



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Research Commons

<http://researchcommons.waikato.ac.nz/>

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

COORDINATING KNOWLEDGE
TO IMPROVE
OPTICAL MUSIC RECOGNITION

A THESIS
SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE
AT
THE UNIVERSITY OF WAIKATO
BY
JOHN MCPHERSON

The University of Waikato

2006

Abstract

Optical Music Recognition (OMR) is the process of automatically processing and understanding an image of a music score. This process involves various distinct phases to transform the image into primitive shapes, musical objects, and ultimately into a syntactic model representing the music's semantics. In general, OMR systems have performed these tasks in a linear sequence, so that the output of one component is the input to the next. However, this means that processing errors that occur in one of the tasks propagate through the system, and often when the error is eventually detected it is too late to reconsider the decisions leading to the incorrect classification or information.

This thesis describes how OMR can be improved by modifying the recognition process from a sequence of linear tasks to a collection of modules that coordinate the information extracted from the data. Methods for data representation and controlling the system's flow of execution are investigated, and a practical implementation of such a system is described. This system has a message-passing design for providing contextual information from one module to another, such as suggesting possible classifications for an object. These messages are used to aid decision-making and to correct faulty decisions. This helps the system to adapt to a particular score while processing the image, increasing accuracy.

This system is designed to aid in the research and evaluation of algorithms to achieve the above aims; therefore it is straightforward to modify various aspects of the system's behaviour, such as adding support for different music symbols. Examining the implemented system's behaviour clearly shows that this coordinated approach can correct many errors and can even identify some objects by only using syntactic information, based on the surrounding objects.

Acknowledgements

Firstly, I must thank my supervisors David Bainbridge and Ian Witten for the advice and support I have received from them. David never ran out of enthusiasm every time a conversation was about music recognition problems and ideas; Ian has an incredible breadth and depth of knowledge of computer science topics. Both of them gave plenty of suggestions during the research and for the thesis itself.

Gwen has been wonderfully patient and supportive through my years of research and writing; I am eternally in her debt. My time at the University of Waikato has been enjoyable. I've also met many interesting people and new friends—the people in the Digital Libraries lab, the WAND lab, and WLUG have both taught me heaps and provided many social occasions.

I'd like to thank my family for their support, especially my sister Melinda for her careful proof-reading as the thesis neared completion.

This research was funded in part by the Marsden Fund of the Royal Society of New Zealand.

Contents

List of Figures	xiii
List of Tables	xvii
Index of Score Figures	xix
1 Introduction	1
1.1 Introduction to Music Notation	1
1.2 The OMR Process	4
1.2.1 Lower-Level Processing	5
1.2.2 Higher-Level Processing	6
1.3 Multiple Knowledge Sources in OMR	7
1.4 Aims of This Research	10
1.5 Thesis Outline	10
1.6 Glossary	11
2 History and Background	15
2.1 Previous Work	16
2.1.1 Carter	16
2.1.2 Kato and Inokuchi	16
2.1.3 Couïasnon <i>et. al.</i>	17
2.1.4 University of Leeds research group	18
2.1.5 CANTOR	19
2.1.6 GAMERA Project/Levy Sheet Music Collection	20
2.1.7 RIEM	21
2.1.8 Stückelberg <i>et. al.</i>	22
2.1.9 The MOODS Project	24
2.1.10 Interactive Music Network	24

2.1.11	Other Systems	25
2.1.12	Commercial Systems	26
2.2	Notation	26
2.2.1	Western Music Notation	26
2.2.2	Other Notations	29
2.2.3	Music Notation Software and File Formats	31
2.3	Related Fields	33
2.3.1	Optical Character Recognition	33
2.3.2	Chinese Character Recognition	34
2.3.3	Document Image Analysis	35
2.4	General Trends in OMR	35
3	Anatomy of an OMR System	39
3.1	Musical Semantics	39
3.1.1	Syntax versus Semantics	40
3.1.2	Syntax	40
3.1.3	Engraving	42
3.2	Object Identification	43
3.2.1	Segmentation	43
3.2.2	Classification	44
3.2.3	Assembly	46
3.3	Staff Processing	46
3.4	Text Processing	47
3.4.1	Identifying Textual Regions	48
3.4.2	Optical Character Recognition	48
3.5	Preprocessing and Filtering	49
3.5.1	Binarisation	49
3.5.2	Image Cleanup and Filtering	50
3.5.3	De-skewing and Deformation	51
4	Coordination of Knowledge Sources	53
4.1	Advantages of a Coordinated Approach	53
4.1.1	Dealing with uncertainty	54
4.1.2	Detecting and Correcting Errors	55
4.2	Knowledge Sources	58

4.3	Strategies for Coordinating Knowledge	59
4.3.1	“Top-down” versus “Bottom-up” Approaches	60
4.3.2	Frames	61
4.3.3	Blackboard Systems	62
4.3.4	Graph Rewriting	63
4.3.5	Genetic Algorithms	64
4.3.6	Discussion and Summary of Coordination Strategies	65
5	Practical Implementation	67
5.1	OMR Knowledge Coordination	68
5.1.1	Request-based Model	69
5.1.2	The Coordinating Process	69
5.2	Musical Semantics and Notation-Related Issues	69
5.2.1	Implementation Details	70
5.2.2	Unconventional Use of Notation	74
5.2.3	Syntactic Feedback for Coordination	75
5.2.4	High-level semantics	82
5.3	Primitive Assembly	83
5.3.1	Implementation	83
5.3.2	Assembly Difficulties and Examples	86
5.4	Primitive Identification	87
5.4.1	Patterns and Template matching	87
5.4.2	Compression-based template matching	87
5.4.3	Identifying Objects with Semantic Feedback	90
5.4.4	Segmentation of Objects	92
5.4.5	Practical Issues in Object Identification	94
5.5	Text Recognition and OCR	95
5.5.1	Musical Semantics of Text	97
5.5.2	Example of Text Detection	98
5.6	Staff Processing/Page Layout	98
5.6.1	Page Layout	101
5.7	OMR System and Coordination Issues	103
5.7.1	Boundaries between System Stages	103
5.7.2	Possible Coordination Strategies	104
5.7.3	System Progress and Tolerance Levels	105

5.8	Other Practical Difficulties	106
5.8.1	Typographical and Editorial Errors	106
5.8.2	Staff System Locations	107
5.8.3	Complex or Poor Layouts	107
6	Evaluation	109
6.1	Measuring System Performance	109
6.1.1	Measuring Effectiveness	110
6.1.2	Issues in OMR Evaluation	111
6.2	Evaluation of System Specialists	113
6.2.1	Preprocessing/Page Layout	114
6.2.2	Staff Processing	114
6.2.3	Object Identification	116
6.2.4	Primitive Assembly	120
6.2.5	Segmentation	122
6.2.6	Text Removal	124
6.2.7	Musical Semantics	126
6.3	Evaluation of System Interaction	133
6.3.1	Request Handling	133
6.3.2	Tolerance Levels	137
6.3.3	Flow of Execution	139
7	Conclusions	145
7.1	Original contributions of this thesis	145
7.1.1	Primitive Identification	146
7.1.2	Coordination of Knowledge Sources	146
7.1.3	Identification based on Assembly Knowledge	147
7.1.4	Identification based on Clustering	147
7.1.5	Run-time Pattern Adaption	148
7.1.6	Re-classification of Objects	149
7.1.7	System Design	149
7.2	Shortcomings and Challenges	150
7.2.1	Request Handling	150
7.2.2	Primitive Identification	151
7.2.3	System Evaluation	152

7.3	Future work	152
7.3.1	Priority of Requests	153
7.3.2	OCR	154
7.3.3	Musical Semantics	154
7.4	Key findings	154
7.4.1	Knowledge Coordination	155
7.4.2	Adaptation	155
7.4.3	Algorithm Settings and Thresholds	155
7.5	In closing	156
	Bibliography	157
	A Practical Implementation	165
A.1	Resource Limits for Specialists	165
A.2	Object Pattern Descriptions	167
A.3	Assembly Rules	170

List of Figures

1.1	A sample of Common Music Notation	2
1.2	A sample of Guitar Tablature (with corresponding WMN above) . .	2
1.3	A sample of Indian Bhatkhande notation	3
1.4	Variations in publishers' glyphs	5
1.5	Ornate example of Gregorian Chant	7
1.6	Generalised framework for OMR systems	8
1.7	Mozart Clarinet Concerto extract ($\frac{6}{8}$ time signature outlined)	9
2.1	The general RIEM framework	22
2.2	The grammar-driven OMR Framework proposed by Stückelberg . . .	22
2.3	An example of Western Music Notation	26
2.4	The traditional hand-engraving process	27
2.5	A ledger line being shortened for aesthetic reasons	28
2.6	An example of White Mensural Notation	30
2.7	An example of Sacred Harp notation	31
3.1	Examples of vertical stacking for simultaneous events	41
3.2	Spacing between notes	43
3.3	Examples of overlapping shapes	44
3.4	Primitive objects that can vary in shape	45
3.5	Finding staff lines by projection	47
3.6	The Chinese character for “flute”, composed of discrete glyphs . . .	48
3.7	Before and after binary thresholding	50
4.1	Sample Primitive Identification problems	56
4.2	Example of unrecognised objects (crotchet rests in the lower stave) .	57
4.3	Extract from model of proposed frame-based OMR system	62
4.4	A Blackboard System for audio transcription	63

5.1	The specialist modules in the OMR System	68
5.2	Extract showing semantic node structure	71
5.3	Semantic diagnostics	72
5.4	Examples of unusual and difficult notation	75
5.5	Notes and note groups found by the earlier modules	77
5.6	Objects as first encountered by semantics module	78
5.7	Objects encountered after feedback requests were processed	79
5.8	Treble clefs found in clarinet concerto extract	80
5.9	Looking for quaver rests with a too low threshold	82
5.10	Tree-graph showing assembly	85
5.11	Assembly of primitives into musical objects	85
5.12	Notes shared by two parts	86
5.13	Quaver rests found by compression-based template matching	89
5.14	Clustered objects identified by use of semantic information	91
5.15	A barline partially removed by the typesetter.	93
5.16	Extract showing dots and noise remaining after object extraction	95
5.17	Sample score showing both text and musical objects off the staff	96
5.18	Suspected text objects	99
5.19	Obvious peaks in staff system projection	100
5.20	Obvious deformation in staff system affecting projection	102
5.21	Using the Hough Transform for calculating skew	103
5.22	Alto (left) and Tenor (right) Clefs	104
5.23	A typographical error (missing rest)	107
5.24	Two staff systems side-by-side	107
5.25	Examples of difficult notation	108
6.1	Notes assembled incorrectly (first, second and fourth)	110
6.2	An error during staff-line extraction	115
6.3	Evaluation of Compression-based Template Matching	117
6.4	Objects identified after Primitive Assembly suggestions	121
6.5	Example objects created by Segmentation specialist	124
6.6	Example of detected, mis-detected and missed text objects	126
6.7	Extract of a score with syncopated rhythm	127
6.8	The time-slice offsets for the example figure	128
6.9	A note with incorrectly calculated pitch	130

6.10 An octave marking	130
6.11 Pitch calculation: Incorrect notes coloured, correct notes outlined . .	131
6.12 An extract from Intermezzo Op. 117 number 1, by Johannes Brahms	132
6.13 A graphical representation of the tolerance step's effect	139
6.14 Graphical representation of program execution flow (<i>Promenade</i> , page 1)	141
6.15 Graphical representation of program execution flow (for a blank image)	141
6.16 Graphical representation of program execution flow (Chopin)	143
A.1 Bitmaps described by sample pattern descriptions	169

List of Tables

6.1	Hypothetical Cost of Different Recognition Errors	112
6.2	Primitive Identification results for <i>Promenade</i>	119
6.3	Modifying the number of actions allowed by the Segmentation specialist	123
6.4	Performance considerations of Text Processing	125
6.5	Classification of Text Objects	125
6.6	Calculation of time-slice offsets and durations	128
6.7	Pitch Calculation algorithm accuracy	129
6.8	(Lack of) effect of request processing order	134
6.9	OMR Resource usage and limits for several scores	136
6.10	Effect of changing the system's maximum number of feedback requests allowed— <i>Promenade</i>	137
6.11	Effect of changing the coordinator's "tolerance step" on <i>Promenade</i>	140

Index of Score Figures

Bach — Prélude from English Suite no. 1 in A BWV 806

Figure 3.4	page 45
Figure 5.4	page 75
Figure 5.15	page 93
Figure 5.20	page 102
Figure 5.25(a)	page 108
Figure 5.14	page 91
Figure 5.17	page 96

Brahms, J. — Capriccio (Opus 116, No. 7)

Figure 2.3	page 26
------------------	---------

Chopin — 24 Préludes, Opus 28

Figure 3.5	page 47
Figure 5.18	page 99
Figure 5.12	page 86

Chopin, Frédéric — Scherzo 58

Figure 3.7(a)	page 50
Figure 3.7(b)	page 50

Mozart — Clarinet and Piano Concerto

Figure 3.1(a)	page 41
Figure 5.3(a)	page 72

Figure 5.13	page 89
Figure 5.23	page 107
Figure 5.11	page 85
Figure 5.8(a)	page 80
Figure 5.8(b)	page 80
Figure 5.9(b)	page 82
Figure 6.11	page 131
Figure 6.6	page 126

Mussorgsky — Promenade, from “Pictures at an Exhibition”

Figure 3.1(b)	page 41
Figure 4.1	page 56
Figure 4.2	page 57
Figure 5.5	page 77
Figure 5.6	page 78
Figure 5.7	page 79
Figure 5.9(a)	page 82
Figure 6.13(b)	page 139

Taupin — Récit (1990)

Figure 5.4(b)	page 75
Figure 5.24	page 107

Cappella Sistina 10 fols. 2 verso-3

Figure 1.5	page 7
------------------	--------

Vajra, Frank — Overdrive

Figure 1.2	page 2
------------------	--------

Chapter 1

Introduction

Technology is becoming increasingly indispensable in many aspects of life—especially for entertainment and leisure activities—and the potential for this trend to grow is enormous. Imagine a violin player rehearsing alone for a duet, taking a photo of the sheet music and having a nearby computer automatically play the accompanying piano part. Optical Music Recognition (OMR), sometimes also called *musical score recognition* or simply *score recognition*, is the process of automatically extracting the musical meaning from a printed musical score. It is this process that would allow the violinist to have a computer play the piano part of the score when the sheet music is available but a piano and pianist are not.

This chapter gives a brief introduction to music notation, the aim of Optical Music Recognition, and an outline of the steps taken by an Optical Music Recognition system. This is followed by a description of some of the problems and challenges faced by such systems, and how some of these issues can be mitigated or resolved by designing a system to allow information to be passed backwards and forwards so that decisions can be made using contextual knowledge. Finally, the research described in this thesis is outlined.

1.1 Introduction to Music Notation

Music notation provides a rich description of the composer’s ideas, but ultimately sheet music is open to some degree of interpretation by performers. Music notations evolve; symbols are added and conventions modified as composers adapt to new instruments and ideas.

Western Music Notation (WMN), sometimes called *Common Music Notation*

(CMN) or *Western staff notation*, is the notation most widely used today—an example of WMN is shown in Figure 1.1. A score consists of staff systems containing one or more staves; the score in this figure shows a staff system with three staves, where the top staff is for the flute and the lower two staves are grouped together for the piano.

More specialist music notations include guitar tablature (Figure 1.2), Gregorian chant, Sacred Harp notation, and various Asian, African and Indian musical notations (Figure 1.3). These can be roughly split into two: a group consisting of notations that are based on a *staff*, and those that are not. These notations are described further in Section 2.2.

