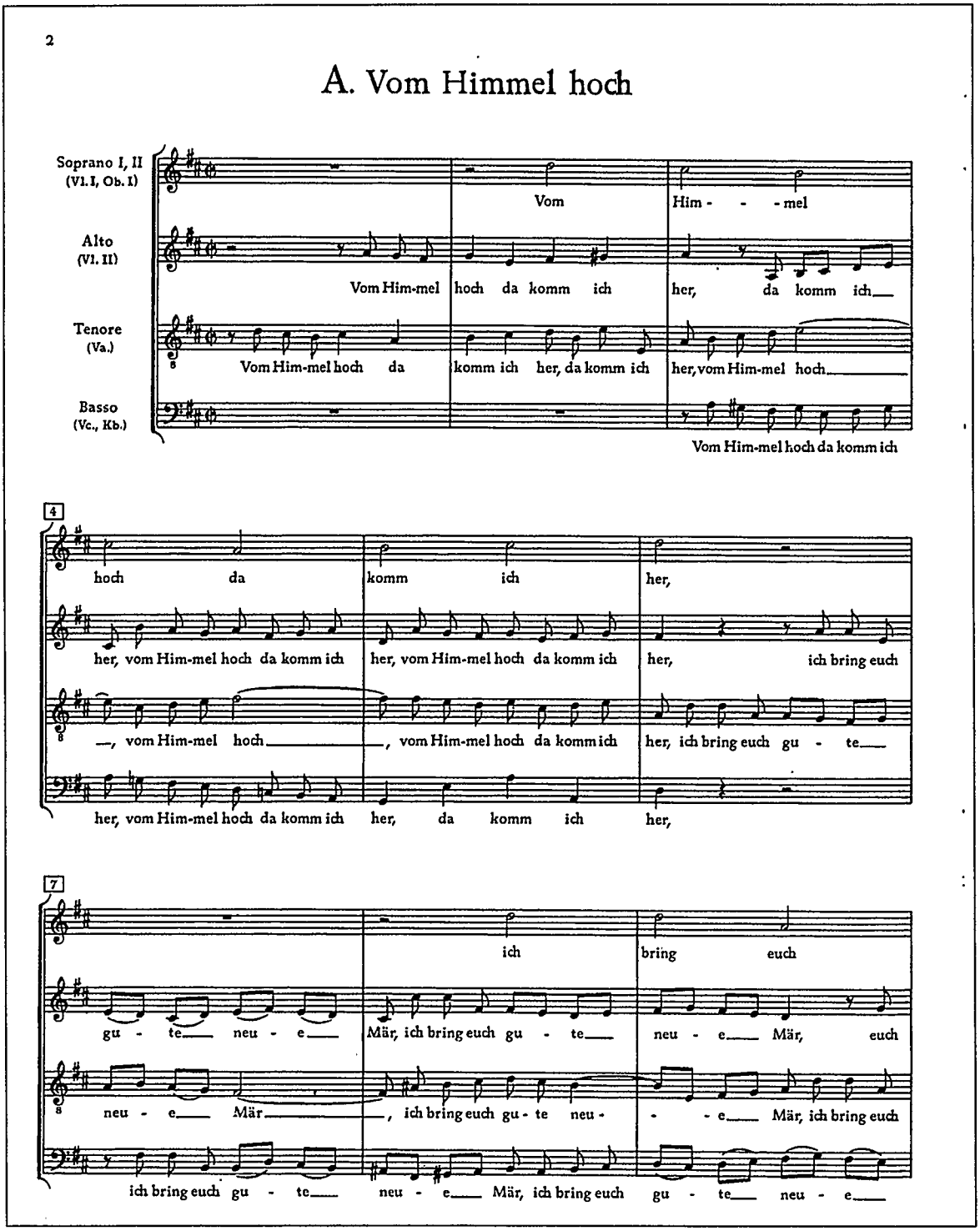


Figure B.1: Test score: *Magnificat (Vom Himmel hoch).*

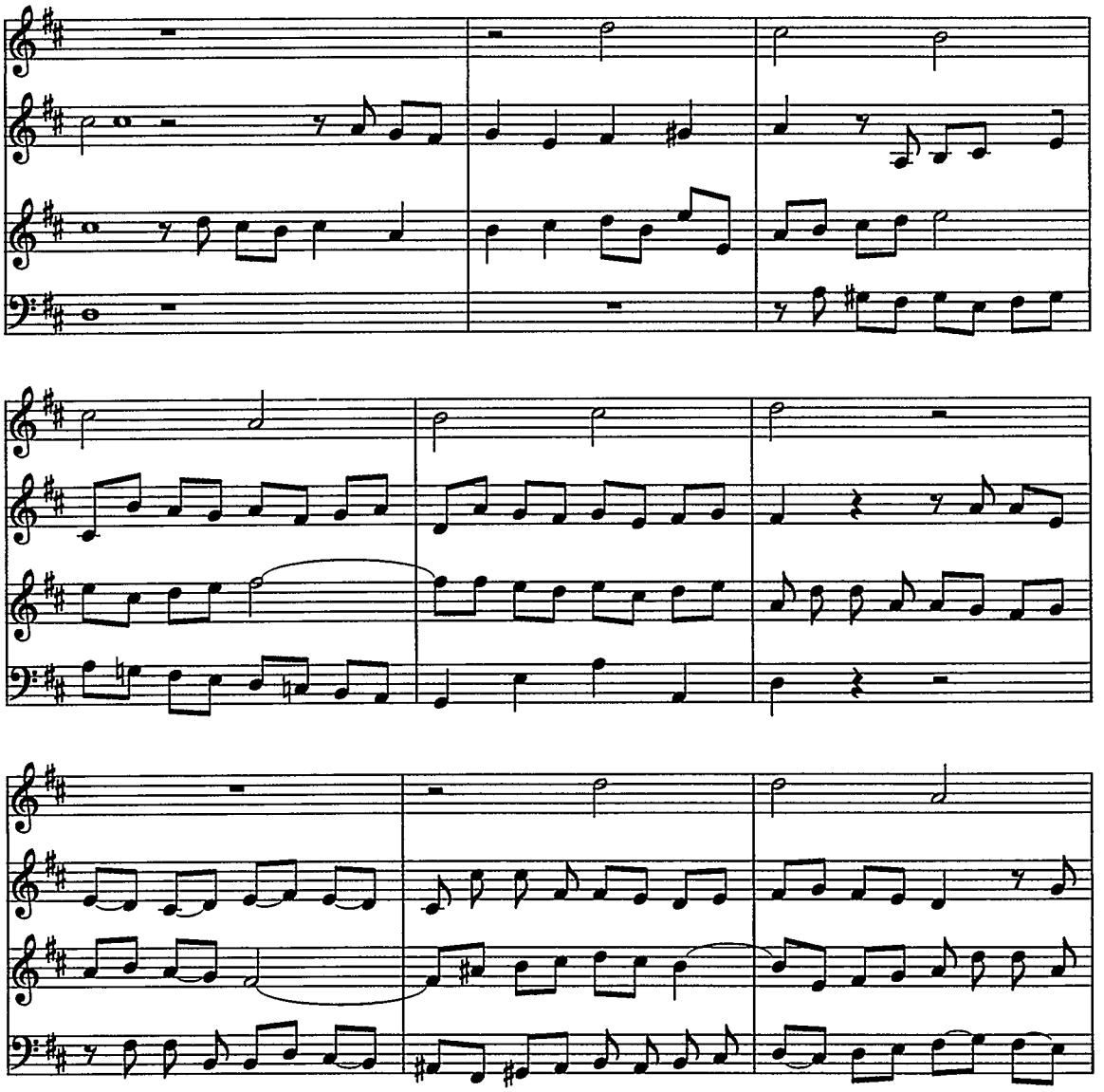


Figure B.2: Reconstructed score: *Magnificat (Vom Himmel hoch).*

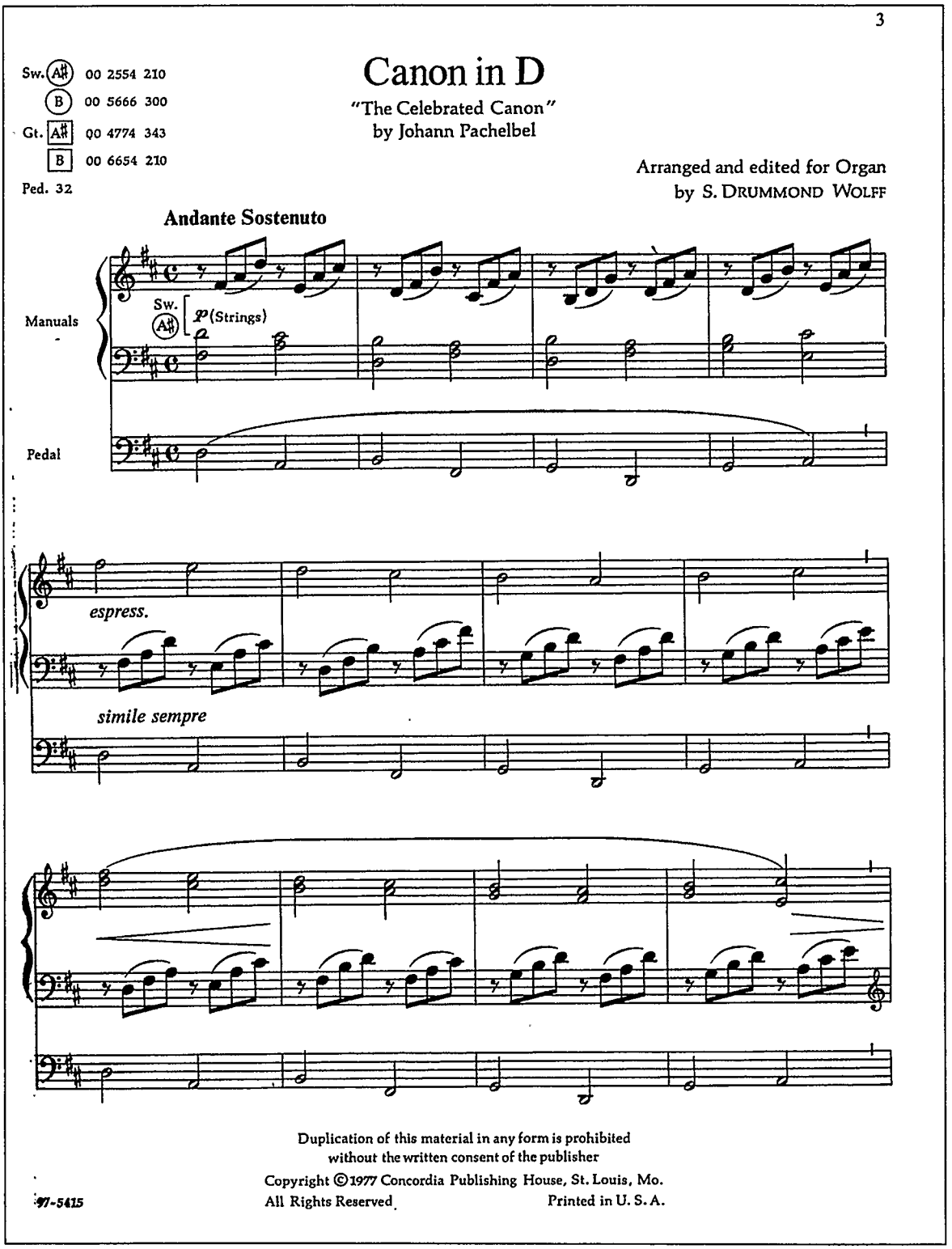


Figure B.3: Test score: *Canon in D.*

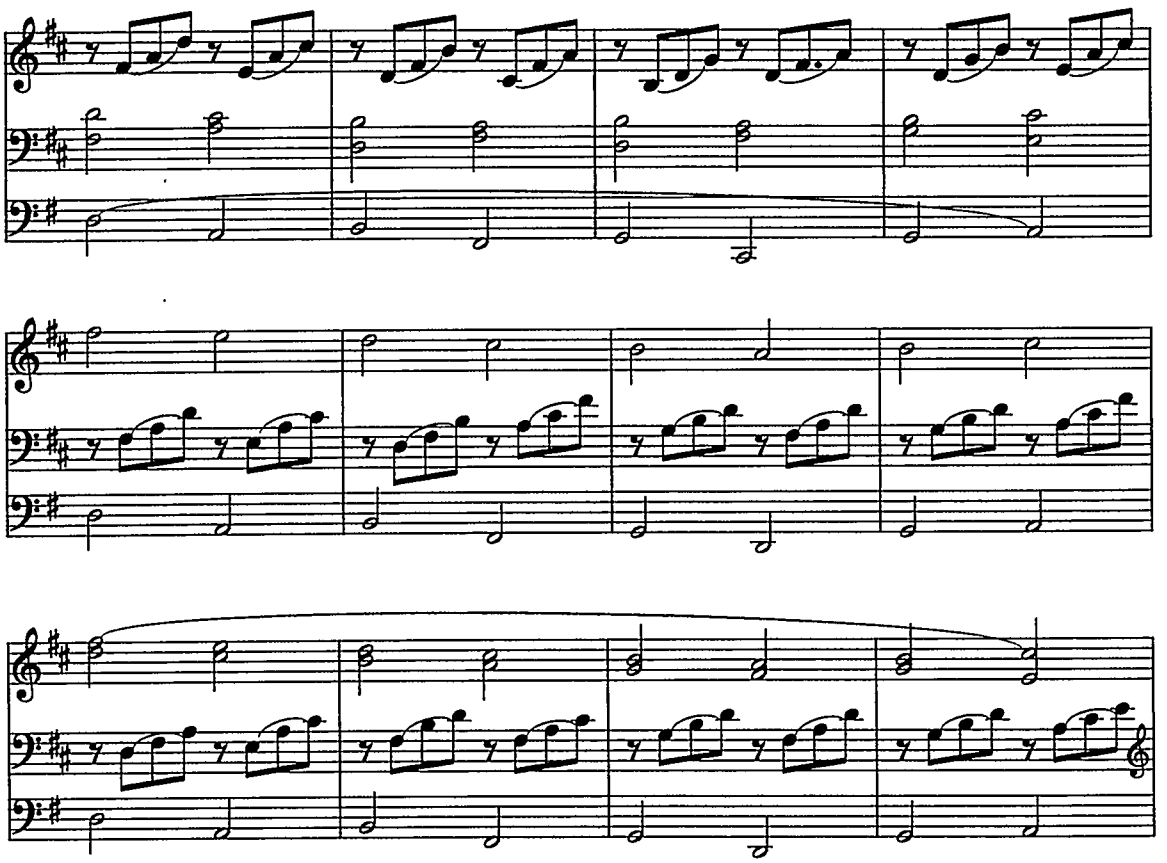


Figure B.4: Reconstructed score: *Canon in D.*

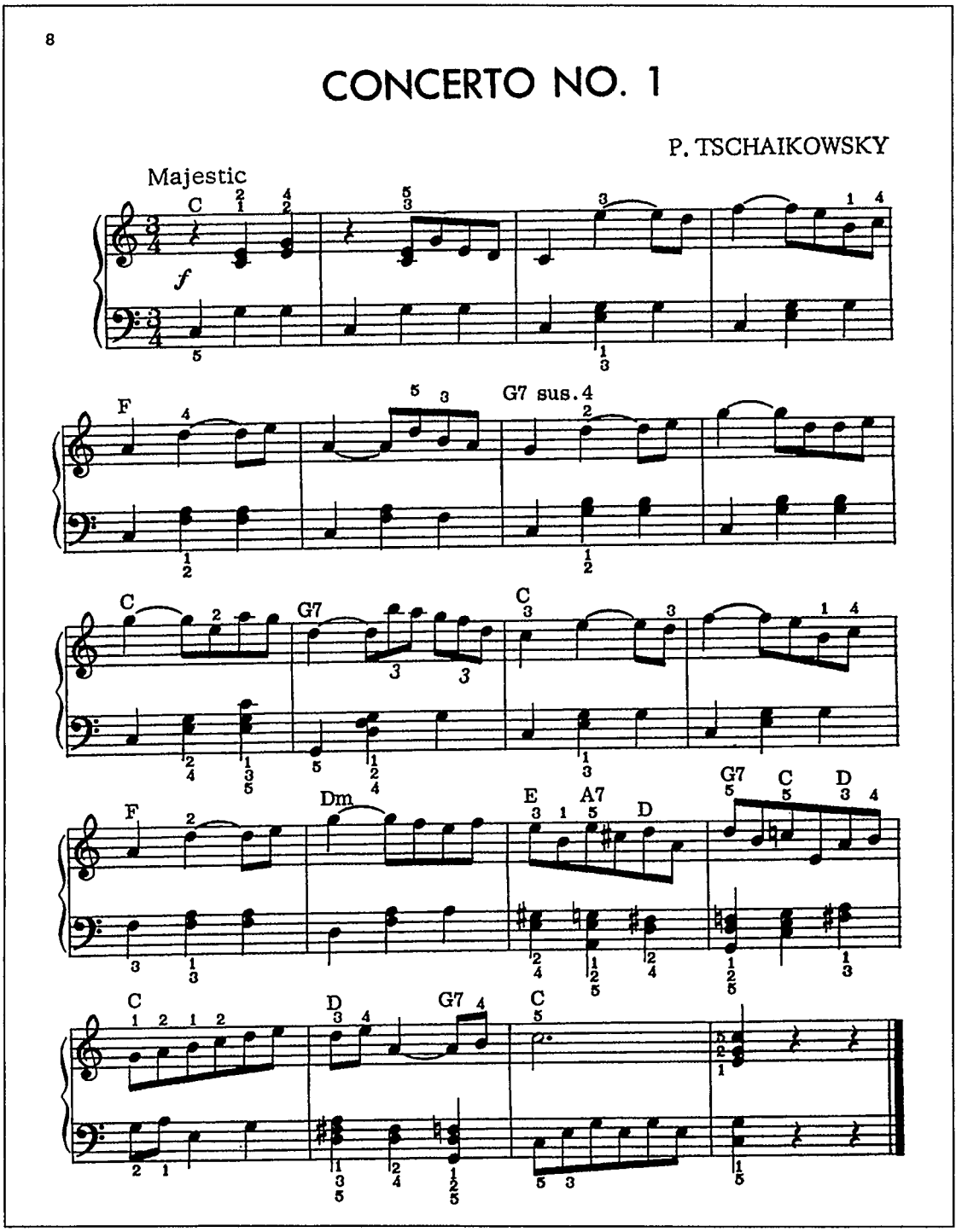


Figure B.5: Test score: *Concerto No. 1.*

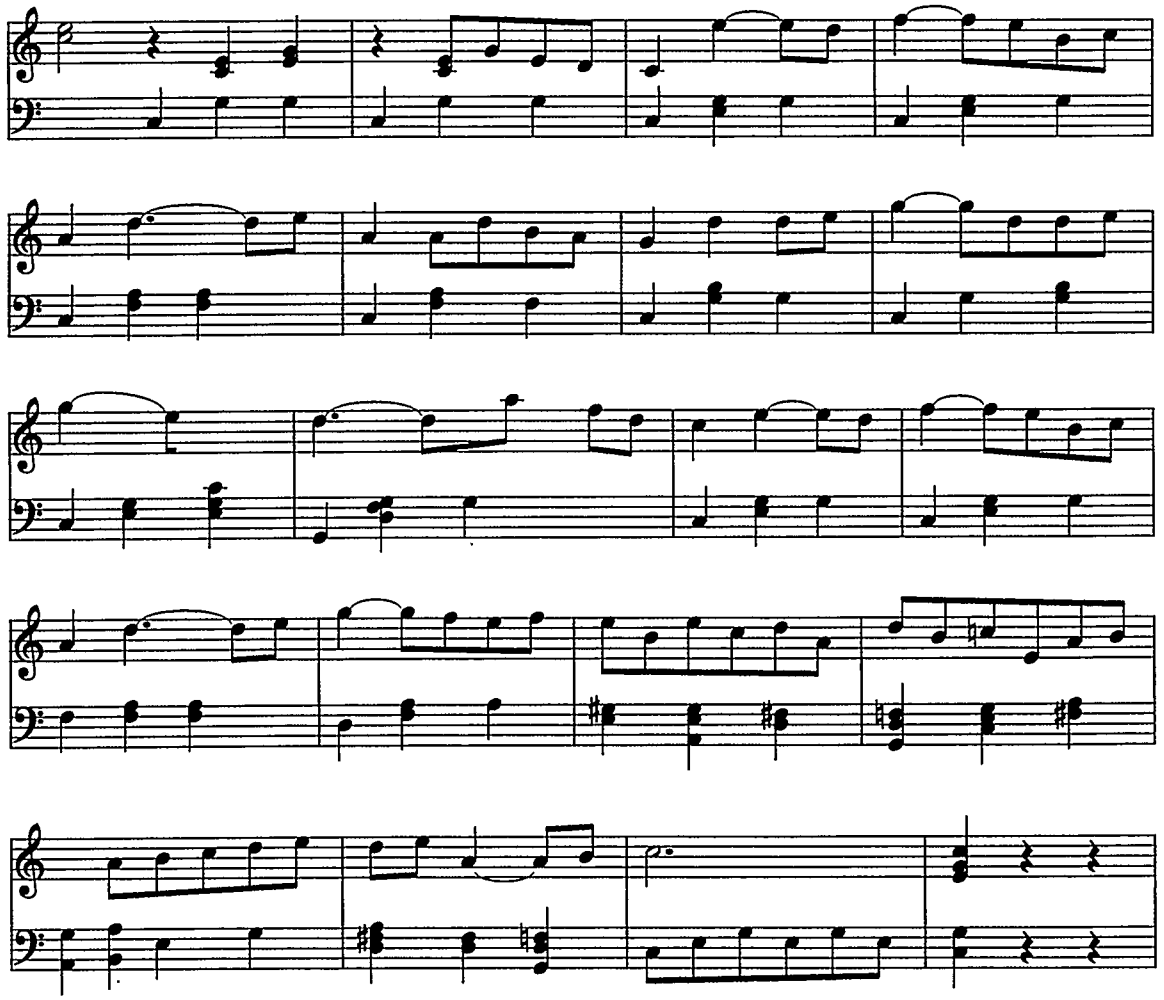


Figure B.6: Reconstructed score: *Concerto No. 1.*

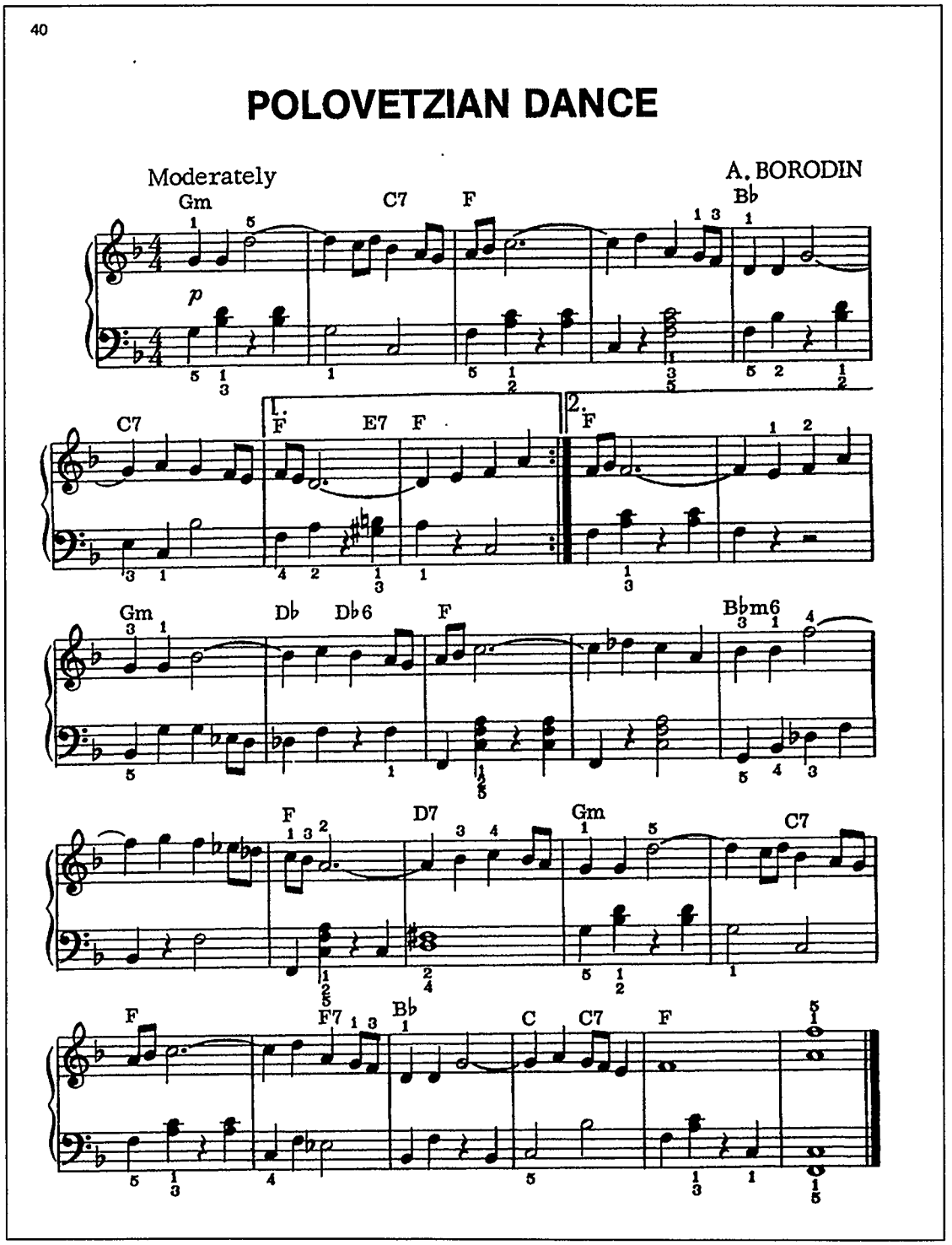


Figure B.7: Test score: *Polovetzian Dance.*

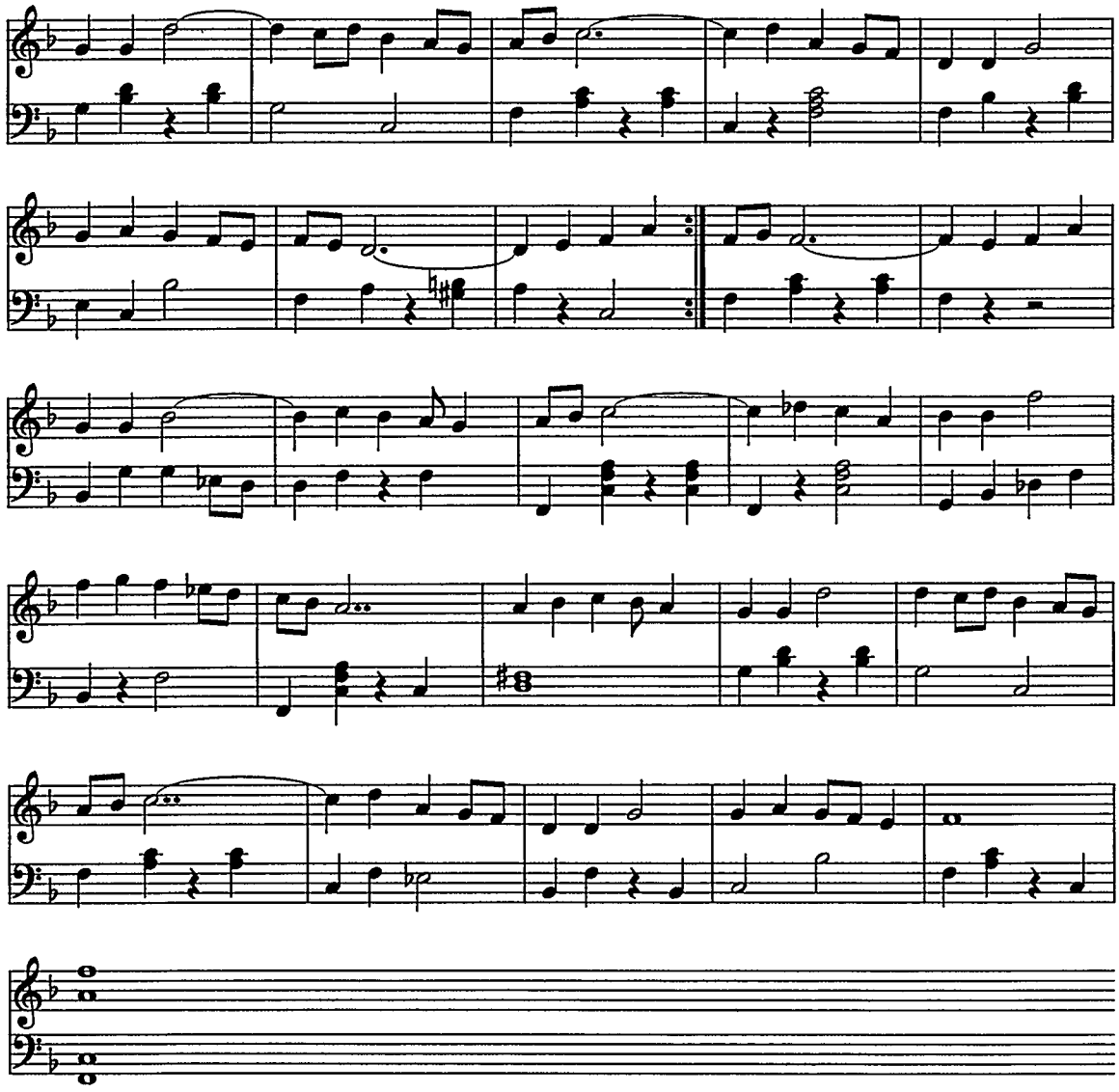


Figure B.8: Reconstructed score: *Polovetzian Dance.*

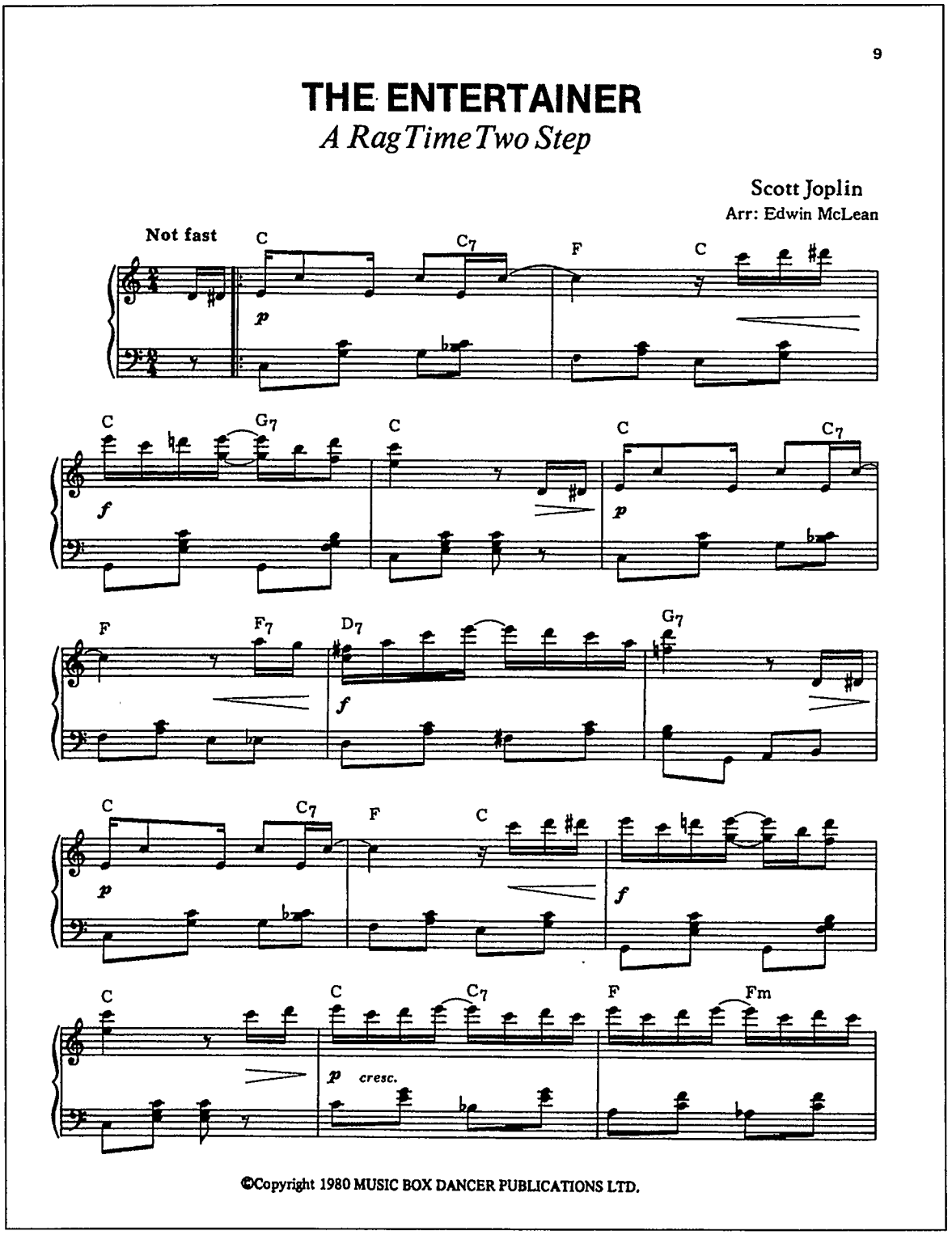


Figure B.9: Test score: *The Entertainer.*

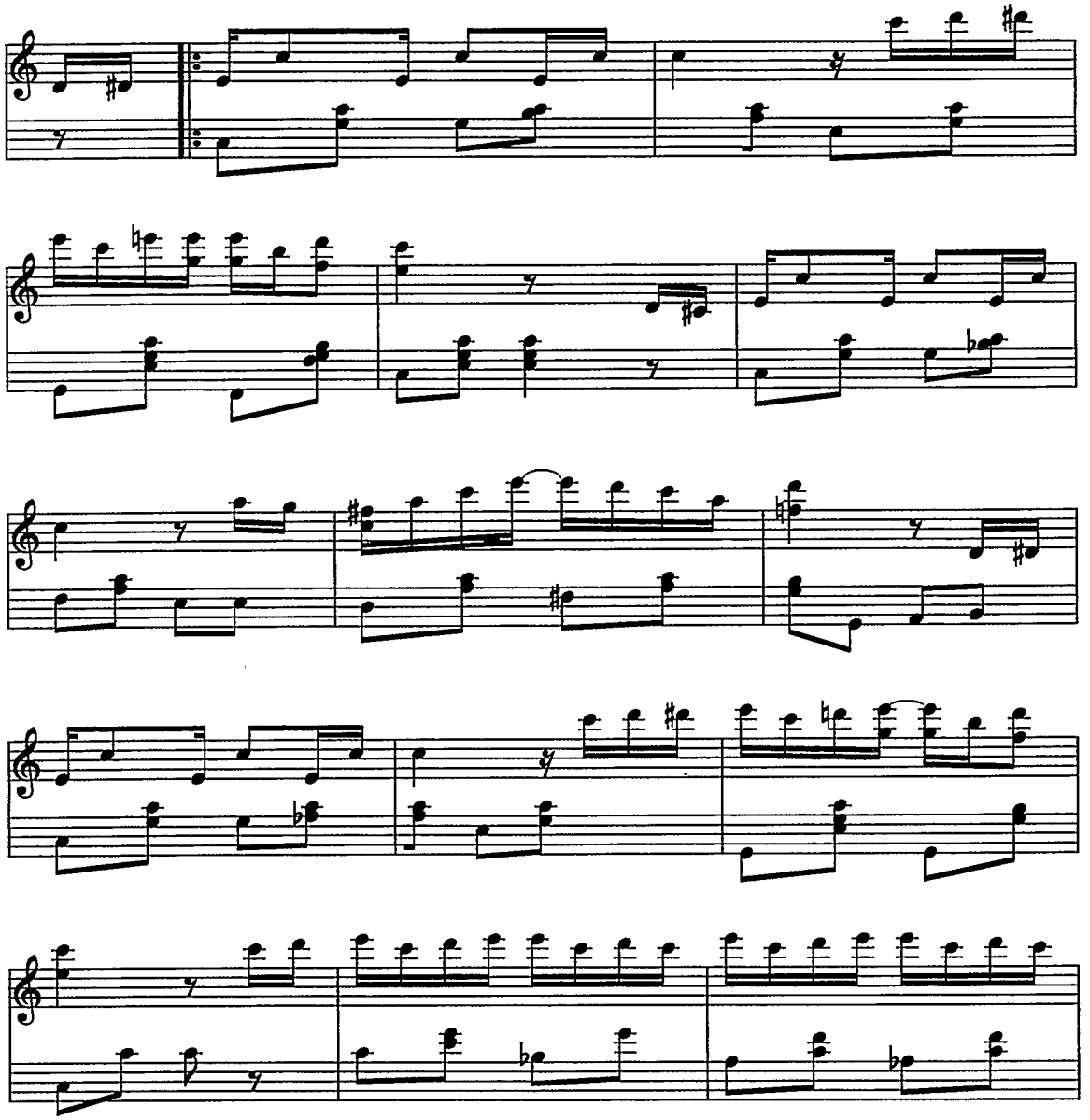


Figure B.10: Reconstructed score: *The Entertainer.*

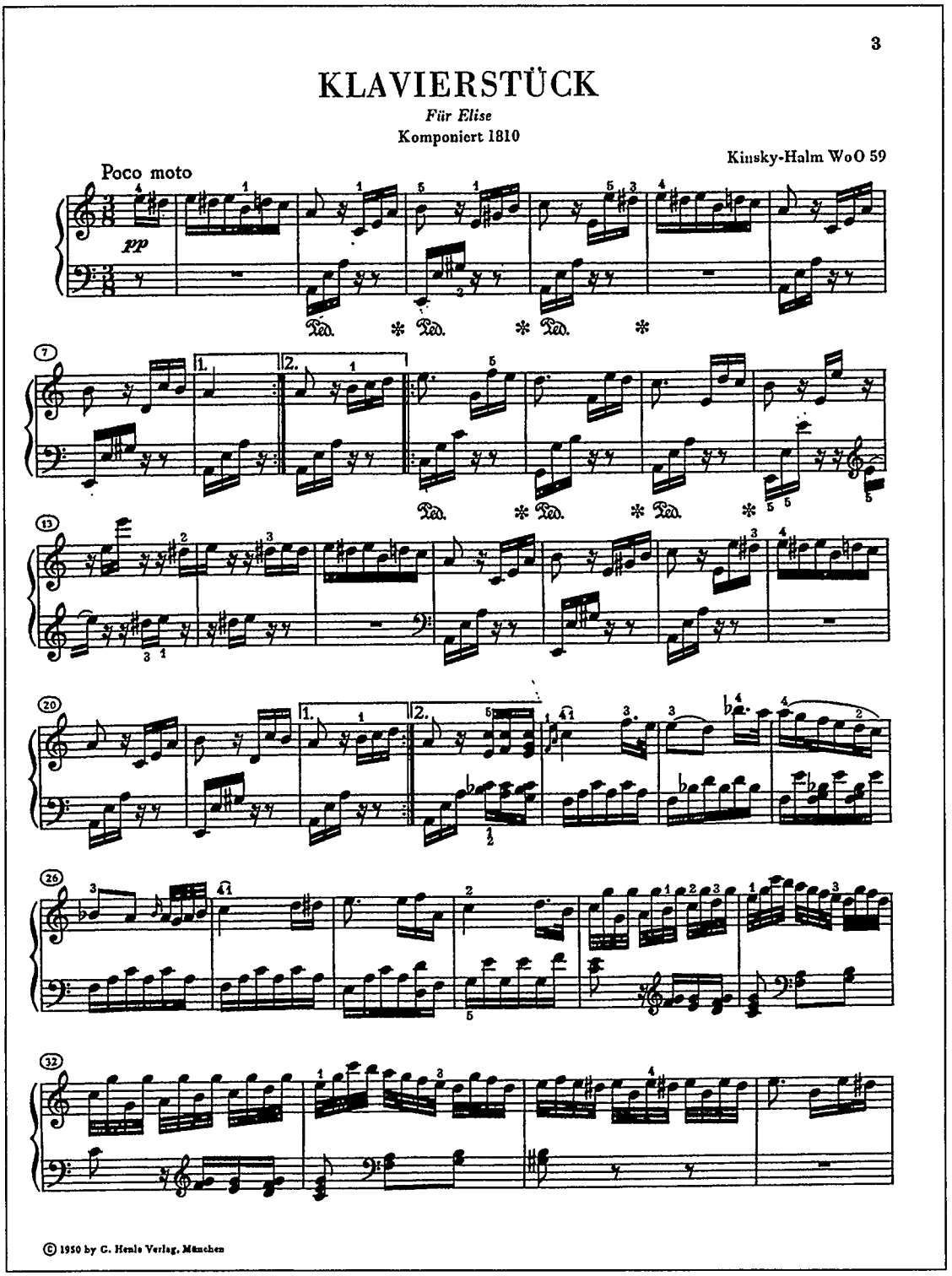


Figure B.11: Test score: *Klavierstück.*

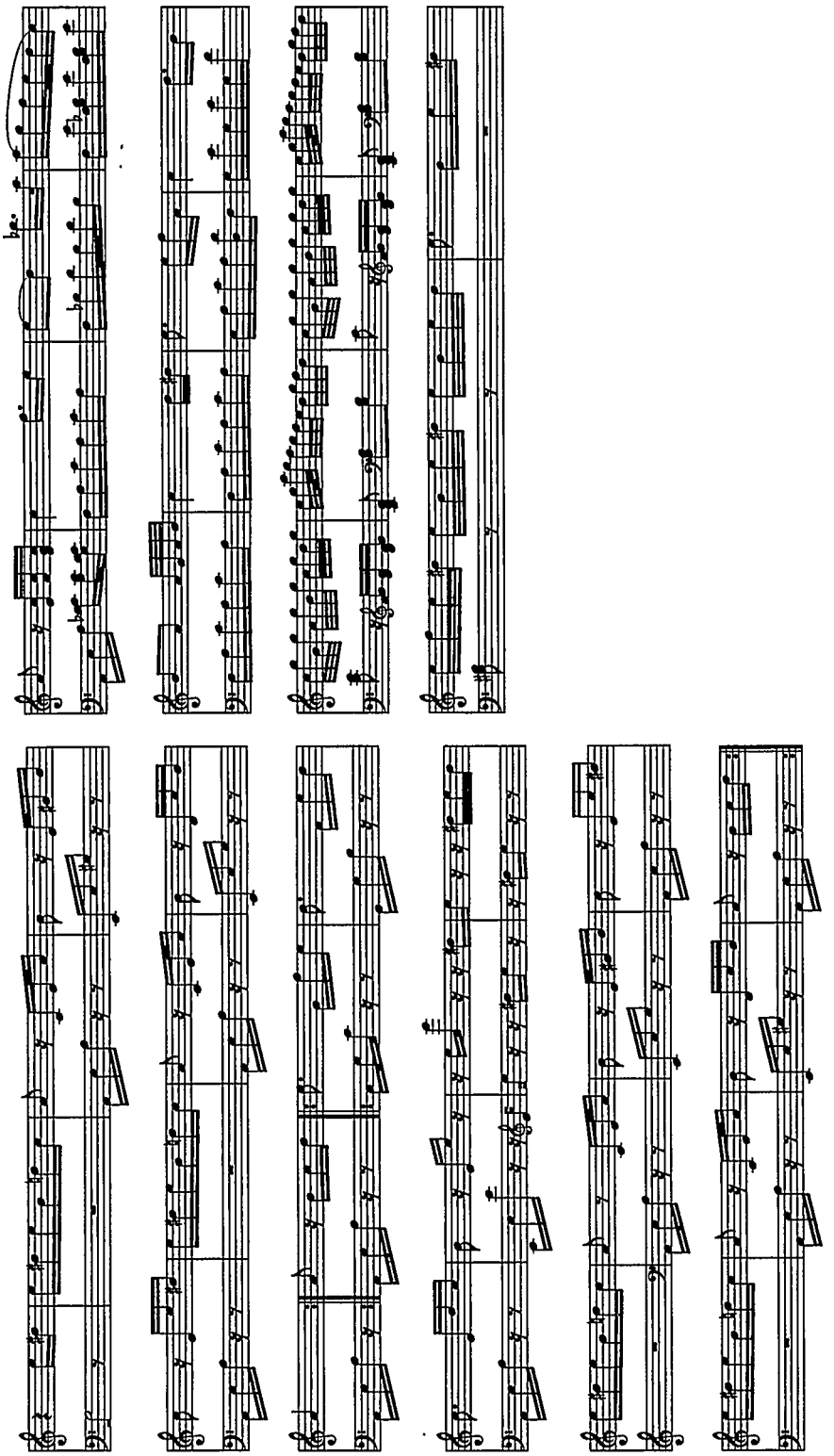


Figure B.12: Reconstructed score: *Klavierstück.*

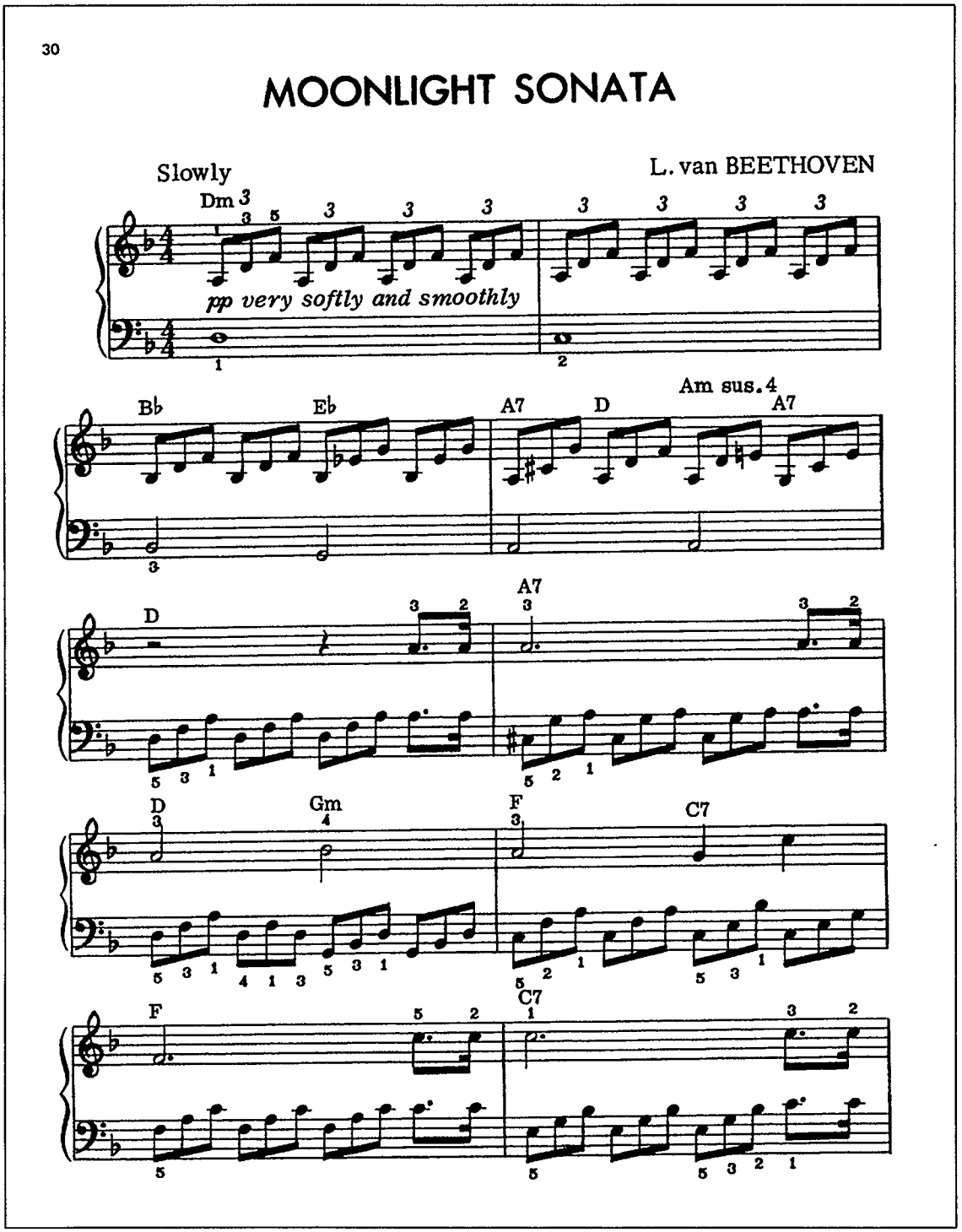


Figure B.13: Test score: *Moonlight Sonata.*

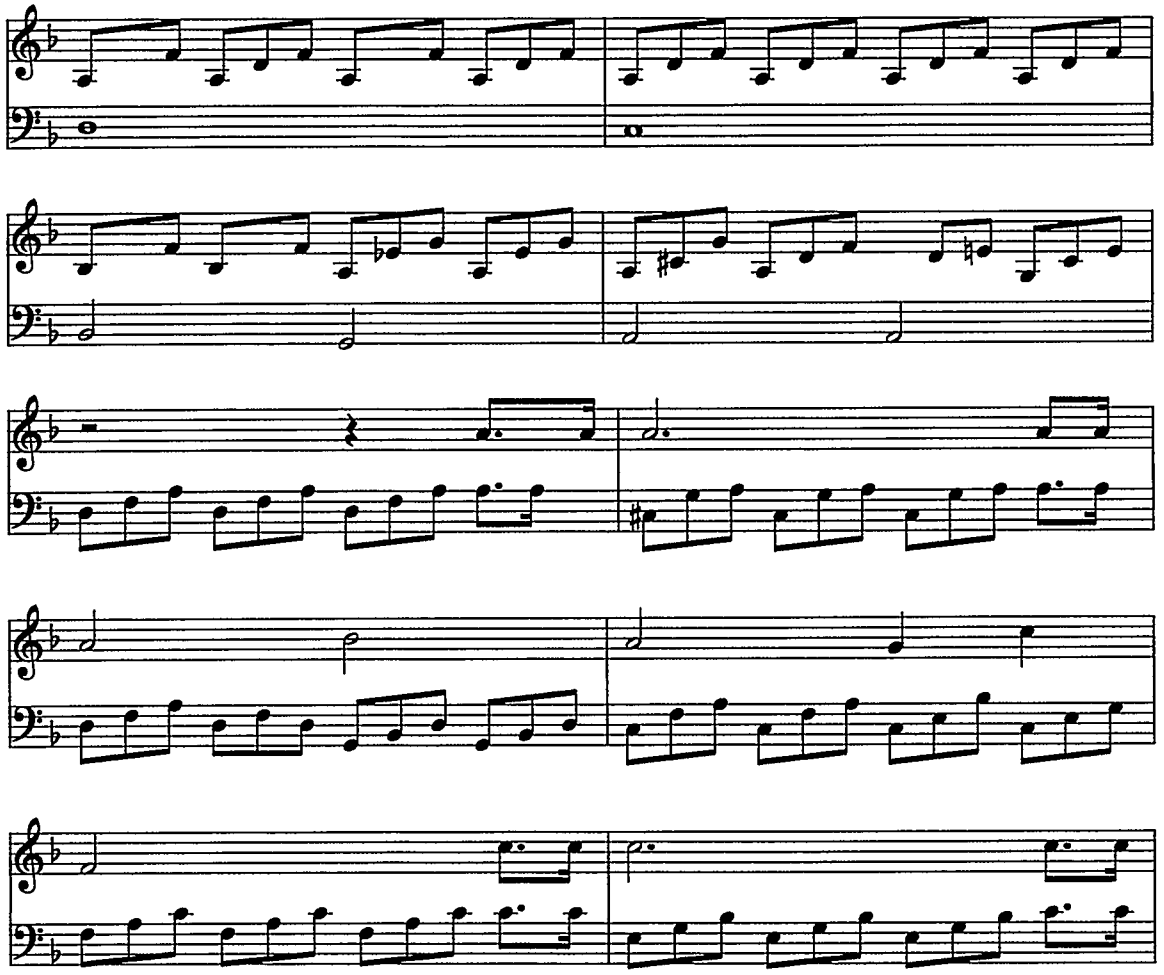


Figure B.14: Reconstructed score: *Moonlight Sonata.*

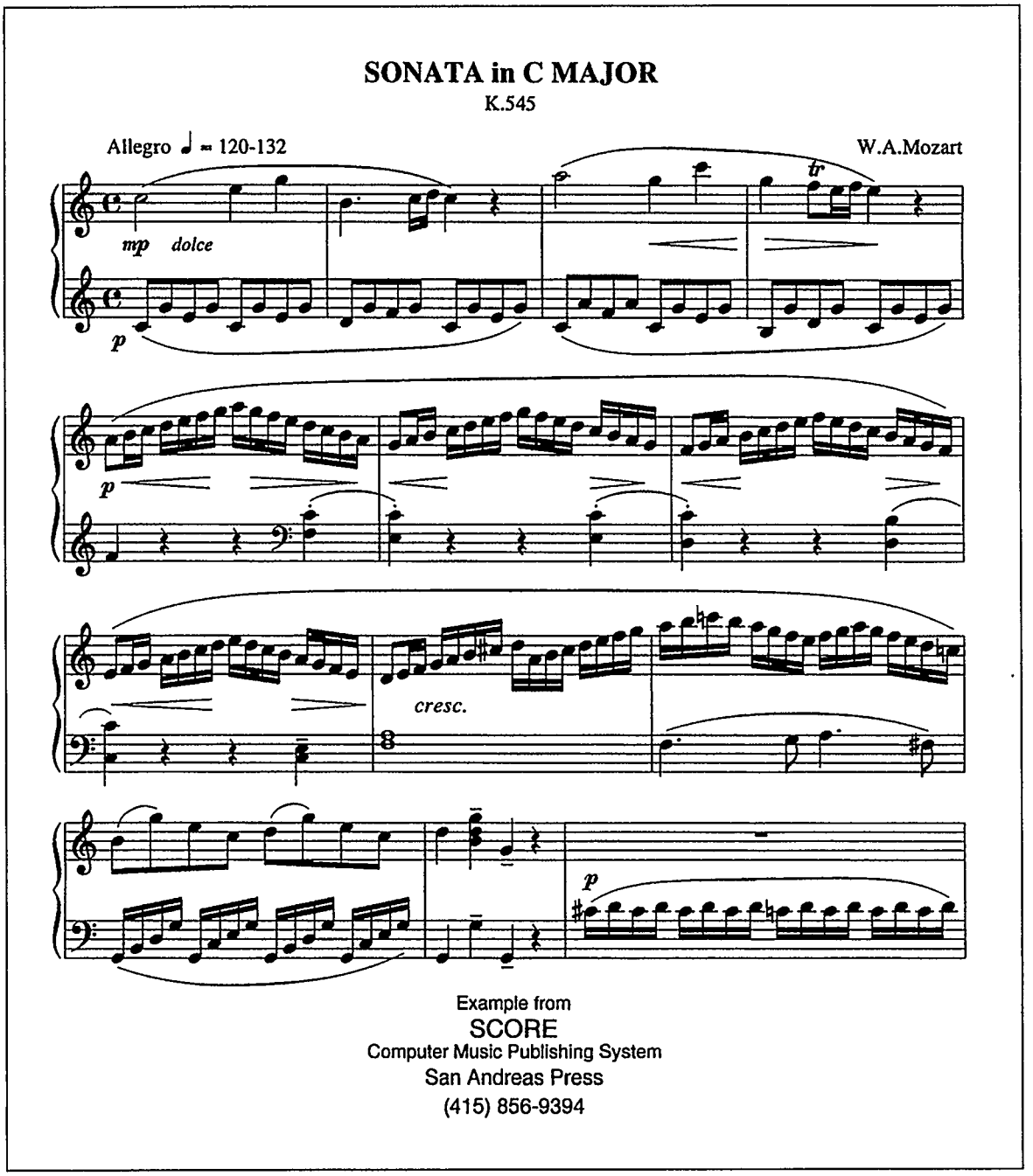


Figure B.15: Test score: *Sonata in C Major.*

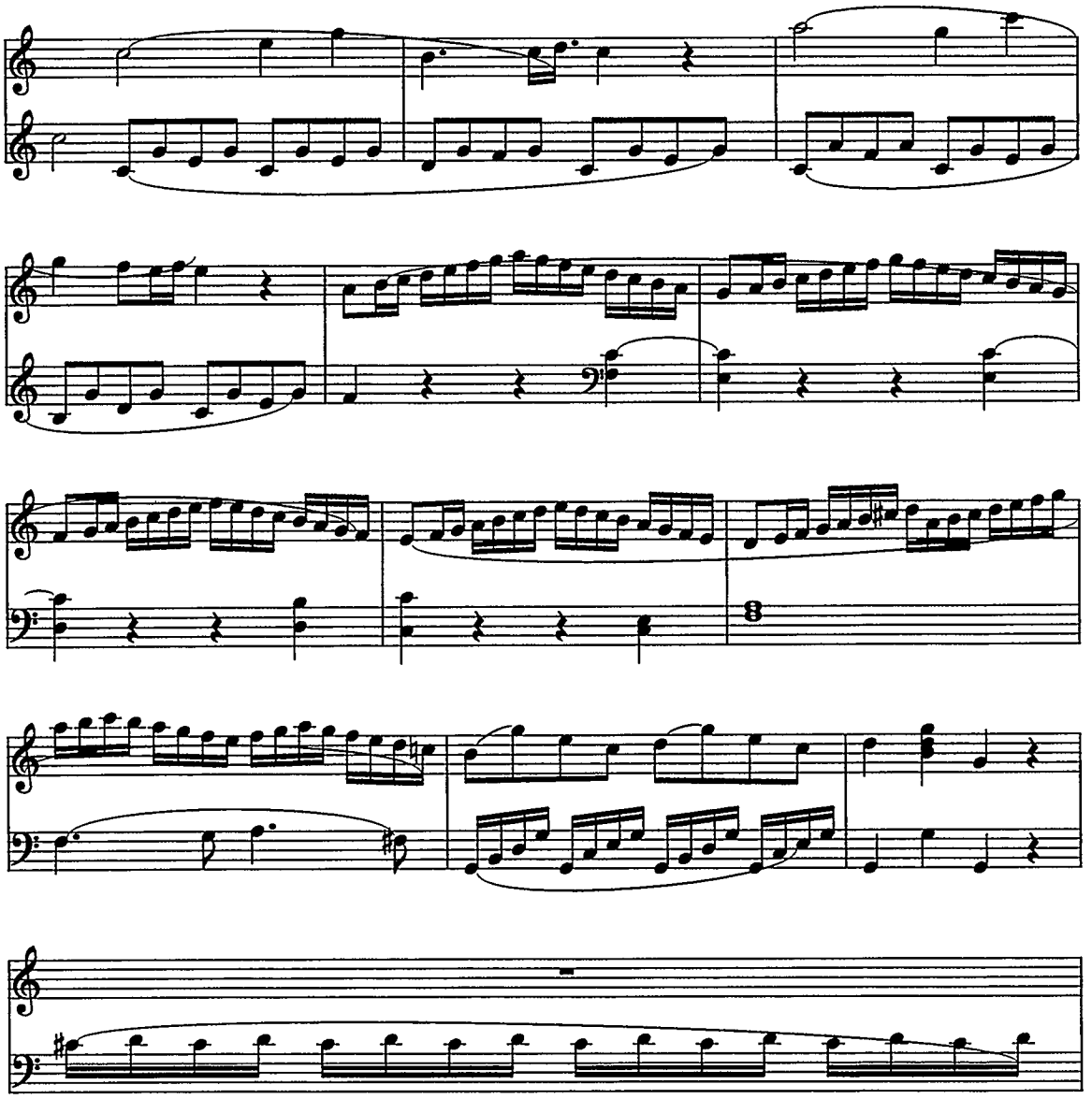