Figure 1.1: A sample of Common Music Notation: Handel’s Sonata V for flute and piano

Figure 1.2: A sample of Guitar Tablature (with corresponding WMN above)

The advantages of a computerised representation of a musical score are numerous. These include commonly-cited musical-specific advantages, such as:

- the ability to automatically transpose a particular voice or part in a score—for

वसंत-त्रिताल (मध्य लय) (Rag, Tal, and Tempo)

स्थायी		Sthai	
नि सा ऋ ३	ग सा म म तु व सं ग म - म म सा ऽ द त	- म नि ध ऽ त व न X म म नि नि X	नि सां नि ध प ऽ ल र र मे ग - मे ह र ऽ फु र
	(प) मंग म ग		ही ऽ ऽ ०
			ग रे - सा ल बा ऽ रि ०
			(Melody) (Lyrics) (Tal Signs) (Melody) (Lyrics) (Tal Signs)
	vibhag	vibhag	vibhag

Figure 1.3: A sample of Indian Bhatkhande notation (from *Hindustani Sangeet Paddhati — Kramak Pustak Malika*, by Vishnu Bhatkhande, 1933–1936, Vol 4.)

example, clarinets read music set in B \flat , so music written for “C” instruments (such as piano, flute, and stringed instruments) should be transposed up one tone for correct reading by these performers, and similarly music written for “B \flat ” instruments (including many brass and woodwind instruments) should be transposed down one tone for performance by “C” instruments. Common transpositions for orchestral instruments include octave transpositions (both higher and lower), B \flat , D, E \flat , and F [70];

- part extraction from a conductor’s score (which has the whole orchestra’s music in a rather small typesetting face), or conversely generation of a conductor’s score from the individual parts;
- converting the representation to other musical formats or notations, such as for Braille-reading machines or for various software packages which use a variety of file formats; and
- allowing musicians to read the music from a computer display, for example to eliminate the need for manual page turns [37, 51];

and more generalised document-processing advantages such as:

- a form of file compression to save disk space [6];
- ease of sharing and archiving;
- increased ease of editing (using appropriate software), aiding in composition and re-typesetting or re-publication; and

- automatic indexing and retrieval of information [50, 31].

1.2 The OMR Process

To extract and understand the musical meaning from sheet music, Optical Music Recognition systems typically perform multiple levels of processing. This generalised problem is not specific to OMR—the sheet music domain is part of a larger field known as Document Image Analysis [7, 61]. Document Image Analysis refers to various categories of systems designed to automatically understand paper documents. Examples include banking systems (cheque recognition), envelope and letter address parsing, technical drawings and blueprints, circuit board diagrams, maps, and so forth. The general process for document image analysis consists of the following steps:

1. Acquire an image of the relevant document.
2. Preprocess to get the image into an acceptable state (for example, perform enhancements such as colour reduction, de-skew, remove noise).
3. Locate “regions of interest”.
4. Extract features from regions of interest.
5. Perform feature analysis.
6. Generate document description.

Steps 1–4 work at the image level, on the raw image data. Steps 4, 5 and 6 work with features, while steps 5 and 6 also use domain-specific knowledge.

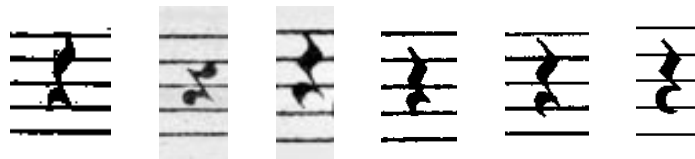
Applying this model to the OMR process, the lower level routines take the image of the music as input and perform different image processing techniques to recognise the different musical shapes and objects. These shapes need to be further processed to build up an internal model of the image. The higher level routines determine the abstract “musical meaning” of the objects found by the lower level routines, by using knowledge of music semantics. The OMR-specific aspects of this process are now briefly described; how these aspects can be used in each step of the process is explored in detail in Chapter 3.

1.2.1 Lower-Level Processing

The low-level processing phases need to recognise all the shapes (and their positions) in the image that make up the musical components. The different shapes will depend on the notation used, as will the layout—for example, as shown in the previous figures, music represented in WMN consists of objects superimposed on a staff consisting of multiple horizontal lines. This process needs to account for graphical details, such as varying glyphs and layout used by sheet music publishers, as well as musical symbols that can change shape within the same piece. Figure 1.4(a) shows a variety of bass clefs as used by different publishers, and Figure 1.4(b) shows variations in crotchet rests. Several of each class have been generated by computer notation software, some are hand-drawn, and there are examples of older shape styles that look quite different to the more modern style for each class.



(a) Various Bass Clef glyphs



(b) Various Crotchet Rest glyphs

Figure 1.4: Variations in publishers' glyphs

Ideally, the physical appearance of the input image would be completely described so that every element on the page is classified as a graphical shape known by the system. In practice, however, the difficulties in achieving this means that recognition systems are far from perfect.

As well as using standard image-processing and document image analysis techniques, there are music domain-specific qualities that can be examined to help improve recognition quality. As an example, for music notated in WMN, the thickness of staff lines can be used to infer the scan resolution, or might be correlated to the brightness/darkness setting of the device when the image was acquired. There are also less obvious qualities specific to music that might be useful for recognition; for example, the Lilypond computer software [60] for typesetting WMN offers the following insight:

Another typical aspect of hand-engraved scores is the general look of the symbols. They almost never have sharp corners. This is because sharp corners of the punching dies are fragile and quickly wear out when stamping in metal.

Rules and observations about the physical appearance of music such as this can be exploited to improve the accuracy of the recognition process.

1.2.2 Higher-Level Processing

The higher-level processing of notation determines the semantic meaning of the sheet music. At an abstract level, the semantics of sheet music are only interpreted when realised by a performer, and different musicians will not give identical performances. However, as far as the OMR process is concerned, “semantics” refers to a model of the music complete enough to faithfully re-create or perform the score such that a person would find it to be functionally identical to the original sheet music.

This involves calculating the musical properties and syntax of the graphical objects found by the low-level processing phases, based on some set of rules for that particular music notation such as WMN or guitar tablature notation. This process commonly includes determining the duration and pitch of objects based on the low-level shapes found for notes, and these attributes are affected by other objects such as durational dots and slur or tie markings that modify the duration of the sounded note, and key signatures and accidentals that modify the pitch. Generally, the final step of an OMR system is to store the calculated semantics of the music into a data file in a format that computer music-notation programs can process.

Figure 1.5 shows an extract of a Gregorian chant manuscript, written in *circa* 1520. A score such as this poses multiple difficulties for automated OMR: the use of colour—the colour of the stave lines is similar to the background colour—makes it harder to process, the use of “reverse” colour where the music and text is surrounded by a different background colour in some regions, the presence of bleed-through of ink from the other side of the page, and decorative images overlapping the music and lyrics. Although these difficulties are image-related rather than music-specific, music knowledge may be useful for correcting any deficiencies in the lower-level processing.

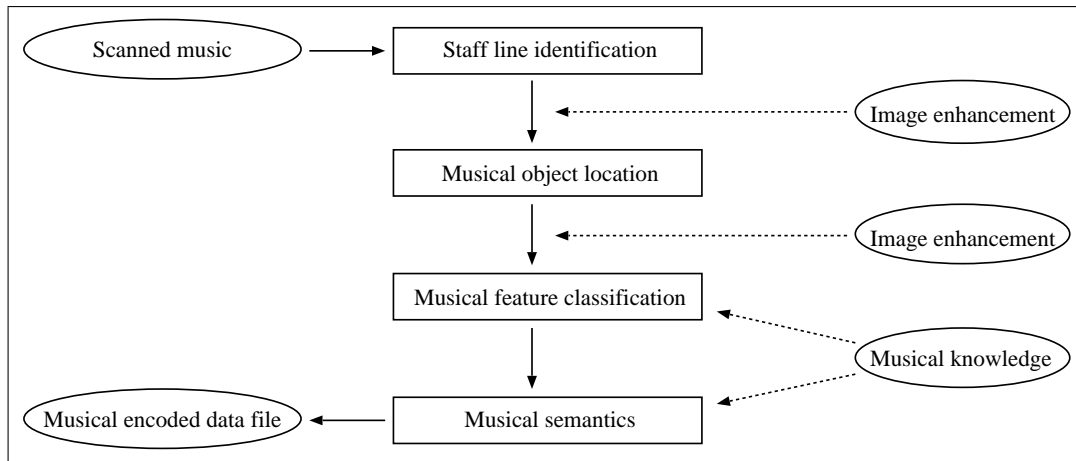


Figure 1.5: Ornate example of Gregorian Chant

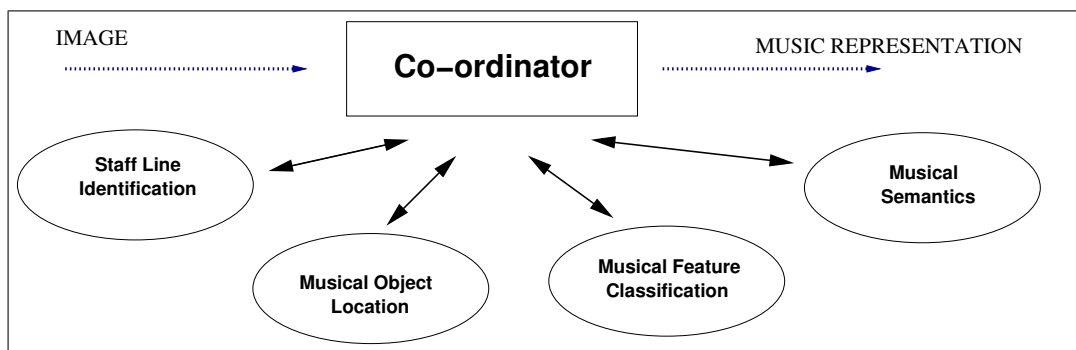
1.3 Multiple Knowledge Sources in OMR

Much data about the music represented by the score can be derived by the low-level and high-level recognition stages previously described, and this requires knowledge of musical symbols and rules. This knowledge is applied in multiple phases to perform those recognition tasks.

Figure 1.6(a) shows how OMR has traditionally been performed. There is a linear sequence of operations performed, with each phase's output being the next phase's input. However, this model has some limitations. Most seriously, errors made in an early step will propagate through each of the following steps; each phase has to try to correct errors and omissions in its input. For example, when doing musical semantic analysis on the recognised components, an error may be detected,



(a) The traditional “pipeline” approach to OMR



(b) The “coordinated” approach to OMR

Figure 1.6: Generalised framework for OMR systems

such as a bar of music not having enough (or too many) notes in it. Because this type of error cannot be corrected within the current context, the system is forced to output something that it knows is not quite right. Some errors, such as a missing or mis-detected accidental in a key signature, could conceivably be inferred correctly in this context. For other types of errors, the semantics analysis may be able to calculate some information about an error, but not enough to unambiguously resolve the mistake. For instance, the expected duration of a single missing object can be determined from the surrounding objects, but not the pitch.

A framework that allows different stages to flag errors (along with extra information about what is wrong and what was expected), and allows the stages to be executed multiple times, taking any new information into account such as errors flagged by other stages has the potential to greatly increase the system’s overall accuracy by using extra data and knowledge to fix errors. Figure 1.6(b) shows a generalised strategy for such a system—some *coordinating* process is required for deciding the execution order of the stages, as well as when to stop processing.

Another example of WMN is shown in Figure 1.7. The music is in $\frac{6}{8}$ time,



Figure 1.7: Mozart Clarinet Concerto extract ($\frac{6}{8}$ time signature outlined)

meaning that each bar has six quaver beats in it although this particular example also has a short lead-in bar. If the $\frac{6}{8}$ time signature on one of the staves is not recognised by the lower level routines, it could still be inferred from the surrounding musical objects. This particular piece has multiple staves in the system, and these should have the same time signature, so if the time signature is successfully identified in another staff it could be assumed that the missing signature is the same. (While some modern scores do have different time signatures in effect at the same time, the vast majority of scores do not.) Alternatively, the time signature could be inferred by examining the duration of the notes in the bars, although this is more problematic because different time signatures can allow the same number of beats— $\frac{6}{8}$ time, with six quaver beats, is the same total duration as $\frac{3}{4}$ time (three crotchet beats), although the former is normally sub-divided into two groups of three.

For the example in Figure 1.7, using a coordinated holistic approach means that instead of the Musical Semantics stage finding an inconsistency, making a ‘best guess’, and continuing, the extra semantic information about the missing object can instead be used to re-perform the low-level identification stage. The identification stage could limit its recognition routines to only the missing types—in this case, types that can be used to make a time signature, or even more specifically, only object types that can make a $\frac{6}{8}$ time signature.

More powerful computing hardware allows an OMR system to perform much more processing in a reasonable amount of time compared to the hardware available to researchers in previous decades. As computers get faster and storage gets cheaper,

an OMR system can afford to try more things to improve the accuracy—for example trying alternative algorithms to perform some task—rather than aiming for a one-shot “best effort”. This behaviour, including re-evaluating decisions when faced with contrary information about objects, is explored and evaluated in this thesis through the development of an experimental OMR system.

1.4 Aims of This Research

The aim of the research described in this thesis is to understand how the various components of an optical music recognition system can interact with each other, and how various factors influence both the dynamic behaviour of the system and the resulting accuracy of the recognised music. In particular, the use of feedback for correcting errors will be examined. This can be broadly summarised as:

1. Determining which types of errors can be correctly detected;
2. How these errors can be represented;
3. How this data can be used by the different phases of the OMR process to improve accuracy.

This leads to many questions. What should be done when an error is detected but is of an indeterminate cause? How should an OMR system handle conflicting information? The extent of these problems, and their possible solutions, is the crux of this research.

1.5 Thesis Outline

Previous work in the field of OMR and relevant document image analysis disciplines is reviewed in Chapter 2, with emphasis on recent research and existing systems. This also gives an introduction to the basics of music notation, and file formats used by computer programs for music representation. This leads into Chapter 3, which describes the general process typically performed by OMR systems, starting with the end goal of the process. The goal, as previously stated, is to produce a computer representation of the musical semantics conveyed by the score, and this chapter describes the tasks and sub-tasks required to achieve this, given an input image.

Different strategies for internally representing and controlling information, and their strengths, weaknesses and suitability for use in an OMR system, are investigated in Chapter 4. This includes methods for dealing with uncertainty and conflicting data, especially for detecting and correcting errors in object recognition.

Chapter 5 describes a practical OMR implementation that uses some of the strategies outlined in the preceding chapter. This includes details of *how* the steps described in Chapter 3 are performed, particularly how they are used in a system that coordinates the flow of data and provides feedback. This also discusses challenges encountered, and methods that can be used to improve recognition quality.

The performance of the individual methods and the system performance of the coordinated approach as a whole is evaluated in Chapter 6. This includes discussion on evaluation strategies for measuring various aspects of the OMR implementation. Finally, Chapter 7 summarises the key concepts of this work, discusses problems encountered, and presents the main findings of the research.

1.6 Glossary

Bitmap A two-dimensional array of image data, corresponding to the *Pixels* (*q.v.*) to be displayed. Pixels in a black-and-white image only need one bit per pixel while a colour image allowing 256 values (8 bits) each of red, green and blue requires 24 bits per pixel.

CMN Common Music Notation (or alternately Conventional Music Notation). An alternate name for *WMN*.

DPI Dots per Inch. This refers to the resolution of an image produced by digital-to-analog (or analog-to-digital) devices such as scanners and printers. 300 DPI is sufficiently high for OMR purposes.

Engraving The process of preparing metal plates containing a score, to be used on a printing press.

Genetic Algorithm An artificial intelligence/machine learning technique that, given a “fitness function” to measure performance, continually finds better solutions by combining the best solutions currently found, as well as introducing some random variation.

Key Signature A series of sharps or flats describing the key for the following bars.

These are normally shown at the start of every staff, and are also found before any bar where the key changes.

Musical Syntax A set of rules describing the interactions between a set of objects.

This refers to the order and attributes of the objects, and different music notations will have different object types that are described by different syntaxes.

Musical Semantics The abstract musical quality expressed by a score. The semantics are represented by objects that are in a particular syntax.

Noise Unimportant artefacts in an image formed during the printing or digitisation of a score. This can be caused by dirt on the paper, printed material on the reverse side showing through, or limitations in the hardware used to capture an image of the score.

Notation Any system for transcribing music, using a set of symbols. Different notations have different sets of symbols and conventions for interpreting them.