Figure B.16: Reconstructed score: *Sonata in C Major.*

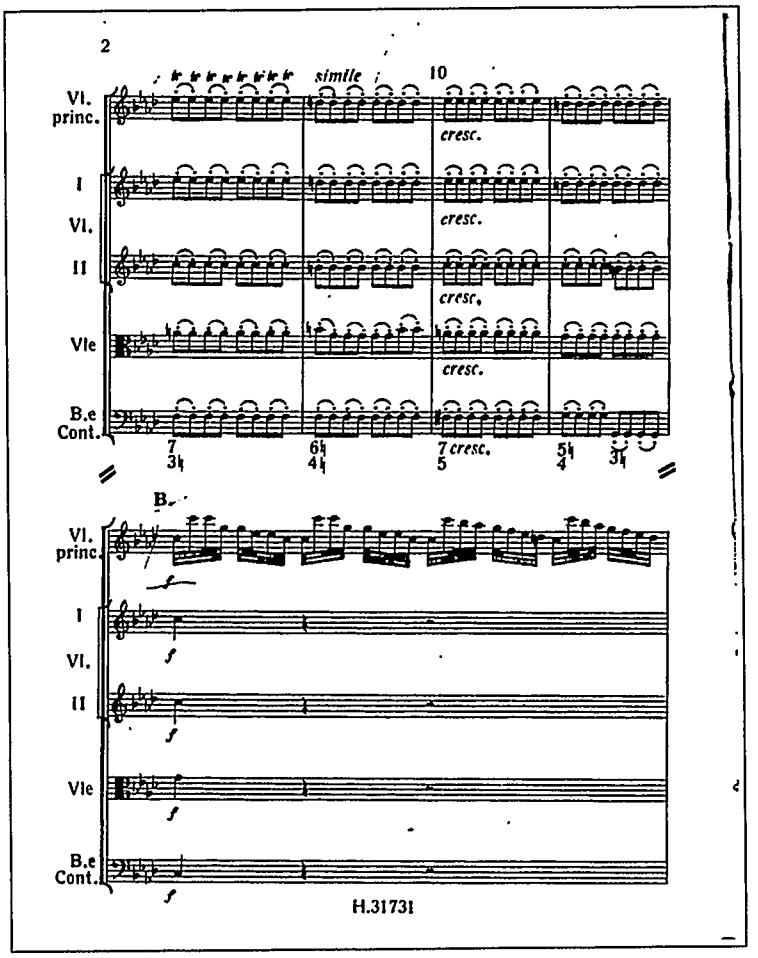


Figure B.17: Test score: *The Four Seasons (Winter).*

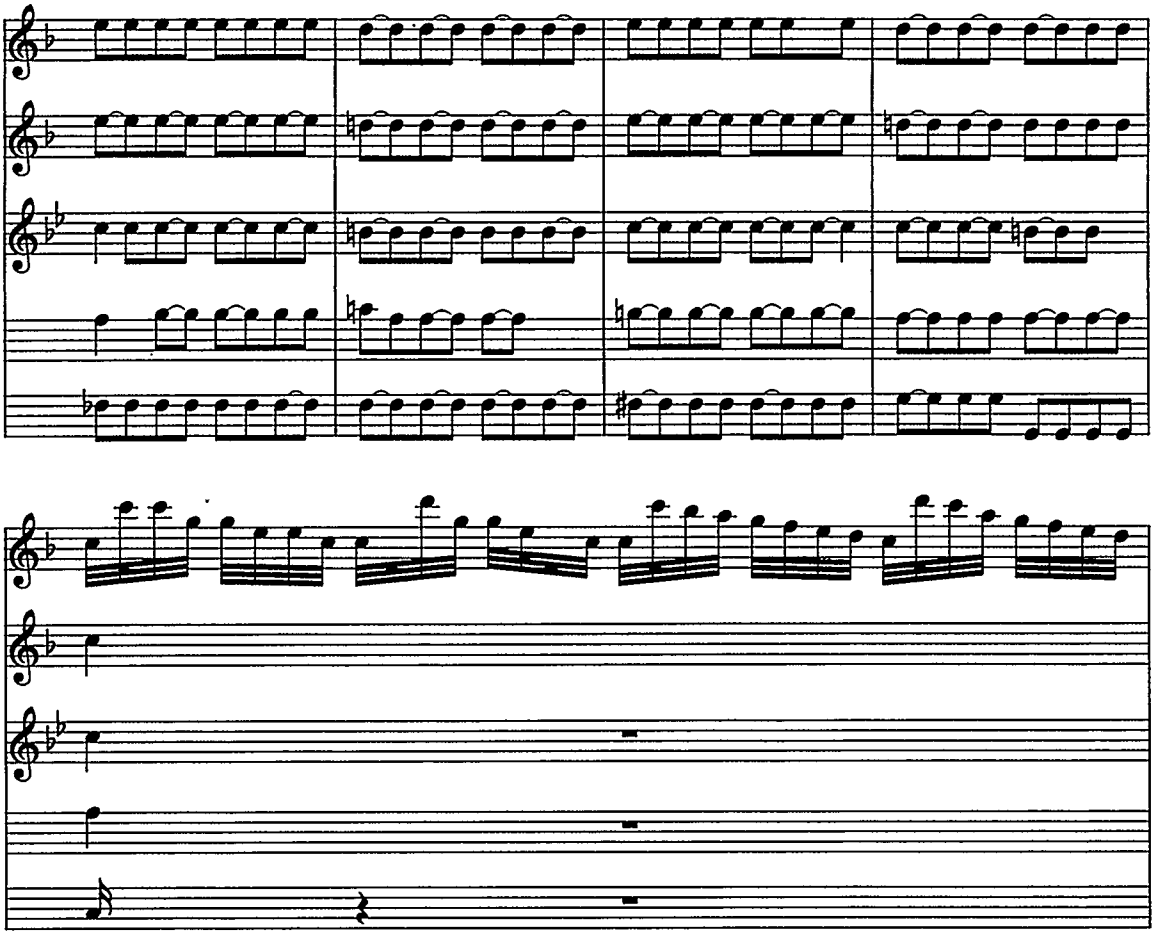


Figure B.18: Reconstructed score: *The Four Seasons (Winter).*

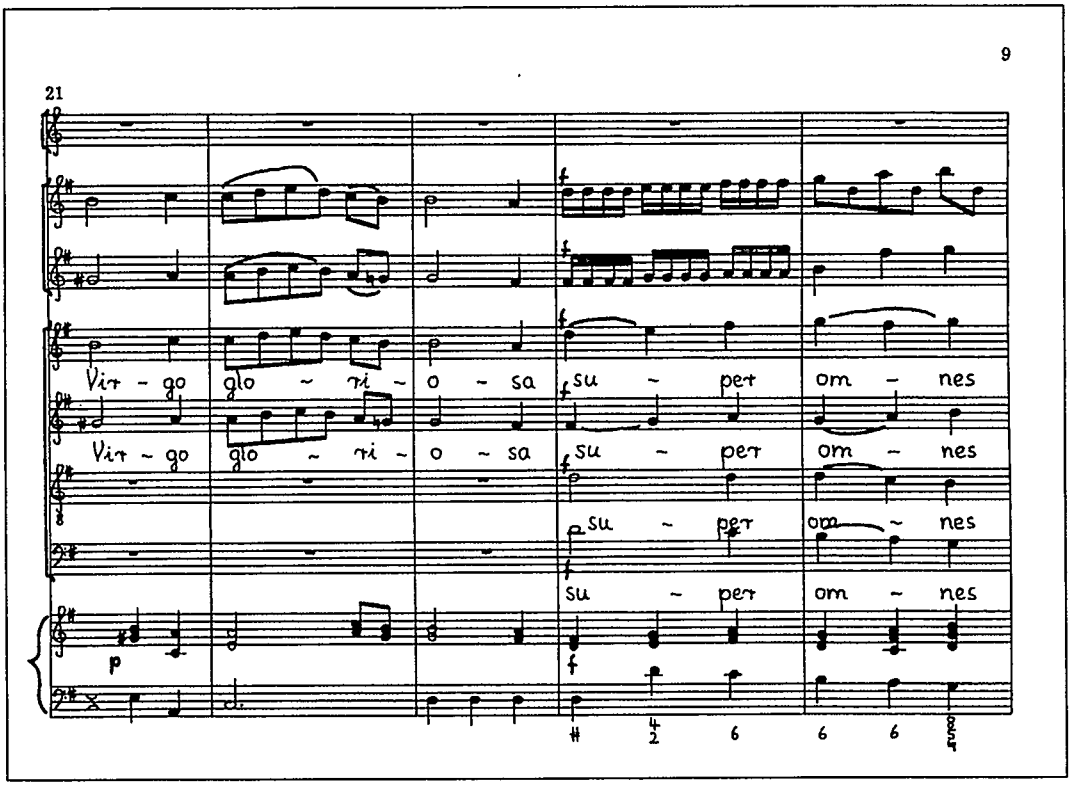


Figure B.19: Test score: *Antiphonae Marianae.*

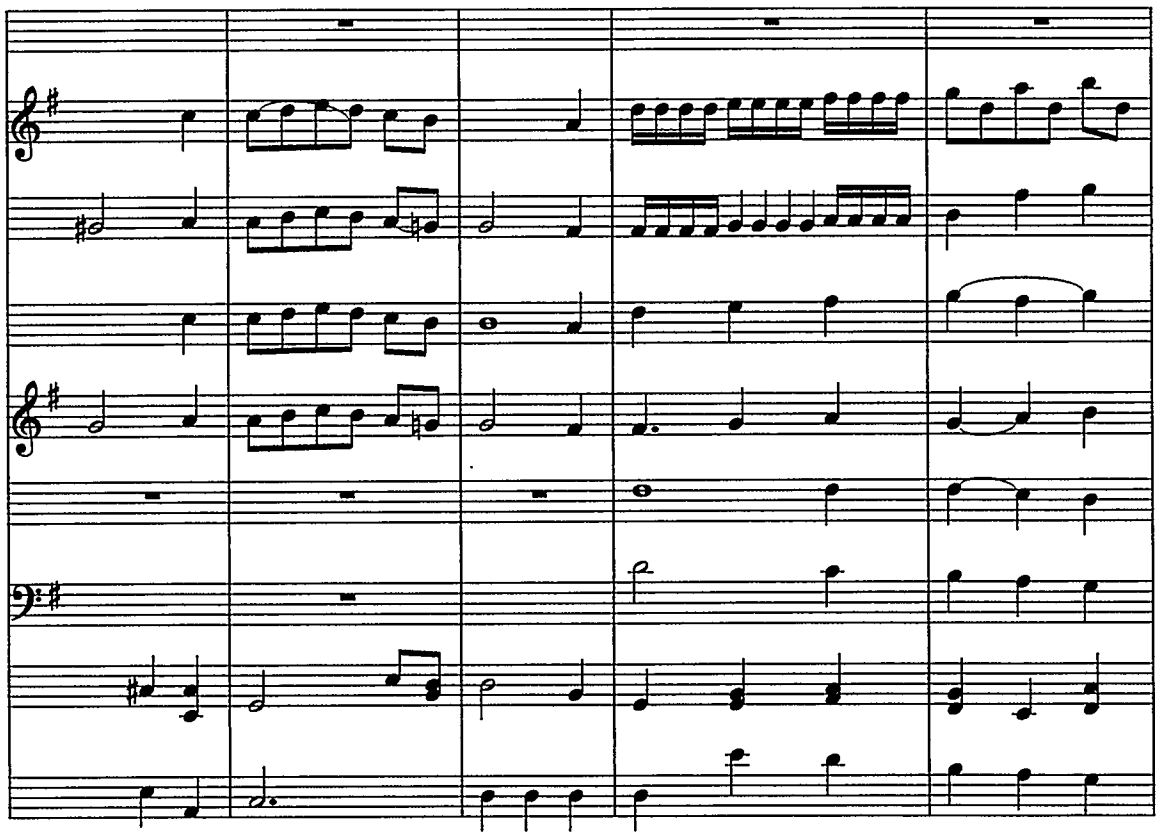


Figure B.20: Reconstructed score: *Antiphonae Marianae.*

B.2 DARMS Output

```
!I1 !G,!G,!G,!F !K2# !M*C \Q RW /
RH  7H  /
6  5  /
!-50 !K2# !M*C !R6:1 6 W $R RH RE 4E 3 2 /
3Q 1 2  3#  /
4  RE -3E -2( -1) \E 1;) /
!-50 !K2# !M*C !R4:1 6W $R RE 7E 6 5 6Q 4 /
5 6 7E 5 8 1 /
4 5 6 7 8H  /
!-50 !K2# !M*C !R4:1 5W $R RW /
 RW  /