OCR Optical Character Recognition. The process of automatically recognising letters and words from an image of typed or printed text.

OMR Optical Music Recognition. Refers to the entire process of automatically processing an image of sheet music.

Primitive A single graphical shape extracted from an image. A primitive either forms an object in isolation, or it is only a component of a larger object and is not a valid object in its own right.

Primitive Assembly The process of joining two or more primitive objects into a single object. For example, an oval and a vertical line—neither of which are musical objects by themselves—can be joined into a note, and two notes and a beam can be joined together into a beamed note-group.

Pixel (from *Picture Element*). An individual ‘dot’ in an image.

Skew Rotation of the image caused by the score not being perfectly aligned during scanning.

Tablature Normally refers to the *guitar tablature* notation. Guitar tablature has six lines (one for each guitar string), and consists of numbers super-imposed on

the lines, representing which fret on the guitar to play. Another instrument that uses a tablature notation is the harp.

Template Matching Comparing two images (where one is an idealised template) on a pixel-by-pixel basis.

Time Signature Two numbers, one above the other, describing a score’s meter: the upper number gives the number of beats per bar, and the lower number gives the division for each beat. These are normally found at the start of a score, and will occur before any other bar where the score’s meter changes.

Transpose Performing or re-typesetting music at a higher or lower pitch than shown in the score. For example, a score for a B♭ transposing instrument such as a clarinet would be typeset one tone higher than it actually sounds.

WMN Western Music Notation. The predominant notation used in sheet music in the Western hemisphere.

XOR Exclusive Or. For two tests, either one or the other—but not both—is true. In this thesis, it refers to comparing binary bitmaps by checking if the corresponding pixels in each location are both set to the same value.

XML Extensible Markup Language. An open standard that allows data to be represented in a consistent and well-structured format.

Voice For scores with multiple performers, each class of performer is called a voice. Some scores use “split voice” with two voices on the same staff; it is common for vocal music to have sopranos and altos on one staff (with stems up and down respectively), and tenor (with stems up) and bass (stems down) voices on another staff.

Chapter 2

History and Background

Computer Science is a young discipline; the first departments were introduced into universities in the 1960s, and the earliest researchers to study the processing of musical scores by computer were Ph.D. students at the Massachusetts Institute of Technology in the late 1960s. Given the limited hardware available, their efforts were targeted at small subsets of music notation—the first recognising primarily notes within note-groups, and the second recognising object types that did not overlap. Since then the availability of computers, particularly the advent of personal computers in the 1980s, has dramatically increased the number of researchers in Computer Science. Computing hardware has seen exponential increases of processing power, and this continues to increase; this, along with other hardware improvements, such as flatbed scanners becoming commodity consumer add-ons, has seen many projects undertaken in image processing and related fields such as OMR. For document image analysis fields, some of the improvements include more faithful image acquisition and representation—due to higher image resolution and available memory storage—and more sophisticated algorithms, due to faster processors and more memory storage.

This chapter focuses on advances in the OMR field, with particular attention given to the methodologies proposed and used. The first section covers OMR systems or sub-systems, the second section gives an overview of computers and music notation, and the third section looks at related image analysis fields that have some similarities to OMR.

2.1 Previous Work

One of the most notable OMR systems was the Wabot 2 project. For this project, undertaken in Japan during the first half of the 1980s, a robot was built that could—among other things—play an electronic organ when sheet music was placed in front of it. Although it could not cope with complex sheet music, it could successfully understand chords as well as a large set of symbols. This was achieved through the use of specialised hardware, using over 60 processors to perform template-based matching, a relatively expensive recognition technique, over the image acquired by a video camera.

A good overview of significant work up until the early 1990s is given by Blostein and Baird [15]. Details are now provided for several of the more important works in the area from the last two decades.

2.1.1 Carter

Carter [22] describes an OMR system that is notable for identifying primitives without first removing the staff lines. This is achieved by using a Line Adjacency Graph technique that examines consecutive vertical slices consisting of runs of black pixels. Primitive shapes have a characteristic profile of vertical slices and this is how they are classified.

2.1.2 Kato and Inokuchi

Kato and Inokuchi [44] describe a complete OMR system. This is a “top-down” system using a “layered working memory” approach, with different layers for raw *image* information, *primitives*, *symbols* composed from primitives, *meaning* of symbols, and the final *goal*. These regions of memory are modified by individual modules for primitive extraction, symbol synthesis, symbol recognition, and semantic analysis. Hypotheses about objects in the symbol layer may be rejected in the primitive layer but not the image layer. The top-most layer is the goal layer, which contains hypotheses consisting of the ordered notes in the bar and their attributes from the meaning layer. The semantic analysis is done at a bar level, and tests that the beats in the bar add up correctly for the time signature. If the bar does not add up, the goal-level hypothesis about those objects is rejected, and heuristic rules are applied—for example, to test if any of the objects are ‘tuplets’ with a different duration.

In practice, the system performs preprocessing to find barlines, and then performs recognition and analysis of objects at the bar level, with a final post-processing stage to find objects that are outside a single bar and to combine the results of each individual bar. Symbols are identified in order of “clearness”, and these symbols are processed first before looking for symbols of a lower quality. The top-down approach also means that musical knowledge is used during primitive extraction from the image—for example, the system only looks for accidentals near recognised noteheads and in key signature locations. One limitation noted about this strategy is that top-down approach for accepting hypotheses in the final goal means that if the current set of objects in a bar fulfills the semantic rules for a bar, no further objects will be found for that bar, such as missing notes in another voice.

Hardware limitations of the time meant that processing a sheet of piano music took ninety minutes, so the set of symbols recognised is restricted to a core set that covers the majority of symbols that represent sound (notes, rests and accidentals) as well as simple shapes such as barlines and dots.

2.1.3 Coüasnon *et. al.*

Coüasnon and others [26, 27] at the IRISA Institute in France describe the use of a parser implementing an attributed grammar to build musical objects from graphical primitives. This is done at a bar level; durational analysis is then done by comparing objects in all staves that occur at the same time onset. Primitives such as noteheads and stems are identified using a Kalman filter—a linear programming method for calculating state transitions [40]—on the input image.

The grammar describes both the construction of musical objects—which they call “constructs”—from graphical primitives (“segments”), and the relationship between constructs and “symbolics”, which are the isolated shapes that are not part of note groups. This grammar has both a graphical component, corresponding to the physical image, and a logical component corresponding to the syntactic meaning. The grammar rules have attributes describing the relationship between objects—for example, relative positions. The grammar controls the high-level recognition process; the system tries to satisfy the first grammar rule, which leads to other rules describing constructs, which are objects made up of other objects, and segments, which are the primitive objects.

However, while the paper describing their prototype system says that “[i]n the

future, it should be possible to improve the recognition by going back to the image level” to correct unrecognised or mis-detected notes, there is little evidence in the literature that they continued this line of research.

2.1.4 University of Leeds research group

A group of researchers at the School of Music at the University of Leeds have been actively researching OMR and related areas. The research group (Ng, Boyle and Cooper) has published several papers and technical reports since the mid-1990s. After Ng completed a doctorate in 1995 [58], the focus of his research shifted more towards handwritten manuscript recognition [55].

They suggest using music domain knowledge to guide and improve the primitive recognition; their method uses *a priori* knowledge of the time signature and key signature. They also suggest that these could be determined by examining the recognised primitives.

Segmentation is performed on graphical objects until the object is made entirely of recognisable primitives. For example, horizontal and vertical cuts are used to remove noteheads from vertical lines. The cutting method is determined by the component’s aspect ratio, relative to the height of the gap between staff lines. The newly segmented objects are recognised, and objects that remain unrecognised are then segmented again. This process loops until all objects are recognised or are too small or too solidly filled to segment again.

After segmentation, a nearest neighbour (NN) classifier is used for recognition, which compares a set of features of each unknown object to the features for a training set of examples. Originally, the bounding box/aspect ratio was the only feature used, with acceptable results.

This is followed by a Reconstruction phase [53] for creating musical primitives from the identified graphical shapes. (This process is referred to as Primitive Assembly in other literature, such as CANTOR and this thesis.) This algorithm looks around recognised note heads for “complementary features” such as stems, durational dots and accents. Accidentals are looked for, and if they are near a detected note head then further processing is performed to identify the type of accidental.

Finally, domain-level musical knowledge is used to confirm or correct the built up model. The time signature is detected and each bar is checked to confirm it contains the correct number of beats. Heuristic observations are also used for metric

correction. For example, they have observed that beamed pairs of notes often consist of a dotted note head and a halved note head, so if the bar duration is incorrect, and a beamed note pair has a dotted note without a halved note (or vice versa), then that is a likely candidate for duration correction.

Related work by these researchers includes papers on automatic key signature detection based on pitch cumulative durations [57], and meter (time signature) detection from semantic analysis based on the number of quaver beats and beat divisions in a bar [52, 54].

2.1.5 CANTOR

Bainbridge’s CANTOR [5] was a pioneering OMR system: it was complete and extensible, able to be configured to recognise arbitrary object shapes and to output the semantics of the score in several musical file formats. Most prior systems were limited to work on small subsets of Western Music Notation (WMN), had hard-wired musical semantics processing rules, and often made assumptions about staff lines, such as that there were always five lines per staff. While CANTOR still has some restrictions, such as requiring the input music to be staff-based (although allowing an arbitrary number of lines per staff), there is considerably more freedom over what can be processed.

CANTOR consists of four main steps:

Staff line identification, which locates staff systems and staves, removes staff lines and locates objects in the bitmap.

Primitive Recognition, which identifies basic shapes, such as slurs, noteheads, tails, accidentals, and lines, using a specially designed programming language.

Primitive Assembly, which joins the basic primitives found into musical objects, such as noteheads, stems and tails into a note; and

Musical Semantics, which determines musical qualities such as pitch and duration of the musical objects found, and can output various musical file formats.

As mentioned above, CANTOR was designed to be *extensible*. Here, extensible refers to the fact that one of the design goals was to research and design a system that did not have hard-coded shapes or rules built into it: different notations can be recognised by supplying configuration data describing these aspects of each one. This research led to the formation of Primela—a **Primitive Expression Language** for describing specific musical shapes in the Pattern Recognition stage. A set of

Primela descriptions can be written to describe symbols within a particular music notation and then loaded and used at run-time, to process an image. Similarly, rule sets are created for the Assembly and Semantics stages, so that the system can be extended to process different music notations.

CANTOR was designed to be a feed-forward system, with the results of each stage becoming the input of the following stage. Because of this, decisions (such as the classification type of a particular object) can only be made in the context of what is already known, and some detected errors cannot be automatically corrected. For example, if it is detected that there are too many notes in one bar, the Musical Semantics stage cannot determine exactly which note or notes are incorrect.

Although CANTOR was designed as a feed-forward system with clearly defined and separated tasks, some semantic knowledge is used to increase the accuracy of the Primitive Recognition stage. For example, the Primela rule provided for matching a treble clef in WMN is based on the symbol size, but includes a proviso that the objects must be near the start of a staff. This is technically semantic information, although allowing such clauses before the “official” semantics parsing can greatly improve the accuracy of the recognition.

Sample rule-sets for use with CANTOR have been written for various notations, including WMN, Sacred Harp Notation [25], and “Plain Song” (used in 16th century Gregorian chants). Other work on CANTOR has been done to improve the efficiency and accuracy of various sub-routines, such as the bitmap matching [76].

2.1.6 GAMERA Project/Levy Sheet Music Collection

The GAMERA Project, which is digitising the Levy Sheet Music Collection at John Hopkins University in Baltimore, Maryland, USA, is one of the few currently active research groups developing OMR technology. Fujinaga’s 1997 Ph.D entitled Adaptive Optical Music Recognition [36] describes a system for graphical recognition of musical shapes. The system can be trained to recognise new shapes, and it uses a genetic algorithm to tune the set of weights for the features used for classifying graphical objects. Due to the large amount of computation required, this training is done ‘off-line’—that is, it must be completed over a relatively long period of time before it can be used for recognition. After the recognition is performed using the computed weights, the results of the run-time classification are stored for future re-computation of the weights.

Later work by the project has included the syntactic and semantic parsing of the recognised objects—this is described as Optical Music Interpretation (OMI) [28]. The input for this OMI process is a set of recognised objects with corresponding attributes (such as size, recognised type, and position). Because the process involves two distinct phases (AOMR and OMI), there is little scope for using feedback by applying semantic context to the graphics recognition stage. Nevertheless, they report good results for their data-set. The sheet music in the Levy collection is predominantly American popular music from the late 18th through to the mid-20th centuries.

GAMERA has evolved from the purely music recognition system described in Adaptive Optical Music Recognition into a more generic document analysis system [30]. For example, it has been used to recognise hand-written medieval manuscripts, and Kanji (Chinese characters used in Japanese writing); however, the object recognition process does not make use of syntactic or semantic information.

2.1.7 RIEM

Portuguese researchers Ferrand and Cardoso at the Universidade de Coimbra and Leite at the Universidade Nova de Lisboa have published several papers describing their OMR system, named “RIEM”. They propose [34, 33] an OMR system for dealing with uncertain and missing information, and describe a system implemented by using Constraint Logic Programming for representing and applying rules and hypotheses. These papers, along with the main paper describing the RIEM system [47] (in Portuguese) describe concepts that are similar to those discussed in this thesis; feedback between “Interpretation” and “Recognition” modules provides location information about possibly missing objects. Their recognition process involves calculating temporal attributes, and consistency checking of object durations between voices. The Interpretation process checks syntax, based on rules encapsulating Western Music Notation. Non-temporal objects are not handled by the feedback between the interpretation and recognition; this feedback is only used to look for temporal objects that are missing from a voice. The general architecture of RIEM is shown in Figure 2.1.

It is unclear from these papers how many of their proposed ideas were implemented into their system, and there was little evaluation of the performance of the system’s architecture. There are no similar journal or conference publications by

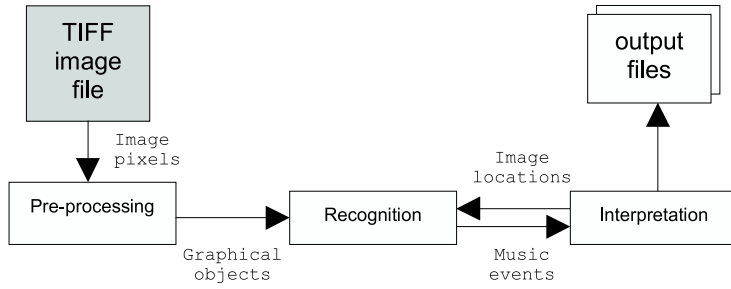


Figure 2.1: The general RIEM framework [34]

these authors after 1999. However, Miguel Ferrand’s completed M. Phil. [35] may have more detail than the other papers, although it is in Portuguese and not readily available.

2.1.8 Stückelberg *et. al.*

Stückelberg *et. al.* have published a series of papers [73, 72, 71] proposing a grammar-driven music recognition system. Using grammatical rules to describe Western Music Notation, the probabilities of individual pixels in the image belonging to particular types of objects can be calculated in a “top-down” manner. The grammar drives the complete process, including the primitive recognition phases, by defining regions of interest for particular primitive types that would satisfy the grammar rules. Figure 2.2 shows a diagram representing this grammar-driven process. Features are calculated on parts of the input image as requested by the higher-level system components.

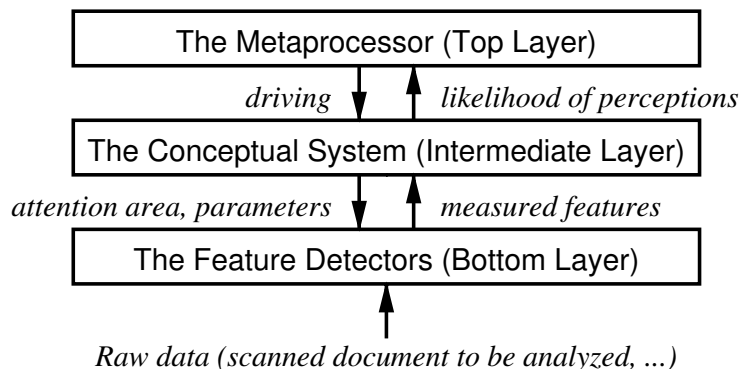


Figure 2.2: The grammar-driven OMR Framework proposed by Stückelberg [73]

As an example of the ‘top-down grammar-driven’ process, once note heads and stems have been found, secondary objects that are associated with notes—such as

accidentals and durational dots—are searched for, but only in the locations that the attributed grammar describes these objects as being in relationship to notes.

At the top layer, the ‘meta-processor’ keeps track of hypotheses about object classifications and relationships, and decides which set of hypotheses best describes the input image. It continuously decides whether to refine a hypothesis or to try a different hypothesis. Each hypothesis is given a probability, which will also depend on the probability of other hypotheses (for example, “there is a treble clef at the start of the first stave in the first staff system”).

The system assigns probabilities that a particular region contains particular objects, based on the syntax described in the grammar. For example, if the grammar leads to a hypothesis that there could be a notegroup in a particular place, the system would try to calculate the probabilities that there are vertical lines and note heads in the appropriate positions. The model that results in the lowest uncertainty rating over the whole image is accepted as the representation of the score; the grammar driven approach tries to find the “most likely sequence of symbols that might have generated the bitmap image” [72].

Primitives such as straight lines and noteheads are identified using a Hough Transform and a “low-band filter”, which means that the image is re-sampled to a low resolution to find solid shapes like filled noteheads. One advantage of using a Hough Transform is that it is resilient to segmentation—shapes are still found regardless of whether or not they are broken up or touching each other. Projections are used to find line endings, and a “Markov Model” is used on the straight lines to detect dashed and dotted lines. Horizontal and vertical cuts are used to locate staff systems, measures, staves, and connected object groups. The grammar will look for things such as durational dots only where they are allowed to be positioned relative to the found noteheads.

One proposed advantage of this design is that primitives do not need to be matched against pre-defined templates, only against a set of features describing an object type. However, the design described in these published works appears to be computationally expensive, and it seems that the model was proposed and only partially implemented for a simple grammar as proof-of-concept.

2.1.9 The MOODS Project

The MOODS project, based at the Università di Firenze (University of Florence) in Italy, is researching the use of computing technology to aid performers of music. MOODS is an acronym for Music Object-Oriented Distributed System. A large part of their research involves implementing digital stands for orchestras—that is, replacing the stand that holds sheet music with a digital display, for coordinating the display of individual parts to each instrumental voice. It allows “automated” page turning, although the pace is controlled by a person, and cooperative visual editing of music notation [10]. For example, the leader of a section such as “second violins” can add performance markings to the score that goes to all of the stands used by the members of that section.

One of their subprojects, called Object Oriented Optical Music Recognition (O³MR), concerns the use of OMR. Several papers [48, 11] describe the extensive use of projections for locating symbols on the staff systems. A neural network is then used on the located symbol areas to identify primitive shapes. However, other than a few published papers, the sub-project seems to have disappeared.

The principal researchers of the MOODS project are now associated with another project, the Interactive Music Network (described below).

2.1.10 Interactive Music Network

The Interactive Music Network¹ is a pan-European collaboration between various institutions, applying technology to music. This large-scope project, set up in 2002, covers many areas, but its main activities are listed as coordinating working groups of experts, organising meetings and conferences, and identifying and stimulating new business models and services. One of these working groups, the Music Imaging group, is concerned with various aspects of music scores and manuscripts, including OMR. Participants in this working group (or in the open workshops) have included authors and projects previously mentioned in this chapter, including the MOODS researchers, Kia Ng, and Bertrand Couïasnon.

This working group has produced dozens of reports and publications related to music imaging and recognition since inception, covering topics such as overviews of OMR products and systems, digital image acquisition (including available hardware products, such as scanners) and best scanning practices, and discussion on archiving

¹<http://www.interactivemusicnetwork.org/>

formats for images of sheet music. One of their publications [56] gives a general review of music imaging, with a large section on OMR. That section gives an overview of OMR Systems, and mentions the difficulty in comparing products. These difficulties include:

- Products requiring input images to be in disparate file formats, or acquire an image directly from an attached scanner.
- Many products only interface with a notation product, making standalone evaluation of the OMR-only functionality difficult or impossible.
- For products that are not integrated into a notation editor, different output file formats are used—there is no standard file format in use (such as NIFF, described below).

This paper also describes their method for measuring the results of OMR systems at 3 levels: primitive-level, note-level, and score-level. This contains three images to be processed by systems, and a questionnaire for human-evaluation of correctness [18]. The paper describes the results for three OMR systems evaluated by a group of 13 people, and the systems can be compared based on:

- “basic symbols recognition”—for example, the number of primitives correctly identified, undetected and mis-detected.
- “symbols and relationships reconstruction”—for example, the number of notes with correct pitch, duration, and accidentals, and correctly grouped.

2.1.11 Other Systems

ROMA

ROMA (Ancient Music Optical Recognition) is a project at the University of Lisbon, Portugal. This project involves the OMR of hand-written Gregorian Chant notated music (described later in this chapter), and uses a decision tree for classification of primitives [64]. Each node in the decision tree is a separate classifier, capable of generating and testing primitive features. Objects are clustered and classified based on examples in a set of training data.

2.1.12 Commercial Systems

As well as the work that has come out of the research community, there are also many commercial OMR systems on the market. Because they generally do not disclose their methodologies, they are not as useful for study, although they represent the state-of-the-art. Several of the more popular systems are Sharpeye, Midiscan, SmartScore, and Sibelius PhotoScore.

A more comprehensive list is provided by Ng *et. al.* [56].

2.2 Notation

Almost all notations currently in use are *mensural* notations; different glyphs are used to convey note lengths. This section gives some background to music notation and general considerations for OMR.

2.2.1 Western Music Notation

The image shows a musical score for Johannes Brahms' 'CAPRICCIO Op. 116, No. 7'. The score is in 2/4 time and is marked 'Allegro agitato' and 'f ben marcato'. It consists of two systems of two staves each. The first system starts with a treble clef and a bass clef, followed by a key signature of one flat (B-flat) and a time signature of 2/4. The music features complex rhythmic patterns and dynamic markings.

Figure 2.3: An example of Western Music Notation

Figure 2.3 shows an example of Western Music Notation (WMN). A score in this notation consists of *staff systems*, with each system consisting of one or more *staves*; in this example, there are two staves per system. A system starts with a *clef* in each staff, followed by a *key signature* in each staff (which consists of either flats [b] or sharps [#], or it may be empty). The first system also has a time signature,

consisting of two numbers, which describes how many beats there are per bar and must be the same for all the staves in the system. In the example, the upper staff has a treble clef while the lower staff has a bass clef, and both have the same key signature, consisting of a single flat. The key signature for the example score is $\frac{2}{4}$, meaning there must be the equivalent of 2 crotchet beats per bar. As well as the musical notes and features, the score also contains performance instructions (such as the tempo — *Allegro agitato*, and the dynamic markings for volume — *f*) and, especially on the first page, metadata such as the composer, title, and work number.

Western Music Notation has evolved over time from the earlier *mensural* notations (described later), and is still evolving to account for changes such as new constructs, new instruments and instrument capabilities; there is no definitive set of symbols or set of rules describing those symbols. A more thorough description of Western Music Notation can be found in many resources. For example, Stone [70] describes in detail the many rules and heuristics used by engravers for the positioning, sizing, and spacing of objects. It is defined by its usage, based on conventions used by composers and engravers. The physical layout of hand-engraved sheet music is designed, foremost, to be readable to a musician reading it. Spacing and positioning of glyphs is important—some glyphs may be moved or modified from conventional to create enough space to aid the reader’s eye.



Figure 2.4: The traditional hand-engraving process [39]

Figure 2.4 shows how a skilled engraver creates sheet music plates for a printing press; the mirrored and sunken image is carefully created so that after being filled with ink and pressed onto paper, the resulting score is clear and readable. This obviously requires much skill and experience at both typesetting and physical engraving.



Figure 2.5: The upper ledger line is shortened for aesthetic reasons

Figure 2.5 shows an example given by the Lilypond [60] authors; ledger lines (for notes above or below a staff) may be shortened by a typesetter to provide extra space if needed by a nearby accidental. In this example, the upper ledger line is shortened so that the accidental can be placed closer to the notehead, while maintaining a sufficient gap. The shape and style of some symbols has evolved, as well as varying slightly in appearance between different publishers, so there is no ‘definitive’ form for any particular primitive type. A brief example of both different shapes and different styles of crotchet rest primitives and bass clef primitives was shown in Figure 1.4.

In addition to the above customs, a skilled engraver will occasionally break with convention (for example, unusual or innovative note layout) if it results in a more readable score. Many of these manual tweaks by the engraver are subtle enough that a reader will probably not even consciously notice them. In light of this, an OMR system needs to be flexible in the shapes that it recognises and the interactions between symbols that are accepted as valid music. These examples demonstrate that an OMR system must be designed to be flexible in the objects it can recognise and in calculating the semantics of those objects. Objects of the same type may have significantly different shapes, even on the same sheet of music. However, while semantic rules can be devised to capture the meaning of a large majority of conventional scores, it is infeasible to formally create a set of rules that can cover every scenario used by publishers and engravers [20, 21].

2.2.2 Other Notations

Guitar Tablature

Guitar Tablature consists of 6-line staves, representing the 6 strings on a guitar, with digits super-imposed to indicate which fret on the guitar's neck to use for a particular note. Durations are implied based on the spacing between digits. As well as notes, other symbols can appear; this includes objects that can affect notes (such as slides, bends, tempo and dynamic changes, or pauses) and other symbols that do not directly affect notes, such as lyrics. An example image of music in guitar tablature notation was given in Figure 1.2.

Because of the limited set of symbols involved, performing OMR on guitar tablature notated sheet music should be no harder than performing OMR on Western Music Notation scores.

Gregorian Chant

Gregorian Chant (sometimes called Plainchant) notation was the precursor to modern Western Music Notation. It was eventually standardised on a set of symbols conveying pitch information, drawn on a four-line staff. A large amount of Roman Catholic church music (up until the early Twentieth century) was published in this notation.

Different shapes are used depending on whether the melody is ascending or descending, and dependent on the number of notes in the group in addition to the note position on the staff showing pitch information. Flat symbols are also occasionally used to modify a note's pitch.

The notes do not convey durational information as rigidly as modern notation; where a pitch is repeated or is both the last note in a group and the first note in the following group, the notes are generally combined. Later additions to the notation included some symbols that are used to modify notes by adding a pause or a slight extension, or to double the note duration.

White Mensural notation

White Mensural Notation was one of the intermediate steps in the evolution from Gregorian Chant to current-day Western Music Notation. It was predominantly in use from *circa* 1400 to *circa* 1600, and is named due to the use of hollow noteheads.

Figure 2.6 shows an example of music in this notation; the top staff shows a historical layout, the centre staff shows the music using a contemporary style, and the lower staff shows the corresponding Western Music Notation representation.



Figure 2.6: An example of White Mensural Notation
(Reproduced from Lilypond [59])

Fasola Notation

The Fasola notation has its origins in an eleventh century system where each note is given one of six syllables, eventually simplified to the four syllables *fa*, *sol*, *la* and *mi*. This system was popular in the colonial United States of America, and these syllables were given different shapes in a system devised by Little and Smith, to aid in teaching notation to unskilled singers. “The Sacred Harp” was a popular Nineteenth century book published in America, and therefore sometimes this notation is referred to as the Sacred Harp notation.

Each pitch has one of the shapes, and all notes at that pitch are given the same shape. Other than note shapes, the notation is similar to Western Music Notation. An example is shown in Figure 2.7.

Non-Western Notations

There are many other music notations in use, including various Indian notations and other Asian notations. These are generally not staff-based, often using an Asian script to describe the notes with syllable names. Some of these forms of music divide an octave into a scale containing more notes than Western music, meaning that Western notation is not capable of adequately representing it. For example, the most widespread Indian notation (an example of which was given in Figure 1.3

LOVED ONES OVER YONDER.

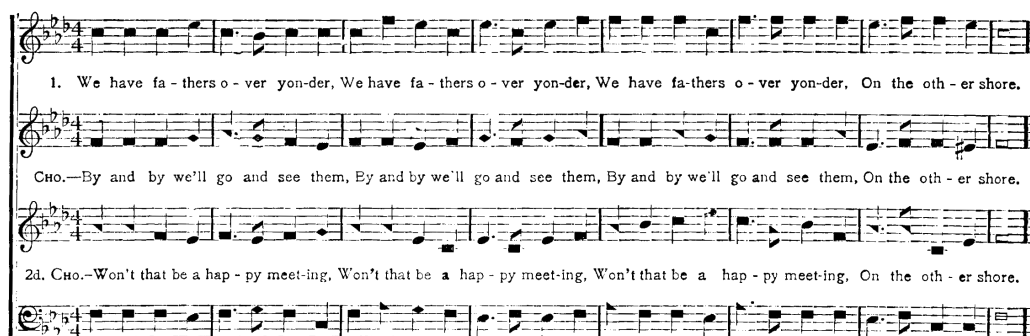
161

"And they ascended up to heaven in a cloud."—REV. 11; 12.

Key of F Minor.

(Old Revival Song)

ARR. BY S. M. DENSON, 1912.



1. We have fa - thers o - ver yon - der, We have fa - thers o - ver yon - der, We have fa - thers o - ver yon - der, On the oth - er shore.

CHO.—By and by we'll go and see them, By and by we'll go and see them, By and by we'll go and see them, On the oth - er shore.

2d. CHO.—Won't that be a hap - py meet - ing, Won't that be a hap - py meet - ing, Won't that be a hap - py meet - ing, On the oth - er shore.

Figure 2.7: An example of Sacred Harp notation

on page 3) consists of rows of note syllable names.

2.2.3 Music Notation Software and File Formats

There are many software packages available for notation sheet music and/or sequencing music; too many to list here. Manually entering and re-arranging music in such a program is time-consuming; this is the primary motivation for OMR. This means that an OMR system needs to create data files in an appropriate format for the software package.

There are a myriad of computer file formats available for storing various levels of music notation; one fairly comprehensive source lists over 60 formats [24]. While some general purpose graphics formats (like Scalable Vector Graphics) or page description languages—such as PostScript or Portable Document Format (PDF)—can be used for the graphical presentation of music, there are many formats written specifically for representing some set of music features, often developed for a particular piece of software. While many formats have been developed or proposed for music notation [67], none has become a widely used industry standard, partly because different areas of music information retrieval need only certain aspects of music in their representations. For example, some systems may only need enough symbolic information for an audio representation, while this would be insufficient for symbolic notation or graphical notation uses. Also, commercial software developers sometimes feel it is in their best interest to not interoperate easily with competitors' products.

MIDI

MIDI has become a very widely used format since being created in the early 1980s. However, while MIDI can represent the performance of a piece of music (effectively “play this note at this volume for this duration”), it is not adequate for storing non-audio performance data, such as performance markings, clef, key and time signatures, or other graphical information.

NIFF

NIFF was an attempt at an industry-wide standard for interchange between different notation programs. NIFF [38] was designed in the mid 1990s by a consortium including music software companies and university researchers. The goal was to create a common, open format that would allow interchange of data between different music programs. The most recent revision of the standard was in June 2002. Some products available today can import and/or export NIFF documents. However, NIFF did not achieve widespread adoption in the commercial software arena, and the format specification is now unmaintained.

Guido

The GUIDO music file format [41] was first prototyped in late 1992 as part of the SALIERI project, which was researching various aspects of music processing. The “core” part of the GUIDO format has been stable since around 1996, and was designed to hold enough information to be able to formulate both the audio and notational aspects of the music it represents. Unlike NIFF, files in the Guido format are written in plain text, meaning that the files are readable by people and editable by hand. Similarly to the NIFF specification, Guido did not achieve widespread support in commercial music notation systems, although it gained a limited following in some teaching and research areas.