RE 9E 8# 7 8 6 7 8 /
!+150 6H 4 /
5  6  /
7  RH  /
!-50 -1E 5 4 3 4 2 3 4 /
0 4 3 2 3 1 2 3 /
2Q  RQ  RE 4E 4 1 /
!-50 8 6 7 8 9HJ1 /
EJ2 9 8 7 8 6 7 8 /
4 7 7 4 4( 3) 2( 3) /
!-50 9 8* 7 6 5 4* 3 2 /
1Q  6  9  2  /
5  RQ  RH  /
!+150  RW /
RH  7H  /
7  4  /
!-50 1E(L1 0)L2 -1(L3 0)L4 1(L5 2)L6 1(L7 0)L8 /
-1 6 6 2 2( 1) 0( 1) /
2( 3) 2( 1) 0Q  RE 3E /
!-50 4( 5) 4(L9 3)L10 2HJ3 /
EJ4 4# 5 6 7 6 5QJ5  /
EJ6( 1) 2( 3) 4 7 7 4 /
!-50 RE 7 7 3 3( 5) 4(L11 3)L12 /
2#( 0) 1#( 2) 3 2 3 4 /
5(L13 4)L14 5( 6) 7(L15 8)L16 7(L17 6)L18 /
```

Figure B.21: DARMS output: *Magnificat (Vom Himmel hoch).*

```
!I1 !G,!F,!F !K2# !M*C RE 2E(L1 4 7)L2 RE 1(L3 4 6)L4 /
RE 0(L5 2 5)L6 RE -1(L7 2 4)L8 /
RE -2(L9 0 3)L10 RE 0(L11 2. 4)L12 /
RE 0(L13 3 5)L14 RE 1(L15 4 6)L16 /
!-50 !K2# !M*C 7H,12 9,11 /
5,10  9,7  /
5,10  9,7  /
10,8  11,6  /
!-50 !K1# !M*C 5L17 2 /
3  0  /
1  -3  /
1  2L18  /
!+100 9 8 /
7  6  /
5  4  /
5  6  /
!-50 RE 7E(L19 9 12)L20 RE 6(L21 9 11)L22 /
RE 5(L23 7 10)L24 RE 9(L25 11 14)L26 /
RE 8(L27 10 12)L28 RE 7(L29 9 12)L30 /
RE 8(L31 10 12)L32 RE 9(L33 11 14)L34 /
!-50 5H 2 /
3  0  /
1  -2  /
1  2  /
!+100 9L35,7 8,6 /
7,5  6,4  /
5,3  4,2  /
5,3  1,6L36  /
!-50 RE 5E(L37 7 9)L38 RE 6(L39 9 11)L40 /
RE 7(L41 10 12)L42 RE 7(L43 9 11)L44 /
RE 8(L45 10 12)L46 RE 7(L47 9 12)L48 /
RE 8(L49 10 12)L50 RE 9(L51 11 13)L52 !G /
!-50 5H 2 /
3  0  /
1  -2  /
1  2  /
```

Figure B.22: DARMS output: *Canon in D.*

```