MusicXML

MusicXML is a format designed for interchange between computer programs. The first released stable version of MusicXML was published in early 2004. It is supported to varying degrees (such as import-only, export-only, or both) by many of the popular music notation software packages—both commercially-produced proprietary software and open-source programs—as well as having software libraries

available to convert other notation formats such as the closed formats Nightingale, Finale, MuseData, and the open formats NIFF, Guido, and Rosegarden.

2.3 Related Fields

This section discusses fields of endeavour that have similar problems and challenges to OMR. Although each has domain-specific focal points, some techniques used may be general enough to be of use in OMR.

2.3.1 Optical Character Recognition

Much research has been performed into Optical Character Recognition (OCR) of image of printed text. Text has several properties that aids in OCR:

- text is arranged in horizontal rows, so there is a predictable sequence of character locations.
- English text has a fairly limited alphabet—twenty-six characters which have upper-case and lower-case variants, along with punctuation markings and other symbols.
- These symbols have an invariant shape; other than changing font or font size, all instances of a particular character or punctuation should look very similar.
- In English, most symbols consist of only a single glyph. Only several symbols such as lowercase ‘i’ and ‘j’ and some punctuation marks consist of two or more separate glyphs.
- The characters are rendered distinctly; they do not overlap. Exceptions, such as ligatures for character sequences like “ff” and “fi”, can be handled on a case-by-case basis.

One difficulty in OCR is recognising texts containing multiple fonts. Simple features are not always sufficient to distinguish between a character in both serif and sans-serif fonts. Serifs are the small ‘hooks’ on the edges of characters: “H” compared to “H”. The accuracy of OCR can be improved by using domain context; for this domain (English text), sources of knowledge can include a dictionary of known or allowed words, and knowledge about expected character frequencies such as examining “*n*-grams”—singleton characters, pairs, triplets, and so on. For example,

“e” is the most frequent letter in English text, and “th” is the most frequent pair, and this knowledge could be used to make better decisions when working with unknown data. Syntax checking (sentence structure) and semantic checking (word meaning) have been also been proposed and used to help with the classification of characters. This requires *a priori* knowledge of a document’s language, although often this can be automatically determined based on the recognised character frequencies. However, recognising documents that are multi-lingual can be problematic.

OCR of handwritten text has also received much attention [68, 46]. Handwriting is more difficult to recognise, due to different handwriting styles between authors, the tendency for characters to be joined together in cursive handwriting, and different shapes for the same character, even within the same text.

More information on OCR can be found in many collections of papers (for example, Document Image Analysis [61], and the biennial ICDAR series [1, 42, 2]).

Relevance to OMR

Some of the low-level feature analysis of shapes used in OCR processing could be useful for classification of symbols in OMR. The methods used for checking and correcting symbols based on the grammatical and semantic meaning of sentences may offer insights of how this could be done with musical symbols. However, text is essentially a one-dimensional list of characters, so any data representation techniques used in OCR may not “scale up” for use in OMR.

2.3.2 Chinese Character Recognition

Recognition of Chinese (and similarly, Japanese and Korean) text is problematic because of the large number of characters. Another complication is that characters can consist of multiple, non-touching fragments. This means extra effort is needed to group glyphs together during classification.

As with OMR of text in Latin languages, research into OMR of Asian text would primarily be relevant to the lower-level object recognition phases of music recognition; again, this is due to the essentially one-dimensional aspect of characters in forming sentences. However, there may be some applicable methods relating to the use of object semantics for recognising characters in context based on the surrounding objects.

2.3.3 Document Image Analysis

OMR is one field in the more general area of Document Image Analysis (DIA). There are a multitude of DIA domains—some of which are specialisations of OCR—and many of them commercially driven. Example domains include:

- address recognition on envelopes (for postal systems)
- form recognition (for archiving bureaucratic data and records)
- Geographic Information Systems (maps and aerial/topological photographs)
- weather modelling (such as infrared satellite photos of clouds)
- medical imaging (for example, 2D and 3D data from ultrasounds)

Many of the lower-level recognition techniques used in OMR are not specific to music, and can be performed adequately by algorithms developed for other DIA fields. These methods include any preprocessing and filtering on the raw input image—such as thresholding to turn a greyscale image into a binary black-and-white image, and enhancements to remove noise or correct brightness and contrast settings—as well as general pattern recognition techniques such as template matching or feature extraction and comparison.

Of course, these techniques should be adapted to take advantage of instances where music domain-specific information can be exploited. Staff-lines are an obvious cue to use for splitting the image up into separate regions and for detecting and correcting page skew. At a higher level, rules about musical syntax can be used to confirm primitive classifications, or used to restrict the types of primitives that may be found in a specified region based on the previously identified objects.

2.4 General Trends in OMR

There is general agreement on the low-level processing stages used for locating staves and objects; some common steps used by many OMR systems are:

1. Performing general image clean-ups and locating staff systems.
2. Locating objects on staff systems; typically, staff lines are located by a horizontal projection and then removed, or objects are found without first removing staff lines by using a ‘low pass filter’, projection cuts or another technique.

3. Using a grammar to determining musical objects from the basic recognised primitive shapes.

In the 1990s, there were several proposals and attempts to use Artificial Intelligence (AI) approaches to make more use of semantic knowledge during graphical primitive recognition. However, these approaches seem to be computationally expensive, and only partially implemented. Generally, these approaches create internal models from a set of training examples, and these models tend to be rather inflexible for understanding input that does not closely resemble the training data.

Using semantic rules to drive the recognition process can arguably lead to worse accuracy—for example, if accidentals are only searched for near noteheads, then misrecognising a notehead results in missed nearby objects that affect it. In contrast, trying to independently recognise all primitive shapes first, and then using semantic information to correct errors or confirm assignments, means that the nearby objects could be recognised and then used to infer the missing notehead. The question is which approach is more robust; the latter might lead to more falsely recognised objects.

The major strengths and themes of the surveyed OMR systems are:

- *extensibility*—modifying pattern descriptions and grammar rules without modifying the base system. For example:
 - CANTOR has external patterns that can describe arbitrary shapes, and rules that can describe the inter-relationships of arbitrary object types. Example patterns and rulesets have been created for different staff-based music notations.
 - GAMERA has an external database that is used for recognition (and can be trained on arbitrary data).
- automatic learning or *adaptation* of patterns for primitive objects.
 - GAMERA uses an AI approach for deciding which tests to perform on primitives for classification. (However, this process requires manual classification of training data for the system to determine which tests are appropriate for each classification type.)
- automatic inference of high-level semantic features.

- For example, Ng *et. al.*'s research into key signature and time signature detection, based on the semantics of the recognised objects.

The final stage of an OMR system should be a conversion from the system's internal representation to one or more external file formats. Depending on the end-user's needs, this might include graphical-only formats (for printing, for example), notation formats for other music-related software, or audio-only formats.

Support of relevant computer file formats can be problematic for an OMR system. There is no single industry standard format that perfectly matches the requirements of OMR, partly because different formats were invented for different purposes. It may be possible to convert from one notational file format to another via specialised software, but this is often imprecise, resulting in errors or loss of information. A better solution is to design the final stages of the OMR system in a way that facilitates the addition of routines for converting from the system's internal state to new file formats.

Chapter 3

Anatomy of an OMR System

The principal goal of an OMR system is to have a computerised version of the sheet music that encapsulates as much information about the music as possible. Some tasks do not require reproduction of all the information found on the score—for example, automatically creating an audio reproduction would only require note pitch, duration and volume information, while OMR for the purpose of re-publishing or typesetting a score would require as much notational information and performance indications (such as fingering or breathing marks) as possible. An OMR system should be able to support all of these tasks to be of maximal benefit.

This chapter describes the stages performed by OMR systems to achieve these goals, and illustrates some of the points with example cases from the experimental system. This description is given starting with the end result of the system, moving on to the lower level tasks required to complete this main goal.

3.1 Musical Semantics

The goal of a complete OMR system is to calculate the semantics of the sheet music so that the score may be recreated; a complete semantic model of the music can be saved to disk and used for the desired purpose, such as score editing, music information retrieval purposes, or computer audio synthesis. As discussed earlier in Chapter 2, there are different computer file formats suitable for different levels of representation. MIDI format files are adequate for a simple audio-only model, for example.

To build a model of the musical semantics, the system must have knowledge of the high-level rules of music notation. Different notations require different sets of

rules, so this stage of an OMR system should be flexible enough to allow arbitrary notation syntaxes to be expressed. These rules determine how all the objects found relate to each other—multi-page scores, text and systems on each page, objects in systems, and so forth.

Applying and checking semantics is fast compared to the image manipulations required for identification of graphical primitives—building and parsing a semantic lattice structure that represents the relationships between musical objects takes a fraction of a second on a modern desktop machine, even for a full page of multi-stave music. This means that more time could be spent analysing the syntax and semantics of the score, and there is the opportunity to use feedback to notify the lower levels of the OMR process when an inconsistency is detected.

3.1.1 Syntax versus Semantics

Strictly speaking, *musical syntax* refers to the validity of the symbols with respect to some set of rules (for example, “*this* type of symbol may only appear immediately after *that* type of symbol”), while *musical semantics* refers to the particular music expression represented by the symbols—what the symbols mean to an observer. This is comparable to language; words such as *verb*, *adverb*, *noun* and *adjective* describe the syntax of a sentence, while the actual expressive meaning that the sentence evokes requires semantic knowledge of the language. In this thesis, the term *musical semantics* generally refers to both the syntax of a score and semantics of the objects described by that syntax, since both are often part of the same task in an OMR system.

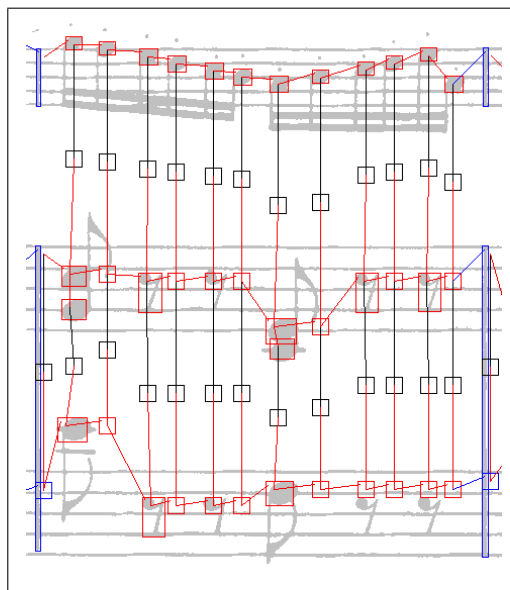
3.1.2 Syntax

For any piece of music, an OMR system needs a set of rules that describes the attributes of the contained musical objects, and how the objects relate to each other. A different set of rules will be needed for each notation that the input images may use. For Western Music Notation (WMN), musical objects can be divided into two broad groups. The first group contains those objects that are performed by a musician (and therefore have a duration), such as notes and rests; these can be described as “time events”. The second group contains those that affect objects in the first group; this contains modifiers such as time signatures, key signatures and accidentals, and performance indicators such as dynamic (volume) markings and

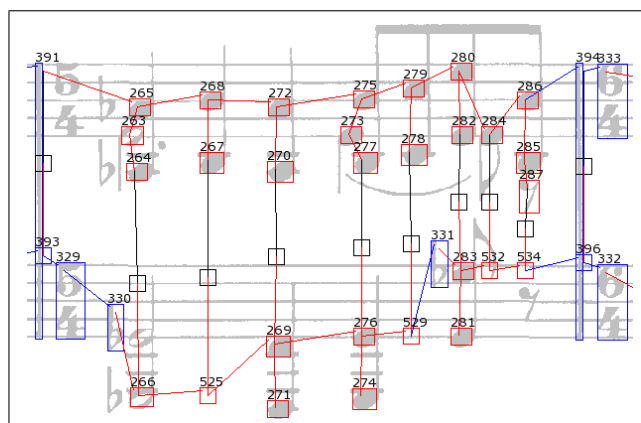
accents.

To understand the syntax, an OMR system needs to determine the temporal order and duration of objects, as well as other attributes such as pitch. Alignment along a vertical axis for time events is obvious for events that occur at the same time—even simple algorithms that use this alignment while parsing the 2-dimensional objects is accurate for most passages of music.

Figure 3.1(a) shows each division and sub-division of time—each box in the upper clef is half of a quaver beat. In the middle and lower clefs, each musical object should last for a quaver beat, so it is easy to check if the detected durations at each time segment are consistent between staves. Figure 3.1(b) shows an example where this



(a) Events occurring simultaneously are nicely aligned



(b) Simultaneously occurring events not aligned (nodes 282 and 284)

Figure 3.1: Examples of vertical stacking for simultaneous events

strategy fails. The notes shown in nodes 282 and 284 should occur at the same time, but do not align vertically. In this case, an incorrect time slice with a duration of half

a beat has been created (nodes 282 and 532). Further semantic processing would be required to correct this, although in this case it is easy to detect that a mistake has been made somewhere in the bar, as the bar’s calculated duration is now more than the 5 beats allowed by the time signature. However, when a bar has too many notes for the time signature, determining where in the bar the error has occurred is not necessarily straightforward. For this example, correctly determining the “split voices” in the bar—the notes with stems pointing up and with stems pointing down at the same logical time—might identify the possible cause.

3.1.3 Engraving

Music engraving is the process of determining the physical layout of sheet music in preparation for publishing. This term is derived from when the music was manually engraved—mirrored and inversed—into a metal plate for a printing press. Automated page layout of music by a computer, sometimes referred to as automated engraving or automatic justification of music, has undergone much research [60, 17, 9, 65]. As discussed in Section 2.2, there is no definitive set of rules for WMN—music engraving is more of an art than a science.

For OMR, knowledge about engraving can be used in the recognition process—many of the same rules can be used (in reverse) to see the relationship between semantics and the position of objects relative to each other. For example, the horizontal spacing between objects should give an indication of relative durations, while the width of bars does not have to be consistent from one bar to the next. This information can be used to double-check the recognised durations, and to suggest alternative objects if the bar does not semantically “add up”. Intuitively, longer notes should be given more horizontal space—Bellini and Nesi suggest using a logarithmic scale [9].

Professional engravers do not divide the space up evenly between noteheads, but also take into account the direction of stems. Figure 3.2 shows an example of spacing as calculated by the Lilypond typesetting program; Figure 3.2(a) shows the result when the noteheads for notes of the same duration are evenly spaced, while Figure 3.2(b) demonstrates Lilypond’s algorithm accounting for the space between the stem lines [59], as a professional engraver would endeavour to make a score more aesthetic for readability.

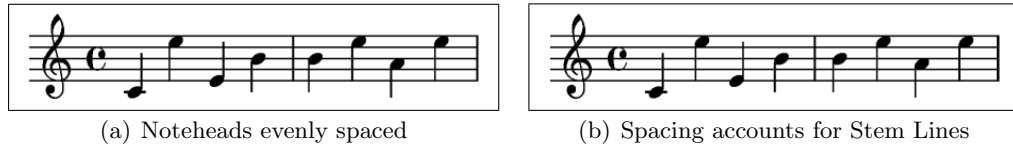


Figure 3.2: Spacing between notes

3.2 Object Identification

Object Identification is responsible for classifying the graphical shapes on the page. These are then processed by the Semantics stage to determine and describe the syntax and interactions of the identified musical objects.

Object Identification is often split up into smaller stages; example tasks include segmentation (separation of touching objects), classification of object shapes, and assembly (joining multiple shapes into a larger musical object). These tasks may be referred to as working with *primitives*—that is, low level objects that are not musical objects in their own right, but are more practical to work with (for example, simpler shapes that are easier for the pattern recognition routines to identify).

3.2.1 Segmentation

Symbol identification in OMR applications is made more difficult by the fact that many symbols overlap or touch. This contrasts to Optical Character Recognition of printed Western text, where letters infrequently overlap. When symbols overlap or touch in a score—resulting in a single large unknown object—it is either because some symbols are supposed to be joined in that notation, or it is accidental and caused by careless typesetting in a printed manuscript or by poor settings during the digitising process which captured the image. An example of the former is in Western Music Notation, two or more quaver notes next to each other are normally joined by a horizontal beam while single quaver notes have a tail. Accidentally touching objects may be caused by bad brightness and contrast settings when scanning a printed score, resulting in a darker image.