!I1 !G,!F !M*3:4 8H,6 RQ -1Q,1 1,3 /
RQ -1E(,1 3 1 0) /
-1Q 8J1 EJ2( 7) /
9QJ3 EJ4( 8 5 6) /
!-50 !M*3:4 [\H] 4Q 8 8 /
4 8 8 /
4 6,8 8 /
4 6,8 8 /
!+50 4 7.J5 EJ6( 8) /
4Q E( 7 5 4) /
3Q 7 E( 8) /
10QJ7 EJ8( 7 7 8) /
!-50 4Q 7,9 7,9 /
4 7,9 7 /
4 8,10 8 /
4 8 8,10 /
!+50 10L1 [\E] 8E;(L2 [\E] [\E] /
7Q.J9 EJ10( [\E] 11) [\E] 9( 7) /
6Q 8J11 EJ12( 7) /
9QJ13 EJ14( 8 5 6) /
!-50 4Q 6,8 11,6,8 /
1 5,7,8 8 /
4 6,8 8 /
4 6,8 8 /
!+50 4 7.J15 EJ16( 8) /
10QJ17 EJ18( 9 8 9) /
8( 5 8 6 7 4) /
7( 5 6* 1 4 5) /
!-50 7Q 7,9 7,9 /
5 7,9 9 /
6,8# 2,8,6 5,7# /
1,7*,5 4,6,8 7#,9 /
!+50 [\E] 4E( 5 6 7 8) /
7( 8) 4QJ19 EJ20( 5) /
6H. /
1Q,3,6 RQ RQ /
!-50 [!R2:1] 2,8 9,3 [$R] 6Q 8 /
5,9,7# 5,7 1,7*,5 /
4E( 6 8 6 8 6) /
4Q,8 RQ RQ /
```

Figure B.23: DARMS output: *Concerto No. 1.*

```
!I1 !G,!F !K1- !M*4:4 3Q 3 7HJ1 /
QJ2 6E( 7) 5Q 4E( 3) /
4( 5) 6H.J3  /
QJ4 7 4 3E( 2) /
0Q 0 3H /
!-50 !K1- !M*4:4 8Q 10,12 RQ 10,12 /
8H 4 /
7Q 9,11 RQ 9,11 /
4 RQ 11H,9,7 /
Q 10 RQ 10;12 /
!+50 3 4 3 2E( 1) /
2( 1) 0H.J5  /
QJ6 1 2 4 :/
2E( 3) 2H.J7  /
QJ8 1 2 4 /
!-50 6 4 10H /
7Q 9 RQ 8#,10* /
9 RQ 4H :/
7Q 9,11 RQ 9,11 /
7 RQ RH /
!+50 3 3 5HJ9 /
QJ10 6 5 4E 3Q /
4E( 5) 6HJ11 \Q /
QJ12 7- 6 4 /
5 5 9H /
!-50 3Q 8 8 6-E( 5) /
Q 7 RQ 7 \E /
0 7,9,4 RQ 7,9,4 /
0 RQ 9H,7,4 /
1Q 3 5- 7 /
!+50 9 10 9 8-E( 7) /
6( 5) 4H.. /
Q 5 6 5E 4Q /
3 3 7H /
Q 6E( 7) 5Q 4E( 3) /
!-50 Q RQ 7H /
0Q 9,7,4 RQ 4 /
5W,7#  /
8Q 10,12 RQ 10,12 /
8H  4 /
!+50 E( 5) 6H..J13 /
QJ14 7 4 3E( 2) /
0Q 0 3H /
Q 4 3E( 2) 1Q /
2W  /
4,9 /|
!-50 7Q 9,11 RQ 9,11 /
4 7 6-H /
3Q 7 RQ 3 /
4H 10 /
7Q 9,11 RQ 4 /
0W,4 /|
```

Figure B.24: DARMS output: *Polovetzian Dance.*

```
!I1 !G,!T !M*2:4  0S(( 0#)) /:
1(;( 6E  1S);) 6E( 1S( 6)) /
Q  RS 13S(( 14 14#))  /
!-50 !M*2:4 RE /:
4E(  11),8  8( 11),10  /
\E 11;),9  6(  11),8  /
!+50 15S(( 13 15* 15)),10 15((,10 12) 9E),14 /
13Q,8  RE \E 0S(( -1#)) /
1(;( 6E  1S);) 6E( 1S( 6)) /
!-50 1E( 11),8,6 0( 7),10,8 /
4( 11),8,6  11Q,6,8 RE  /
4E(  11),8  8( 11),10- /
!+50 6Q RE 11S(( 10)) /
9#((,6 11 13 15J1)) 15J2(( 14 13 11)) /
9*Q,14  RE 0S(( 0#)) /
!-50 7E( 9),11 6( 6) /
5(  9),11  7#(  11),9  /
10(,8  1) 2( 3)  /
!+50 1S(;( 6E 1S);) 6E( 1S( 6)) /
Q  RS 13S(( 14 14#)) /
15(( 13 14* 15J3)),10 15J4((,10 12) 9E),14 /
!-50 4( 11),8 8( 11),9- /
 9;),11 6(  11),8  /
1(  11),8,6  1(  8),10 /
!+50 13Q,8 RE 13S(( 14)) /
15(( 13 14 15)) 15(( 13 14 13)) /
15(( 13 14 15)) 15(( 13 14 13)) /
!-50 4E( 11) 11 RE /
11(  15),13  10-(  15)  /
9(  11),14  9-(  11),14  /
```

Figure B.25: DARMS output: *The Entertainer.*

```
!I1 !G,!F !M*3:8 RQ 8S(( 7#)) /
8(( 7# 8 5 7* 6)) /
4E RS -1S(( 1 4)) /
5E RS 1S(( 3# 5)) /
6E RS 1S(( 8 7#)) /
8(( 7# 8 5 7* 6)) /
4E RS -1S(( 1 4)) /
!-50 !M*3:8 !R2:1 8H;) $R RE /
 RW  /
2S(( 6 9)) RS RE /
-1(( 6 8#)) RS RE /
2(( 6 9)) RS RE /
 RW  /
2(( 6 9)) RS RE /
!+50 5E RS 0S(( 6 5)) /
4Q :/
E RS 5S(( 6 7)) /:
8E. 3S(( 9 8)) /
7E. 2S(( 8 7)) /
6E. 1S(( 7 6)) /
5E RS 1S(( 8)) RS /
!-50 -1(( 6 8)) RS RE /
2(( 6 9)) RS :/
2(( 6 9)) RS RE /:
4(( 8 11)) RS RE /
1(( 8 10)) RS RE /
2(( 6 9)) RS RE /
-1(( 6 13)) RS RS !G 1;(( /
!+50 RS 8(( 15)) RS RS 7#(( /
8)) RS RS 7#(( 8 7)) /
8(( 7# 8 5 7* 6)) /
4E RE -1S(( 1 4)) /
5E RS 1S(( 3# 5)) /
6E RS 1S(( 8 7#)) /
8(( 7# 8 5 7* 6)) /
!-50 8;)) RS RS 7#(( 8)) RS /
RS 7#(( 8)) RS RE /
RW !F /
2(( 6 9)) RS RE /
-1(( 6 8)) RS RE /
2(( 6 9)) RS RE /
RW /
!+50 4E RS -1S(( 1 4)) /
5E RS 0S(( 6 5)) /
4E RS 5S(( 6 7)) :/
4E RS 1S((,6 6,2 1)),3,6 /
Q 9S.(( 8)) /
E(L1 7)L2 12-S.(( 11T));) /
S((L3 10 9 8 7 6))L4 /
!-50 2(( 6 9)) RS RE /
-1(( 6 8#)) RS RE /
2(( 6 9)) RS RE :/
2(( 6 9)) 11((,10- 11,9 11)),10,8 /
7(( 9 11 9 11 9)) /
7(( 10- 12 10 12 10)) /
7(( 13 7,8,10- 13 7,8,10 13)) /
!+50 5E( 4) T((( 3 4 5))) /
6Q  7S(( 7#)) /
8E. S(( 9 4)) /
6Q  7S.(( 5)) /
6T((( 10 3 10))) 4((( 10 5 10)))
   6((( 10 7 10))) /
8((( 10 13 12))) 11((( 10 9 8)))
   7((( 10 9 7))) /
!-50 S(( 9 11 9 11 9)) /
7(( 9 11 9 11 9)) /
6(( 9 11 9 5,12 7)) /
8(( 13 8 13 8 14)) /
11E,13 RS !G 3S((,2 3,1 3)),0,2 /
-1E,3,1 !F 9(,7 8),10 /
!+50 6T((( 10 3 10))) 4((( 10 5 10)))
   6((( 10 7 10))) /
8((( 10 13 12))) 11((( 10 9 8)))
   7((( 10 9 7))) /
8((( 9 8 7#))) 8((( 5 8 7#)))
   8((( 5 8 7))) /
8E. 5S(( 8 7#)) /
!-50 11E RS K Parse error $
!G 3S((,2 3,1 3)),0,2 /
-1E,3,1 !F 7(,9  10),8  /
8#,10  RE  RE  /
 RW  /
```

Figure B.26: DARMS output: *Klavierstück.*

```
!I1 !G,!F !K1- !M*4:4 !R3:2 -3E( \E 2) -3( 0 2) -3( \E 2) -3( 0 2) /
-3( 0 2) -3( 0 2) -3( 0 2) -3( 0 2) $R /
!-50 !K1- !M*4:4 5W /
4 /
!+50 !R3:2 -2E( \E 2) -2( \E 2) -3( 1- 3) -3( 1 3) /
-3( -1# 3) -3( 0 2) \E 0( 1*) -4( -1 1) $R /
!-50 3H 1 /
2 2 /
!+50 RH RQ 4E.( S);) /
H. E( S);) /
!-50 !R3:2 5E( 7 9) 5( 7 9) 5( 7 9) 9.( S);) /
4#E( 8 9) 4( 8 9) 4( 8 9) 9.( S);) $R /
!+50 4H 5 /
4 3Q 6 /
!-50 !R3:2 5E( 7 9) 5( 7 5) 1( 3 5) 1( 3 5) /
4( 7 9) 4( 7 9) 4( 6 10) 4( 6 8) $R /
!+50 2H \Q 6E.( S);) /
H. E.( S);) /
!-50 !R3:2 7E( 9 11) 7( 9 11) 7( 9 11) $R 11.( S);) /
!R3:2 6E( 8 10) 6( 8 10) 6( 8 10) $R 11.( S);) /
```

Figure B.27: DARMS output: *Moonlight Sonata.*

```
!I1 !G,!G !M*C \H 6HL1 8Q 10 /
5. 6S(( 7.))L2 6Q RQ /
11HL3  10Q 13 /
10 9E( 8S( 9)) 8QL4 RQ /
!-50 !M*C 6H -1E(L5 3 1 3) -1( 3 1 3) /
0( 3 2 3) -1( 3 1 3)L6 /
-1(L7 4 2 4) -1( 3 1 3) /
-2( 3 0 3) -1( 3 1 3)L8 /
!+50 4( 5S(L9 6)) 7(( 8 9 10)) 11(( 10 9 8)) 7(( 6 5 4)) /
3E( 4S( 5)) 6(( 7 8 9)) 10(( 9 8 7)) 6(( 5 4 3)) /
2E( 3S( 4)) 5(( 6 7 8)) 9(( 8 7 6)) 5(( 4 3 2))L10 /
!-50 Q RQ RQ K Parse error $
!F 7,11J1  /
11J2,6 RQ RQ 11J3,6  /
5,11J4 RQ RQ 5,10  /
!+50 1E(L11 2S( 3)) 4(( 5 6 7)) 8(( 7 6 5)) 4(( 3 2 1)) /
0E( 1S( 2)) 3(( 4 5 6#)) 7(( 4 5 6)) 7(( 8 9 10)) /
11(( 12 13 12)) 11(( 10 9 8)) 9(( 10 11 10)) 9(( 8 7 6*))L12 /
!-50 11Q,4 RQ RQ 4,6 /
7W,9  /
7Q.L13  8E 9Q.  7#EL14 /
!+50 5(L15 10L16 8 6) 7(L17 10L18 8 6) /
7Q 5,7,10 3 RQ /
 RW  /
!-50 1S((L19 3 5 8)) 1(( 4 6 8)) 1(( 3 5 8)) 1(( 4 6L20 8)) /
1Q 8 1 RQ /
11#S((L21 12 11 12)) 11(( 12 11 12)) 11(( 12 11 12)) 11(( 12 11 12))L22 /
```

Figure B.28: DARMS output: *Sonata in C Major.*

```
!I1 !G,!G,!G,!T,!T !K1- !M*4:4 8E( 8 8 8) 8( 8 8 8) /
7J1( 7J2 7J3 7J4) 7J5( 7J6 7J7 7J8) /
8( 8 8 8) 8( 8 [\E] 8) /
7J9( 7J10 7J11 7J12) 7J13( 7J14 7 7) /
!-50 !K1- !M*4:4 8J15( 8J16 8J17 8J18) 8J19( 8J20 8J21 8J22) /
7*J23( 7J24 7J25 7J26) 7J27( 7J28 7J29 7J30) /
8J31( 8J32 8J33 8J34) 8J35( 8J36 8J37 8J38) /
7*J39( 7J40 7J41 7J42) 7( 7 7 7) /
!-50 !K2- !M*4:4 [!R2:1] 6Q [$R] E( 6J43 6J44) 6J45( 6J46 6J47 6J48) /
5*J49( 5J50 5J51 5J52) 5( 5 5J53 5J54) /
6J55( 6J56 6J57 6J58) 6J59( 6J60 6J61) [!R2:1] QJ62 [$R] /
EJ63( 6J64 6J65 6J66) 5*J67( 5J69J68 5J70) [\E] /
!-50 !K1- !M*4:4 9Q 10EJ71( 10J72) 10J73( 10J74 10 10) /
11*( 9 9J75 9J76) 9J77( 9J78) [\Q] /
10*J79( 10J80 10J81 10J82) 10J83( 10J84 10J85 10J86) /
9J87( 9J88 9 9) 9J89( 9J90 9J91 9J92) /
!-50 !K1- !M*4:4 7-( 7 7 7) 7( 7 7J93 7J94) /
7J95( 7J96 7J97 7J98) 7J99( 7J100 7J101 7J102) /
7#J103( 7J104 7 7) 7J105( 7J106 7 7) /
8J107( 8J108 8 8) 1( 1 1 1) /
!+200 6T((( 13S 13 10T))) 10((( 8S 8 6T)))
6((( ≠ [\T] 14 10)))
10((( 8S ≠ [\T] 6T))) 6((( 13S 12E 11T))) 10((( 9S 8 7T)))
6((( 14 13 11T))) 10((( 9 8 7T))) /
!-50  6Q [\Q \H] /
!-50 6 [\Q] [!R2:1] RW [$R] /
!-50 9 [\Q] [!R2:1] RW [$R] /
!-50 4S [\E \S] RQ [!R2:1] RW [$R] /
```