There are two approaches for recognising symbols that are part of a larger object; either look for matching symbols regardless of connectivity, or try to fix connectivity problems before looking for symbols. For the former, this would mean looking for the smaller object types within the larger connected area. For the latter, this would mean using image processing techniques (possibly including notation-specific settings or algorithms) to split the larger objects up into the primitive components.

An example of the latter method uses projections to determine the various notes, vertical lines, and beams found in connected note groups [48].

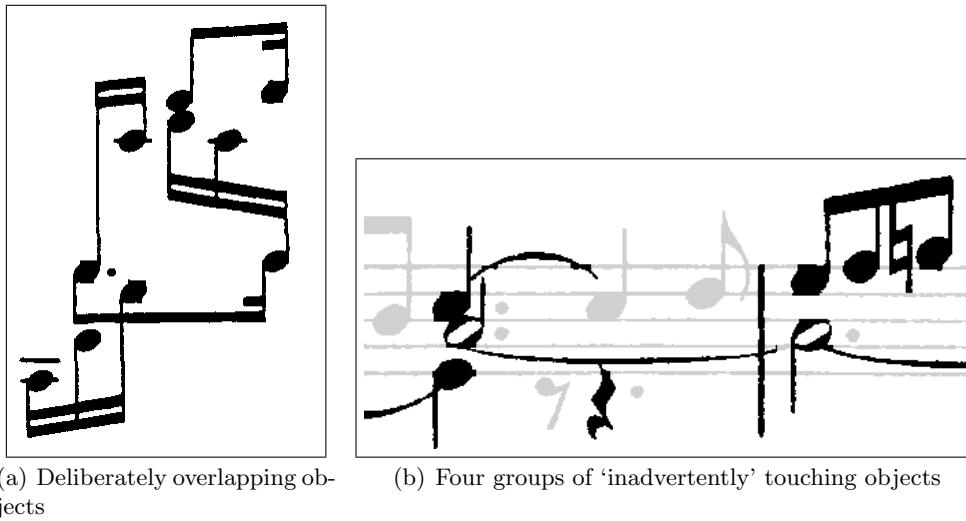


Figure 3.3: Examples of overlapping shapes

For the first cause of joined objects (that is, inherent to the notation), it should be possible to determine which object types may be joined to other types, and use one of the previous approaches to recognise those types. The second cause of joined objects (typesetting errors and digitisation errors) is more problematic, as this can potentially occur to objects of any type. Figure 3.3 shows examples of both causes; Figure 3.3(a) shows a rather complicated layout with the lowest notegroup overlapping the middle notegroup, which also touches the uppermost notegroup. (This could be mitigated by accounting for this when looking for noteheads, vertical stem lines, and beams.) In contrast, Figure 3.3(b) shows several objects joined into groups due to the typesetting. While the noteheads are expected to touch other objects, and should have recognition routines that account for this, crotchet rests and natural symbols are expected to be ‘isolated’, and not joined to other objects.

There is also some relevant research in OCR that could be applied to OMR, especially in the automatic segmentation of handwritten text and Asian languages (for example, in [2]), which has been an active area of research in the last decade.

3.2.2 Classification

In general, graphical objects can be divided into two classes: those that always have the same shape and proportion, and those that vary in appearance. Figure 3.4 shows examples of objects that have a changing shape in Western Music Notation—slurs (and ties) and decrescendo markings may cover an arbitrary number of notes.

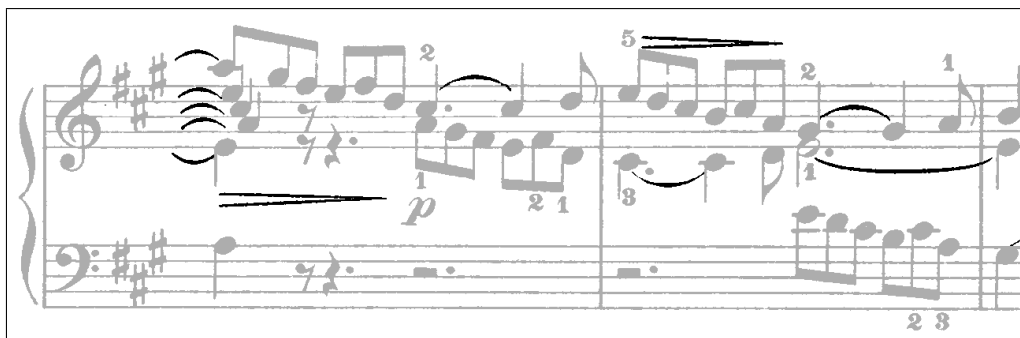


Figure 3.4: Primitive objects that can vary in shape

For Western Music Notation, examples of the former class include digits, text, noteheads, rests, and accidentals, while examples of the latter class include note-group beams, slurs and ties, crescendo/decrescendo markings, and note stems.

Some features that could be used for comparing objects to known, ideal example objects include:

- the size of the object's bounding-box
- the ratio of black-to-white pixels in the bounding-box
- the maximum and minimum width-to-height ratios
- horizontal and vertical projections
- Hough Transforms
- template bitmap match, including variations such as weighted template match
- connectivity analysis (checking whether a path of black pixels exists between two particular points)
- vectorisation or thinning algorithms
- the 'moment' of a bitmap, which can be used to measure the distribution of pixels or the centre of mass.

This creates a multi-dimensional feature vector of values for each graphical object. These vectors can be compared for similarity to the vectors of ideal templates by using the multiple nearest neighbour (k -NN) method. This is used by both Ng [55] and Fujinaga [36]; the latter also makes extensive use of the bitmap moment as one of the main features for comparison. *Decision trees* are another method that can be used for classifying objects based on a set of different tests. Popular algorithms for implementing these include ID3 and C4.5 [77]. These rely on examples

being used as training data to build the decision tree before being used on new, unknown objects.

These techniques have varying levels of effectiveness for poor quality input images. For example, if an image is scanned at a low resolution, or with too much brightness, single musical objects might be broken up into multiple graphical objects separated by some white space. Ideally, the identification process will be able to account for this, either by having recognition routines that are invariant to noise, or by attempting to repair or enhance the input image before performing object recognition.

3.2.3 Assembly

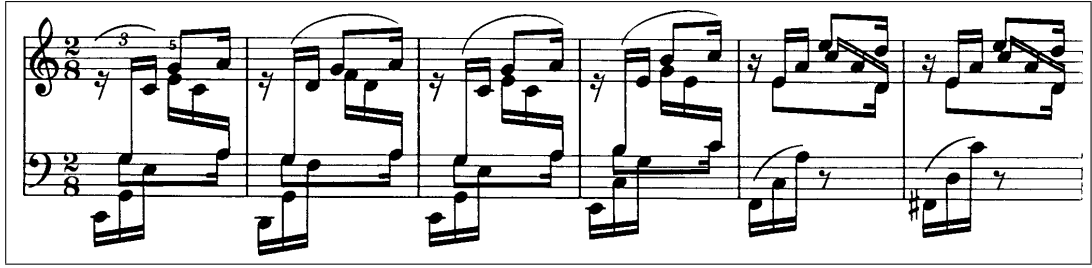
Sometimes it is easier to recognise individual components of a musical object separately, then join them together afterwards. In this thesis, the term *Primitive Assembly* is used to describe this. For example, recognising noteheads and vertical lines separately and then assembling them into a note is arguably more robust than trying to recognise all possible chord arrangements in the image.

In some cases, there is an arbitrary separation between assembly and semantics—for example, joining durational dots, accidentals, and accents to noteheads could each be viewed as being either an assembly or a syntactic/semantic action. Another example is differentiating between a semibreve (a hollow notehead that has a duration of four beats) and a minim (a hollow notehead attached to a stem line that has a duration of two beats); assembly joins noteheads to any possible stem lines, but calculating the durational meaning of that is probably a semantic quality.

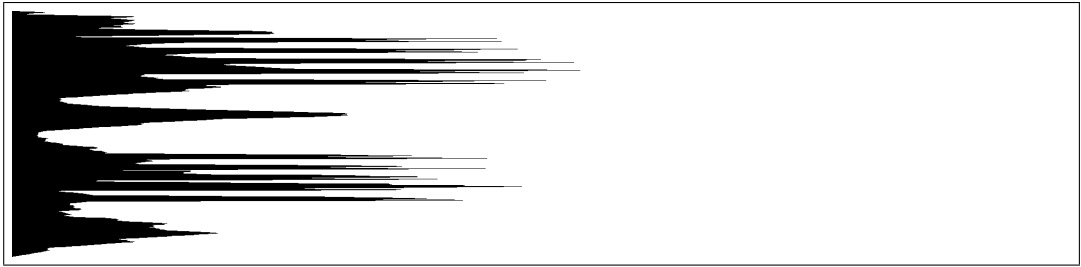
3.3 Staff Processing

The majority of OMR systems remove staff lines, with their later identification routines accounting for any damage that this procedure may have caused. Of course, not all music notations involve the use of staff lines, so an extensible OMR system must not rely on staff lines being present.

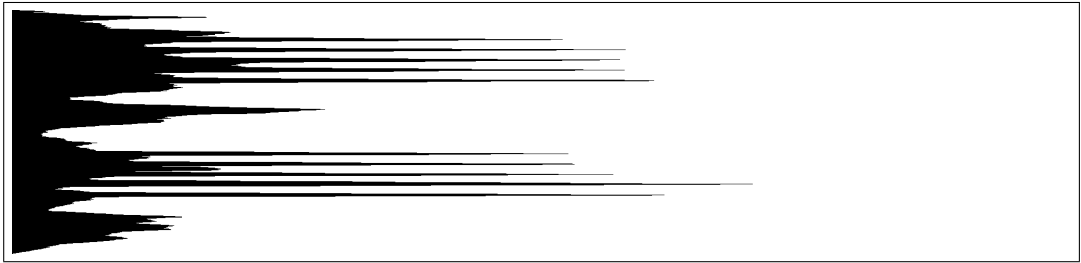
The consensus is to perform a horizontal projection—projecting the x -axis onto the y -axis—to determine the peaks corresponding to the long horizontal lines. Incidentally, this method can also be used to detect small angles of skew. Figure 3.5 shows this for an extract of a page of sheet music that was scanned in slightly off-centre. Figure 3.5(b) shows the projection, with peaks corresponding to the



(a) Staff system as scanned (slightly skewed)



(b) Horizontal projection of skewed staff system



(c) Horizontal projection of de-skewed staff system

Figure 3.5: Finding staff lines by projection

horizontal runs of black pixels. Figure 3.5(c) shows more pronounced peaks of the projection after the extract was rotated to remove its skew.

Although most implementations extract the staff lines, some earlier work did not, notably the Line Adjacency Graph method used for OMR by Carter [22]. This is a method for identifying primitives based on consecutive vertical runs of pixels, and therefore is able to ignore the long horizontal staff lines.

3.4 Text Processing

A complete OMR system needs to locate and identify any text on the score. Text, like musical notes, is needed to help convey the composer's intentions to the performers for realisation. Text can even be an integral part of a music notation, such as the fingering numerals that appear in tablature notations.

In Western Music Notation, common uses of text are:

- performance-only instructions — for example: dynamics, tempo, fingering

hints.

- note and score modifiers (such as ‘octave’ markings, repeat marks, coda markings).
- metadata — identifying features of the score, such as title, composer, editor, and publisher.
- instrument names (for multi-part and multi-voice scores, useful for aligning staves on systems, as well as being metadata).
- lyrics for vocal scores.

3.4.1 Identifying Textual Regions

Text has some properties that helps to identify regions of interest — for Western languages, bounding boxes of characters are generally squarish and roughly the same size, in a horizontal line, and divided into words (that is, sequences with small inter-box spacing, with larger space between words). Asian languages have similar properties that help identify characters — the bounding boxes of Asian characters are generally square and the same size, and are in rows or columns with even spacing between characters. One complicating factor is that many characters are composed of a number of smaller non-touching glyphs, so lots of small objects rather than one large object might be initially outlined. An example of this is shown in Figure 3.6.



Figure 3.6: The Chinese character for “flute”, composed of discrete glyphs

3.4.2 Optical Character Recognition

For text in Western alphabets written in regular rows, it is fairly straightforward to identify the height, as there are four common vertical points: the baseline, the x-height, the caps-height, and the descender depth. There has been much work in the field of OCR; for example, see the ICDAR conference series [1, 42, 2].

Asian languages (sometimes referred to as “CJK” text, for *C*hinese, *J*apanese & *K*orean) require different recognition strategies to Latin-based texts, because of

different characteristics. The size of the alphabets are much greater, with thousands of characters. The characters are much more complicated than letters in Western alphabets, composed of many strokes that form many different geometric shapes. These characters can contain several non-touching components, while Latin texts only have simple accents or dots above certain characters. Asian text recognition is an active field of research [42, 19].

As well as text needing to be identified because it is an integral component of a score, many of the recognition techniques used may be transferable to music object recognition. For example, methods that can process the extra complexity of CJK text may be useful recognising musical objects that are similarly composed of multiple smaller components.

The use of OCR with the music recognition process is further explored in Chapter 5.

3.5 Preprocessing and Filtering

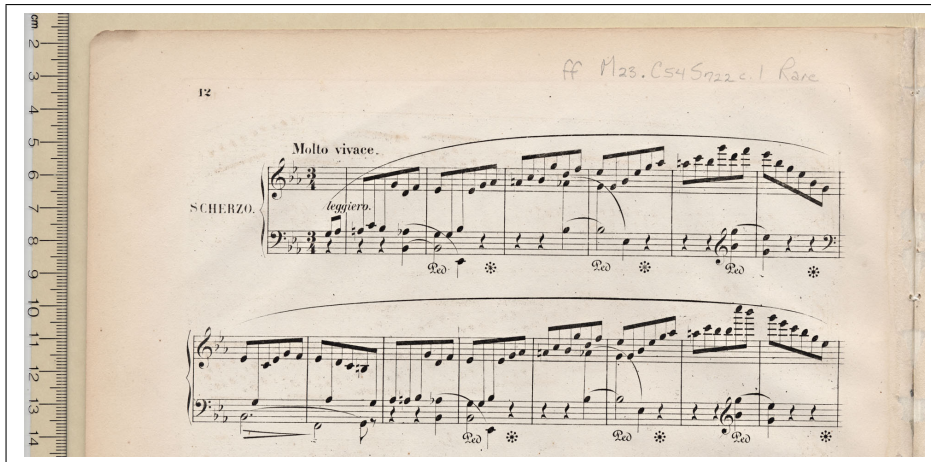
In image analysis domains, acquired images are typically “preprocessed” to correct defects before the main recognition routines are performed. This section describes some common transformations performed on the raw input.

3.5.1 Binarisation

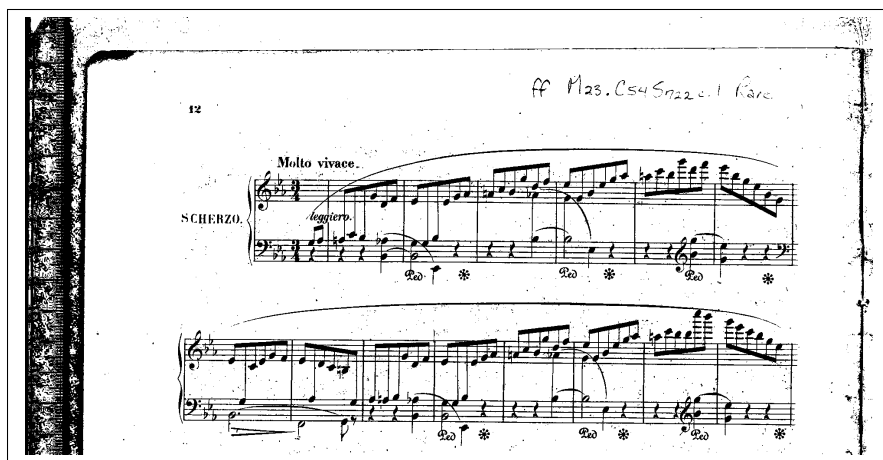
Although binary (black and white) images are easier to process (for example, for finding object edges), greyscale and colour images contain more visual information. Curves, corners, and diagonal lines look more smooth with greyscale or colour images; binary images suffer from *aliasing*, sometimes described as a “staircase effect”. Each pixel in a binary image can be in either of two states—black or white—while in a greyscale image, each pixel can typically be assigned a value between 0 and 255.

Converting from a colour image to a greyscale image, or from either of those to a binary image, requires *thresholding*. Figures 3.7(a) and 3.7(b) show the result of thresholding a scanned image from greyscale into a binary image. The resulting image has noticeable dots, sometimes called “speckle” or “salt and pepper” noise, that require further processing (normally called *de-speckle*) to prevent them from interfering with the object recognition process.

Because of colour variations, and changes in brightness while scanning, it is possible that what is an acceptable threshold in one region of the image results in



(a) Portion of a scanned image



(b) Image after Binary Thresholding from grey to black-and-white

Figure 3.7: Before and after binary thresholding

unacceptable quality in other regions of the page. One solution is *adaptive thresholding* [79], where a threshold for any particular area is based on the surrounding region.

3.5.2 Image Cleanup and Filtering

Most of the preprocessing steps performed on scores during OMR processing are general document image analysis techniques, and not specific to OMR. A brief overview of several common preprocessing actions is now given.

- ‘Pixel sampling’ can be used to reduce the image to a preferred resolution—that is, reduce the image size. This technique might not be necessary if the main processing routines are flexible enough to cope with arbitrarily sized images.
- Background noise in the image can be reduced or eliminated by performing a

‘de-speckle’ routine. The simplest form of this is to merely remove all objects that are smaller than some threshold, such as nine pixels in area.

- Various algorithms can be used to correct an image if the brightness and contrast settings at capture time resulted in an image being too light or too dark.
- Correction for page skew and deformation introduced when the image was captured. Methods for performing this are described in detail below.

These techniques are well known, and described in many sources, for example in the document image analysis area [19, 66, 61].

3.5.3 De-skewing and Deformation

De-skewing refers to the process of rotating the input image to account for any skew introduced by the digitalisation process. Although a person manually placing a sheet of paper onto a scanner can get an image with only a few degrees of skew, image processing algorithms will perform more accurately if there is as little skew as possible.

Coping with skewed input images is not music-specific, and there has been much research in the document image analysis fields into different techniques for detecting and removing skew [4]. However, OMR may be able to take advantage of unique qualities of a particular notation. For example, Western Music Notation uses staff systems consisting of parallel horizontal lines, which can easily be used to detect any skew; common methods for doing this include horizontal projections and Hough Transform [62], and both of these are described later in Chapter 5.

Deformation, such as distortion introduced when scanning from a tightly bound book, is harder to detect and correct, partly because the deformation is not necessarily uniform for the whole page. This can be detected by examining the long horizontal stave lines for slight curvature. The CANTOR system [5] corrected curvature such as this (once it is accurately determined) by performing a shearing operation to move each vertical column of pixels up or down, and calculating the amount of deformation in three locations in case it was more deformed at one end than the other.

Chapter 4

Coordination of Knowledge

Sources

The preceding chapter discussed the general tasks performed by a theoretical OMR system, and outlined some of the techniques used to perform these tasks. The aim of such a system is to have an internal representation of the music's syntax and semantics, as well as any metadata about the music, in a computerised form which can then be used for other purposes. Each processing task can be thought of as adding knowledge about the representation of the music, contributing to the internal state of the system.

Intuitively, having more information available should lead to more accurate decision-making. This chapter describes different methods for controlling the interactions between the various specialised stages, representing knowledge in a way that facilitates the interactions, and dealing with uncertain and contradictory data. Making better use of information calculated by these tasks will improve the quality of the OMR process.

4.1 Advantages of a Coordinated Approach

The aim of using a coordinated methodology in an OMR system is to improve the accuracy of the system's output. This should be achieved by the system modifying both its data and behaviour as better information becomes available. Such a system should:

- allow knowledge sources to examine and modify any object (or data about an object); and

- keep a history of decisions, so that they may later be undone (and possibly redone) as new information is uncovered.

In addition, there are other desirable qualities that a recognition system in general should have; for the work described here, extensibility is important. That is, the system should have flexibility in accepting and representing *a priori* knowledge (such as rules describing pattern shapes or notation syntax) so that rules can be easily changed to accommodate new object types or music notations. Also, this research has another aim; examining the behaviour of such a system. This means that the ability to see what the system is doing should be taken into account.

4.1.1 Dealing with uncertainty

Attaching a certainty rating to classifications rather than having a simple model with only single Boolean (true or false) decisions has a number of advantages:

1. This allows a system to store alternate hypotheses at the same time, instead of only one.
2. Related to the previous point, this is useful when trying to decide whether or not to change the state of an object when given new information, as the system can check which classification makes the most ‘sense’.
3. Having some sort of certainty rating allows the system to decide if it is making progress towards its goals, based on whether the internal state is becoming more or less ‘certain’ overall.

These advantages come at the expense of increased complexity in model design and program runtime. The third point requires uncertainty to be quantitatively measurable so that different states can be compared to see which is ‘better’.

The different knowledge sources can be used to help confirm hypotheses made by other sources. For example, the lower-level graphical recognition routines find vertical lines, the musical syntax rules will determine that some of these vertical lines appear to be barlines (based on their height and position on the staff), and musical semantics can determine if a detected barline is in the right place for a barline based on the notes and bars before it having the correct duration for the current time signature. Each of these different steps should be reducing uncertainty; both for the individual objects, and consequently for the system’s total uncertainty.

Another example is accidentals being next to noteheads; the knowledge that a detected accidental and a nearby detected notehead are syntactically valid (and that the accidental makes semantic sense) is worth more than merely knowing there is one of each object in a particular place on the page, and this knowledge should be reflected somehow in terms of how “complete” the system’s internal model of the music is.

4.1.2 Detecting and Correcting Errors

A coordinated approach could help correct some errors made by the individual OMR components that require extra contextual information to resolve. Some examples of how this could work are now demonstrated.

Low-Level Errors

There are some common mistakes that are often made by the low-level processing phases when looking for and extracting musical shapes from the staff system. These mistakes include segmentation of objects due to poor image quality (for example, broken staff lines or vertical lines), objects touching due to poor image quality, and objects broken or joined to other objects caused by the system failing to completely remove the horizontal staff lines.

The higher-level phases can often indicate where missing objects should be, and can sometimes indicate or narrow down the types of objects. This information can be used to modify the unrecognised objects currently there—for example, by breaking larger unknown objects apart, in an attempt to fix objects that are joined together by faulty staff-line removal or by the image being too dark when acquired. Similarly, another modification that could be made is to join two nearby unknown objects to create a new object. This will repair an object that was split into distinct parts by a poor image acquisition process or by the staff-line removal process.

Figure 4.1 has two examples of flats between stave lines that were damaged during staff line processing/removal and split into two separate shapes. As described later in this thesis, the OMR system developed for this research is able to successfully use feedback to repair the bottom-right flat, coloured blue, and correctly identify it although the top-left flat, coloured red, is not identified. Another problem demonstrated by this example is that after an object is broken into two or more pieces, one of those parts might be incorrectly classified as another object type; here, part of a

broken flat may be recognised as a short vertical line. Syntactic knowledge is able to rule out a vertical line being in that position, causing the shape to be eventually merged with its nearby shapes as described.

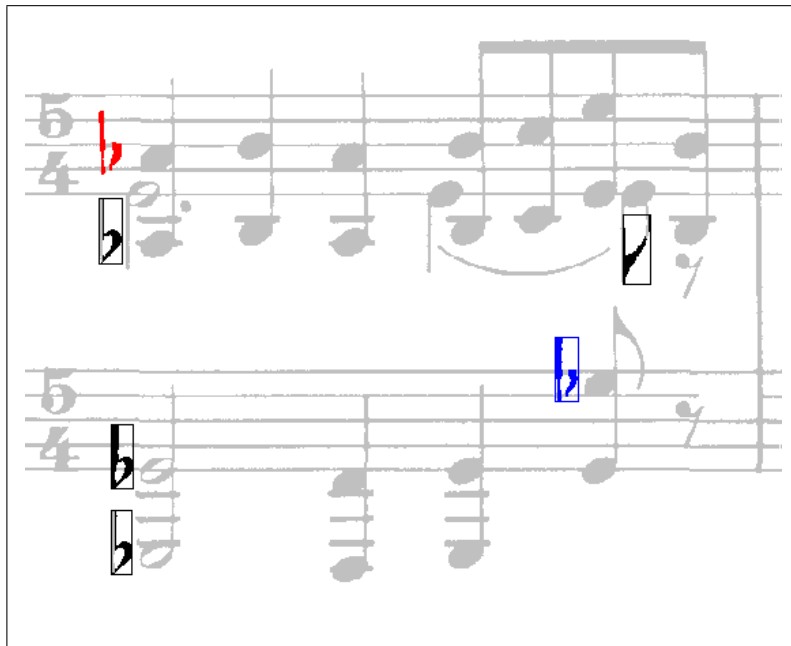


Figure 4.1: Sample Primitive Identification problems caused by low-level processing errors (identified flats are outlined)

Mis-classified and Un-classified Objects

Document recognition systems make classification mistakes when objects look similar to other object types, when objects look too different from the expected shapes, or when objects of an unknown type are encountered. The knowledge from other parts of the system should be used where possible to prevent and correct mis-classified objects.

Figure 4.2 shows the semantic state for a bar in a score where the OMR system has no low-level pattern descriptions for crotchet rests, so no musical objects were found at the start of the lower stave. The syntactic rules that create and parse the lattice expect to have notes or rests for every beat in the bar, and complain about not having any notes or rests leading the lower stave. For the node labelled 456, the syntactic parsing calculated that there are objects missing for two beats, until the first following node with a duration (node 102). Theoretically the duration of node 456 could be anywhere above zero up to and including two beats. Based on the calculated node durations for the upper stave, the suggested durations of objects missing from the lower stave are a quaver beat (to match node 99), one beat

(to match nodes 99 + 100), or two beats (to match nodes 99 + 100 + 101). The suggestions made by the syntax processing for a missing object at node 456 based on these possible durations are ‘crotchet_rest’, ‘full_notehead’, ‘hollow_notehead’ and ‘quaver_rest’.

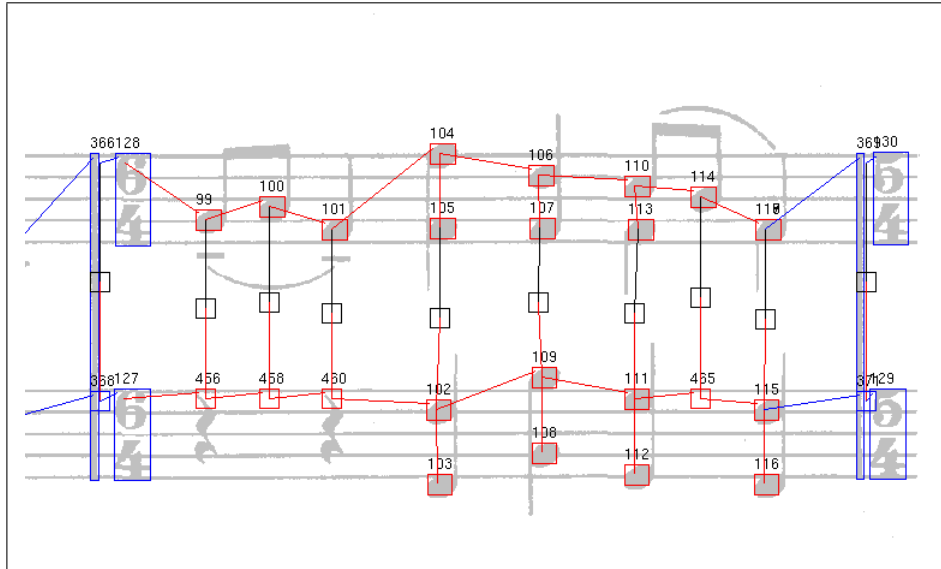


Figure 4.2: Example of unrecognised objects (crotchet rests in the lower stave)

Higher-Level Errors

Errors in the higher-level representation of the music objects are specific to the syntax and semantics of each particular notation. For example, in WMN, it is straightforward to detect bars where the duration of the notes does not match the detected time signature, but harder to determine what the cause of this is. Droetboom [28] used a set of seven algorithms to change the durational assignments of objects in bars with an incorrect duration:

- changing the duration of a rest of that is the only object in the stave;
- swap the duration for whole bar/half bar rests, since they have the same primitive shape;
- remove durational dots (and assume they were background noise instead);
- assume that nearby barlines are actually stems, and see if they can be joined to noteheads;
- assume that a barline was missed and truncate the bar;

- change the duration of notes if they mismatch in vertical alignment; and
- add rests to the end of the bar until the durations match.

Recall that the GAMERA system has a clear separation between the recognition of musical primitives and the semantic interpretation, so there is little opportunity to provide feedback for graphical re-examination of objects. These steps describe a course of action for trying to correct bad data, but testing these hypotheses by re-performing recognition on the affected objects may lead to better accuracy.

Another possibility for using semantic knowledge to detect errors is the use of machine learning techniques to examine note pitches and durations. For example, this might include detecting repeated themes or phrases with incorrect notes in them, using the note pitches to predict or check the key signatures, or comparing the note durations to the time signatures [54, 52]. Taking this idea further, the musical genre could be automatically calculated [49], based on note characteristics as well as document and staff-level metadata such as instrument names or even the composer or artist, and this knowledge applied to other rules that depend on the style of music, such as predicting the likelihood that an “odd” note or chord progression is in character. This could be done on a statistical basis, similar to how works of art such as texts or paintings can be examined to determine if they were created by a famous master, or merely by an impostor using a similar style.

4.2 Knowledge Sources

If OMR is now approached as a process that adds information about the semantics of the sheet music to some ‘internal state’—instead of a series of transformations taking inputs and output—it makes sense to think of each phase as a “knowledge source”, or a “specialist”.

As well as the tasks described in Chapter 3—Image Preprocessing and Page Layout, Text Identification, Object Identification, Semantic Parsing, and so on—there can be further examination of the internal state at different times to find more information that can be used when forming or testing hypotheses. For example, often the syntactic and semantic properties of musical objects could be used to predict semantics of nearby primitive shapes; the use of accidentals in Western Music Notation is strongly influenced by the key signature currently in effect, and accidentals with certain characteristics (such as the combination of pitch and accidental type)

are unlikely to appear.

Some examples of information that do not directly relate to the musical semantics of an object, but could be invaluable in recognising musical objects include:

- image quality—by analysing the input image, properties such as the scan resolution and scan quality (for example, brightness and noise levels) could be automatically determined, and these properties would influence the low-level graphical routines (for example, differentiating between legitimate dots and background noise).
- heuristic or problem-specific knowledge rules to pick up or work around commonly detected errors. For example, if a recognition system incorrectly identifies dots because parts of ledger lines are leftover after noteheads and stems are removed, there could be a specific syntax rule to ignore these.
- analysis of the higher-level musical semantics—for example, theme detection, genre classification [49], key and time signature calculation [55, 52] from the pitch distribution and durations of notes.

Similarly, an OMR system could have specialists that solely apply domain knowledge to fix mistakes caused by limitations in other specialists. For example, after performing object identification there may still be a large number of unknown objects, such as objects that are unrecognised due to touching other objects or broken into separate objects due to a poor quality starting image. A new specialist could be used to create new unknown objects by merging or breaking up these unrecognised objects so that the identification step gets further attempts to classify them. This is explored further in the following chapter.

4.3 Strategies for Coordinating Knowledge

Now that some of the possible knowledge sources and the types of information that they calculate have been described, strategies for representing and controlling this knowledge and the sources are investigated.