Figure B.29: DARMS output: *The Four Seasons (Winter).*

```
!I1 !T,!G,!T,!T,!G,!T,!F,!T,!T !M*3:4 \W /
 RW  /
 \W  /
 RW  /
 RW  /
!-50 !K1# !M*3:4  \H  6Q /
E(L1 7 8 7)L2 6( 5) /
 \H  4Q /
7S(( 7 7 7)) 8(( 8 8 8)) 9(( 9 9 9)) /
10E( 7 11 7) 12( 7) /
!-50 !K1# !M*3:4 3#H 4Q /
E( 5 6 5) 4(L3 3*)L4 /
H  2Q /
S(( 2 2 2))  !R4:1  3Q 3 3 3  $R  4S(( 4 4 4)) /
5Q  9  10  /
!-50 !M*3:4  \H  6 /
E( 7 8 7) 6( 5) /
 !R2:1  W  $R  4Q /
7  8  9  /
10L5  9  10L6 /
!-50 !K1# !M*3:4 3H 4Q /
E( 5 6 5) 4( 3*) /
H  2Q /
```

```
 !R3:2  2.  $R  3  4  /
3L7  4L8 5 /
!-50 !K1# !M*3:4  RW /
 RW  /
 RW  /
 !R2:1  7W  $R  Q  /
7L9  6L10 5  /
!-50 !K1# !M*3:4  \W  /
 RW  /
 \W  /
12H  11Q  /
10  9 8 /
!-50 !M*3:4  \Q  4# -1,4 /
1H  6E( 3),5 /
H  3Q /
1  1,3  2,4  /
3,0  -1 0,4 /
!-50 !K1# !M*3:4  \Q  6 2 /
4H.  /
5Q 5 5 /
5  13  12  /
10  9 8  /
```

Figure B.30: DARMS output: *Antiphonae Marianae.*

B.3 Recognition Results

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	0	0	0
Treble Clef	3	3 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	0	0	0
Thin Barline	36	36 (100%)	0
Thick Barline	0	0	0
Whole Rest	4	4 (100%)	0
Half Rest	5	5 (100%)	0
Quarter Rest	2	2 (100%)	0
Eighth Rest	7	7 (100%)	0
Sixteenth Rest	0	0	0
Flat (in key signature)	0	0	0
Flat	0	0	0
Sharp (in key signature)	8	8 (100%)	0
Sharp	5	5 (100%)	0
Natural	2	2 (100%)	0
Whole Note	0	0	3 (0.3311%)
Half Note	14	14 (100%)	1 (0.1121%)
Quarter Note	17	17 (100%)	0
Flagged Eighth Note	87	87 (100%)	0
Beamed Eighth Note	48	47 (98%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	0	0	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	15	12 (80%)	0
Space Code	165	165 (100%)	
Slur/Tie Association	24	24 (100%)	
Total	254	250 (98%)	

Table B.1: Recognition Results: *Magnificat (Vom Himmel hoch)*. See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	0	0	1 (0.1297%)
Treble Clef	2	2 (100%)	0
Bass Clef	2	2 (100%)	0
Repeat	0	0	0
Thin Barline	36	36 (100%)	0
Thick Barline	0	0	0
Whole Rest	0	0	0
Half Rest	0	0	0
Quarter Rest	0	0	0
Eighth Rest	24	24 (100%)	0
Sixteenth Rest	0	0	0
Flat (in key signature)	0	0	0
Flat	0	0	0
Sharp (in key signature)	6	5 (83%)	0
Sharp	0	0	0
Natural	0	0	0
Whole Note	0	0	0
Half Note	64	64 (100%)	0
Quarter Note	0	0	0
Flagged Eighth Note	0	0	0
Beamed Eighth Note	72	72 (100%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	0	0	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	26	26 (100%)	0
Space Code	136	134 (99%)	
Slur/Tie Association	52	52 (100%)	
Total	232	231 (100%)	

Table B.2: Recognition Results: *Canon in D.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	1	1 (100%)	3 (0.7833%)
Treble Clef	1	1 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	0	0	0
Thin Barline	40	40 (100%)	0
Thick Barline	2	0 (0%)	0
Whole Rest	0	0	0
Half Rest	0	0	0
Quarter Rest	6	6 (100%)	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	0	0	0
Flat	0	0	0
Sharp (in key signature)	0	0	0
Sharp	5	4 (80%)	0
Natural	4	3 (75%)	0
Whole Note	0	0	0
Half Note	1	1 (100%)	2 (0.5208%)
Quarter Note	113	111 (98%)	4 (1.4760%)
Flagged Eighth Note	0	0	0
Beamed Eighth Note	75	67 (89%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	0	0	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	13	11 (85%)	0
Space Code	179	179 (100%)	
Slur/Tie Association	21	22 (95%)	
Total	262	246 (94%)	

Table B.3: Recognition Results: *Concerto No. 1.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	**Occurences**	**Recognition Rate (%)**	**False Identification (%)**
Duration Dot	6	5 (83%)	2 (0.4405%)
Treble Clef	1	1 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	2	2 (100%)	0
Thin Barline	52	52 (100%)	0
Thick Barline	4	4 (100%)	0
Whole Rest	0	0	0
Half Rest	1	1 (100%)	0
Quarter Rest	17	17 (100%)	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	2	2 (100%)	0
Flat	7	5 (71%)	0
Sharp (in key signature)	0	0	0
Sharp	2	2 (100%)	0
Natural	1	1 (100%)	0
Whole Note	7	7 (100%)	0
Half Note	28	28 (100%)	0
Quarter Note	120	120 (100%)	2 (0.5882%)
Flagged Eighth Note	0	0	2 (0.4348%)
Beamed Eighth Note	36	32 (89%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	0	0	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	12	7 (58%)	0
Space Code	187	187 (100%)	
Slur/Tie Association	14	14 (100%)	
Total	299	287 (96%)	

Table B.4: Recognition Results: *Polovetzian Dance.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	0	0	0
Treble Clef	1	1 (100%)	0
Bass Clef	1	0 (0%)	0
Repeat	2	2 (100%)	0
Thin Barline	30	29 (97%)	0
Thick Barline	2	2 (100%)	0
Whole Rest	0	0	0
Half Rest	0	0	0
Quarter Rest	0	0	0
Eighth Rest	7	7 (100%)	0
Sixteenth Rest	2	2 (100%)	0
Flat (in key signature)	0	0	0
Flat	6	4 (67%)	0
Sharp (in key signature)	0	0	0
Sharp	7	7 (100%)	0
Natural	3	3 (100%)	0
Whole Note	0	0	0
Half Note	0	0	0
Quarter Note	9	9 (100%)	3 (0.9174%)
Flagged Eighth Note	6	1 (17%)	0
Beamed Eighth Note	92	86 (93%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	69	69 (100%)	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	10	2 (20%)	0
Space Code	165	164 (99%)	
Slur/Tie Association	4	4 (100%)	
Total	247	224 (91%)	

Table B.5: Recognition Results: *The Entertainer*. See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	8	8 (100%)	0
Treble Clef	4	4 (100%)	0
Bass Clef	4	4 (100%)	0
Repeat	6	6 (100%)	0
Thin Barline	76	76 (100%)	0
Thick Barline	6	6 (100%)	0
Whole Rest	5	5 (100%)	0
Half Rest	0	0	0
Quarter Rest	0	0	1 (0.1698%)
Eighth Rest	19	19 (100%)	1 (0.1754%)
Sixteenth Rest	45	44 (98%)	0
Flat (in key signature)	0	0	0
Flat	5	4 (80%)	0
Sharp (in key signature)	0	0	0
Sharp	22	20 (91%)	0
Natural	4	4 (100%)	0
Whole Note	0	0	0
Half Note	0	0	1 (0.1698%)
Quarter Note	4	4 (100%)	0
Flagged Eighth Note	27	27 (100%)	0
Beamed Eighth Note	12	12 (100%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	232	232 (100%)	2 (0.5602%)
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	67	65 (97%)	0
Slur/Tie	2	2 (100%)	0
Space Code	342	342 (100%)	
Slur/Tie Association	4	4 (100%)	
Total	548	542 (99%)	

Table B.6: Recognition Results: *Klavierstück.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	11	9 (82%)	0
Treble Clef	1	1 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	0	0	0
Thin Barline	20	20 (100%)	0
Thick Barline	0	0	0
Whole Rest	0	0	0
Half Rest	1	1 (100%)	0
Quarter Rest	1	1 (100%)	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	2	2 (100%)	0
Flat	1	1 (100%)	0
Sharp (in key signature)	0	0	0
Sharp	2	2 (100%)	0
Natural	1	1 (100%)	0
Whole Note	2	2 (100%)	0
Half Note	10	10 (100%)	0
Quarter Note	2	2 (100%)	0
Flagged Eighth Note	0	0	0
Beamed Eighth Note	116	111 (96%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	8	8 (100%)	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	0	0	0
Space Code	133	131 (98%)	
Slur/Tie Association	0	0	
Total	179	172 (96%)	

Table B.7: Recognition Results: *Moonlight Sonata.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	3	3 (100%)	1 (0.3145%)
Treble Clef	2	2 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	0	0	0
Thin Barline	26	26 (100%)	0
Thick Barline	0	0	0
Whole Rest	1	1 (100%)	0
Half Rest	0	0	0
Quarter Rest	12	12 (100%)	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	0	0	0
Flat	0	0	0
Sharp (in key signature)	0	0	0
Sharp	3	3 (100%)	0
Natural	3	1 (33%)	0
Whole Note	2	2 (100%)	0
Half Note	2	2 (100%)	1 (0.3135%)
Quarter Note	33	33 (100%)	0
Flagged Eighth Note	2	2	0
Beamed Eighth Note	46	46 (100%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	122	122 (100%)	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	14	13 (93%)	0
Space Code	205	205 (100%)	
Slur/Tie Association	26	24 (92%)	
Total	272	269 (99%)	

Table B.8: Recognition Results: *Sonata in C Major.* See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	0	0	0
Treble Clef	3	3 (100%)	0
Bass Clef	1	1 (100%)	0
Repeat	0	0	0
Thin Barline	25	25 (100%)	2 (0.3236%)
Thick Barline	0	0	0
Whole Rest	0	0	3 (0.4666%)
Half Rest	4	0 (0%)	0
Quarter Rest	4	1 (25%)	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	20	4 (20%)	0
Flat	0	0	1 (0.1555%)
Sharp (in key signature)	0	0	0
Sharp	1	1 (100%)	0
Natural	10	6 (60%)	0
Whole Note	0	0	0
Half Note	0	0	0
Quarter Note	4	3 (75%)	3 (0.4695%)
Flagged Eighth Note	0	0	1 (0.1555%)
Beamed Eighth Note	160	152 (95%)	1 (0.2070%)
Flagged Sixteenth Note	0	0	1 (0.1555%)
Beamed Sixteenth Note	0	0	12 (1.9017%
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	32	17 (53%)	0
Slur/Tie	80	54 (68%)	0
Space Code	172	172 (100%)	
Slur/Tie Association	108	108	
Total	344	267 (78%)	

Table B.9: Recognition Results: *The Four Seasons (Winter)*. See footnotes in Table 6.2 for explanation of how results are tabulated.

Symbol	Occurences	Recognition Rate (%)	False Identification (%)
Duration Dot	1	1 (100%)	1 (0.4000%)
Treble Clef	7	2 (29%)	0
Bass Clef	2	1 (50%)	0
Repeat	0	0	0
Thin Barline	45	45 (100%)	0
Thick Barline	0	0	0
Whole Rest	11	7 (64%)	0
Half Rest	0	0	0
Quarter Rest	0	0	0
Eighth Rest	0	0	0
Sixteenth Rest	0	0	0
Flat (in key signature)	0	0	0
Flat	0	0	0
Sharp (in key signature)	8	3 (38%)	0
Sharp	3	2 (67%)	0
Natural	2	2 (100%)	0
Whole Note	0	0	2 (0.7968%)
Half Note	15	6 (40%)	0
Quarter Note	62	52 (84%)	8 (4.2328%)
Flagged Eighth Note	0	0	0
Beamed Eighth Note	34	33 (97%)	0
Flagged Sixteenth Note	0	0	0
Beamed Sixteenth Note	24	20 (83%)	0
Flagged Thirty-second Note	0	0	0
Beamed Thirty-second Note	0	0	0
Slur/Tie	10	5 (50%)	0
Space Code	109	107 (98%)	
Slur/Tie Association	10	10 (100%)	
Total	224	177 (79%)	

Table B.10: Recognition Results: *Antiphonae Marianae.* See footnotes in Table 6.2 for explanation of how results are tabulated.