There are several important tasks to be performed by the controlling component of an OMR system. It must decide which knowledge is potentially most useful, and choose which knowledge source can benefit from a piece of information. It needs to decide when no further processing can be done (or at least, will affect the final

result), and detect or prevent loops in the system's state. Also, the method used to represent knowledge "facts" (the system's state) is dependent on the strategy chosen. The knowledge coordination strategy must also deal with conflicting information to some extent, although the knowledge specialists themselves should work out conflicts for their own music-specific areas of expertise.

An overview of methodologies used in the general field of diagram analysis is given by Blostein, Lank and Zanibbi [14]. Several frameworks that have been proposed or implemented for OMR systems are now described.

4.3.1 "Top-down" versus "Bottom-up" Approaches

A framework for a processing system such as an OMR system can be viewed as either "top-down" or "bottom-up". A "top-down" approach refers to a system that takes a holistic approach to a problem, breaking the problem into smaller and smaller sub-problems as needed until an acceptable solution is reached. For the problem of OMR, a top-down approach means that the system has a model of what is a valid score, and tries to fulfill those syntax rules, which themselves have syntax rules, and so on. Such a system would try alternate tree structures that fulfill the rules, and an evaluation function would find the best solution; this might be judged on least number of errors, or rules might be given scores so that one is used in preference over others. A simple partial demonstration might be:

A *Score* consists of *Staves*

A *Staff* consists of *Bars*

Each *Bar* consists of a **barline**, followed by *OptionalSignatures*, followed by *TimeEvents*

An *OptionalSignature* consists of an *OptionalClef*, an *OptionalKeySignature*, and an *OptionalTimeSignature*

⋮

(and so on)

Rules (in italics) are made up of other rules or terminal symbols. These rules might be applied to make sense of the set of graphical primitives that have been recognised by another part of the system. Alternatively, a more top-down strategy would be using these rules to control the graphical recognition phase. For example, if the system is looking for a note in a particular place, and it then recognises a vertical line in that region, it could look for noteheads in the region where the rules would

allow noteheads to be joined to stems.

Top-down approaches are generally computationally expensive; examples of top-down methods include Probabilistic Models and grammar-driven approaches, and OMR systems using these methods were described in Chapter 2. However, they are increasingly used in the artificial intelligence community because these models are believed to behave more like a person’s thought process and are generally better for learning or prediction tasks. For these approaches to be effective they often require training—learning from a set of examples so that the model can make better predictions when processing the input data.

Contrary to a top-down approach, a “bottom-up” approach generally focuses on the details first, and then builds them up into larger and larger components. This is sometimes called a “data-driven” approach, as the focus is on the data as it moves through various stages of processing. For music semantic processing, this would be the opposite of the example given above—noteheads and stems would be joined into notes, then notes and beams or tails joined to form notegroups, then notegroups joined together into bars, and so on. In a bottom-up system, the rules are applied to *existing* objects, rather than using rules to *predict* objects. CANTOR (described in Chapter 2) is a prime example of an OMR system that follows this model.

4.3.2 Frames

Frames are an artificial intelligence method for representing knowledge, and have been used in many domains—of relevance for OMR, frames have been used in computer vision and natural language processing tasks.

Each frame stores knowledge about a scenario or object—this knowledge is a list of attribute values, and actions that point to other frames to represent change in the state. Frames also use these links to other frames for inheritance. Rules for logical inference are then written, incorporating domain knowledge for a particular domain or application.

Stückelberg *et. al.* [73] propose using frames to represent syntactic rules and hypotheses. The frames encode information representing hypotheses on object interrelationships. Hypotheses are linked together, and these are examined to find good sets of hypotheses. Part of the musical syntax captured by the system is shown in Figure 4.3.

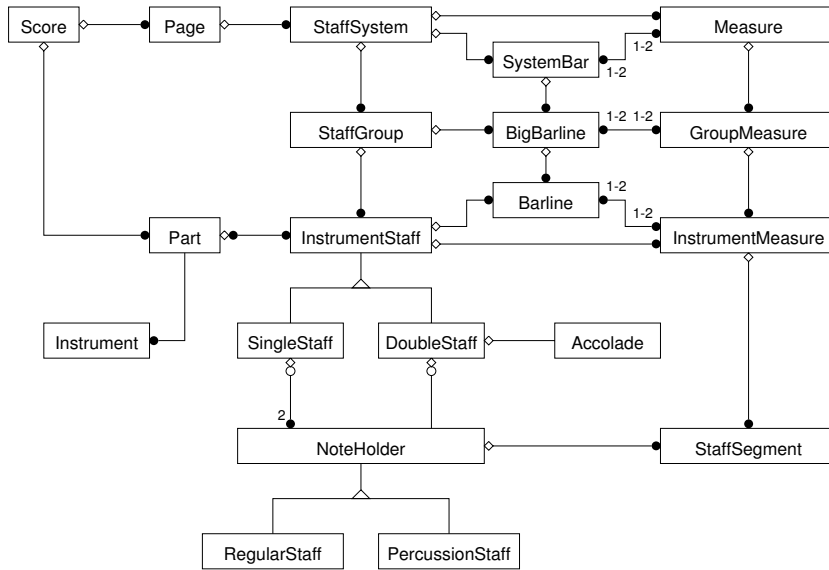


Figure 4.3: Extract from model of proposed frame-based OMR system (figure from Stückelberg *et. al.* [73])

4.3.3 Blackboard Systems

A Blackboard System is based around the idea of people in a meeting using a shared blackboard to explore and expand ideas. The system has a global memory (a blackboard) with various knowledge sources adding to and modifying the data stored on it. The blackboard has multiple ‘layers’, with higher layers storing higher-level information. Each knowledge source uses one level for its input objects, and one level for its output. A controlling process decides on the order in which the knowledge sources take their turn to process and modify the layers of the blackboard, and when to finish processing.

Blackboard-based systems have been used for image analysis [45]; their use for diagram analysis is described by Blostein *et. al.* [14] and a blackboard-based OMR system by Kato and Inokuchi is described in Chapter 2. Also in the Music Information Retrieval field, Bello and Sandler [12] describe the use of a blackboard system for converting digitised audio music into a symbolic representation, and this work was expanded on as part of the OMRAS project [63].

Figure 4.4 shows the blackboard consisting of storage areas for information about tracks—based on the frequencies of the processed audio signal input and their magnitudes, and “partials” and “notes” which are used internally by the knowledge sources for the system to eventually produce symbolic data as output. The system’s scheduler is responsible for deciding which type of information is needed and calling the appropriate knowledge source.

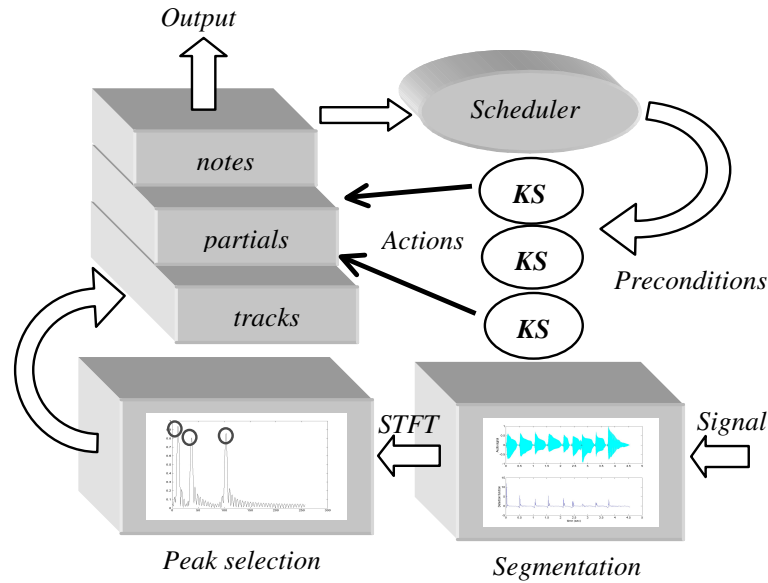


Figure 4.4: A Blackboard System for audio transcription showing the knowledge sources modifying a shared blackboard. (figure from Bello and Sandler [12])

4.3.4 Graph Rewriting

A graph consists of nodes representing data or objects, and a set of links between nodes representing inter-relationships. Graphs can be tree-like structures with a hierarchy that does not form loops between nodes, or lattices with links between any arbitrary nodes. Graph rewriting is the process of replacing a subset of a graph's nodes and links with a new set of nodes and links, based on rewrite rules. There are different mechanisms for applying the rewriting rules such as event-driven rewriting and graph grammars, as well as different notations for representing the rules. Also, the order that rules are applied in may be significant.

Graph grammars are widely used in document image analysis domains and have been used in OMR [8, 5], although the use of graph *rewriting* in OMR [32] has been much less common. Difficulties recognised by Blostein et. al. [16] in applying graph rewriting include:

- the difficulty for developers to understand rewriting expressions and to follow the transformations during the computation process; and
- the lack of standard algorithms, software tools and implementations making use of graph rewriting.

The processing cost of parsing a graph and looking for sets of nodes (and their links, which may have individual attributes) that match any of the rewriting rules

has been considered prohibitive in the past, but like many processor-intensive tasks of the past this is now much less of an issue.

4.3.5 Genetic Algorithms

Genetic Algorithms are well suited for problem classes where there is no simple algorithm for calculating answers or where the possible solution search space is prohibitively large or multi-dimensional, and finding a ‘good’ solution is acceptable rather than one single correct answer.

Genetic algorithms are an attempt to emulate nature’s survival-of-the-fittest behaviour of passing DNA from one generation to the next. The main idea is to somehow encode the system’s internal state as a sequence of attributes or decisions (analogous to DNA), and to determine a fitness function that can rank different sequences in terms of overall quality. Then, starting with a pool of random solutions (where each solution is a sequence of decisions), the fitness function chooses the best ones, and new solutions are generated by pairing the best ones at random and swapping some parts of the sequences. The process then iterates with the new sequences added to find the best solutions to pass on to the next generation, and so on. Also, some new solutions can be created by having a small probability of random mutation of some part of a solution’s sequence.

When applying a genetic algorithm to a problem, the difficulty is finding a good representation for mapping a particular domain’s internal state onto a sequence. For OMR, each individual solution might be a sequence that encodes the musical type of every graphical object extracted from the score. This could be represented as a list of numbers where each object type has a number (for example, “treble clef”=1, “crotchet rest”=2, and so on for each type) and the first number in the list refers to the first object found, the second number refers to the second graphical object found, and so forth.

The fitness function would have to calculate how well each individual sequence (representing a set of object type assignments) makes syntactic/semantic sense for the whole score.

Although genetic algorithms have been used for a wide variety of problems, including use in many graphical recognition and document analysis domains, there is little published research on their use for high-level purposes in optical music recognition. However, genetic algorithms have been used in music recognition for object

classification, for example in the GAMERA system (as described in Chapter 2) and Yoda, Yamamoto and Yamada[80]. Alander [3] gives an extensive list of publications describing the use of genetic algorithms for image processing.

4.3.6 Discussion and Summary of Coordination Strategies

The rise of “object-oriented programming” has mostly replaced the concept of a blackboard system with a globally shared area of memory. Object-orientation is a more flexible paradigm which allows more fine-grained control over how data can be shared between different objects.

Similarly, the use of frames has become less widely used as object-oriented programming has become more popular; in many ways, object-orientation follows a similar approach to storing data by allowing objects to have arbitrary attributes, inter-object links, and different functions that operate on different types of objects.

Graph rewriting is not a widely used technique, and its use depends to a large extent on data representation issues; graph rewriting makes more sense if a graph is used as the primary method for storing all information within a system. For OMR purposes it may simply be easier and less resource-intensive to recreate a syntactic graph when updated information is calculated, rather than the using the complexity of modifying the graph via graph rewriting. This has the disadvantage that any information that might be calculated about the graph structure itself is lost; keeping track of changes to the graph may be useful as higher-level knowledge (for example, finding common recognition mistakes based on similar changes made at different parts of the graph at different times).

Genetic algorithms generally require a lot of computing resources to calculate fitness values for solutions, as well as iterating through many generations until a solution of acceptable accuracy is reached. Also, determining an efficient genetic representation of the problem can be challenging. These types of algorithms are not really appropriate for controlling the OMR process, although they may be useful within knowledge sources for certain tasks that have many possible solutions that are difficult to evaluate. An example is the GAMERA system’s use of a genetic algorithm for finding a good set of attributes and attribute weightings to use in the recognition module’s decision tree.

Chapter 5

Practical Implementation

The previous chapters have described the individual tasks and sub-tasks performed by an Optical Music Recognition (OMR) system, and investigated methods for treating those tasks as distinct knowledge sources that can work together to improve the accuracy of the process as a whole. Providing more contextual information when making decisions means that a higher recognition accuracy can be achieved. A variety of approaches for coordinating knowledge sources (and knowledge) in an optical music recognition system were described.

The design decisions of a system play a factor in choosing an approach. A major goal of the work described in this thesis is to discover how different types of knowledge from different knowledge sources can be used to improve the recognition process, and which types of knowledge are more useful. For example, is feedback from the assembly stage more useful than syntactic feedback? Does the style of music represented in the score affect the usefulness of various types of information? The design of such a system should allow measurement and evaluation of its performance, rather than using a design that may give better results but is essentially a “black-box” that cannot have its internal functionality and behaviour examined.

Another important goal of this system is to be easily extensible; allowing new primitive types and syntax to be recognised should not require modification of the system’s “core” algorithms. This chapter discusses practical considerations for performing OMR tasks—and controlling the interaction between those tasks—in a coordinated fashion, and provides empirical examples of difficulties in the automatic recognition of printed music.

5.1 OMR Knowledge Coordination

An OMR system was developed for the purposes of investigating methods of representing data and knowledge, and coordinating the knowledge sources that are generating and processing the data. A primary design goal was to facilitate the development of algorithms in one section of the program without affecting the remainder of the system. This led to a modular design, with a global state that any specialist module can inspect and modify. Of course, different specialists will only be interested in relevant parts of the state—for example, low-level image processing methods are generally not interested in accessing the high-level syntactic structures. This design is similar to the blackboard systems described in Chapter 4.

The typical tasks performed by an OMR system, described in Chapter 3, are implemented as separate modules which are independent of each other, and these tasks have been split into modules as shown in Figure 5.1. Although independent, to communicate with each other these modules need to agree on common data structures for knowledge representation, the message passing interface, and the names and formatting of messages.

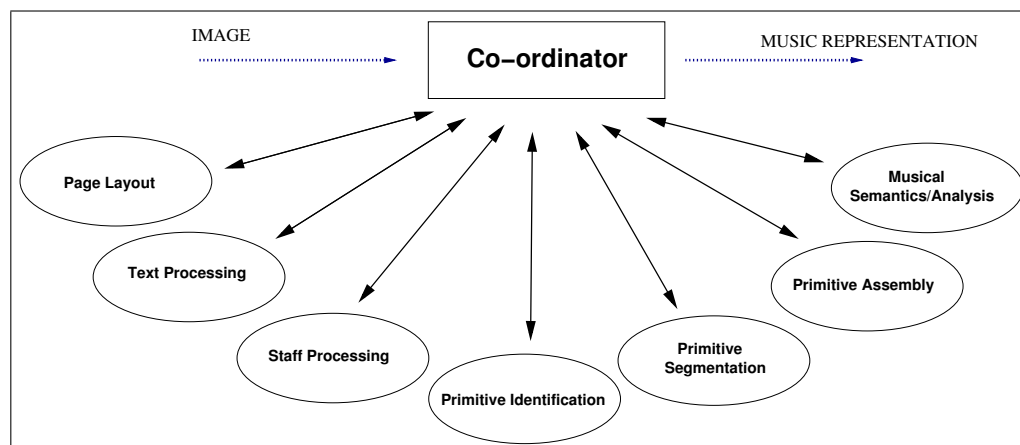


Figure 5.1: The specialist modules in the OMR System

The behaviour of the system has aspects of both bottom-up and top-down design: initially, processing occurs in a bottom-up manner as low-level features are extracted and processed and then further refined by rules to create higher-level objects, but then attributes of a top-down approach become apparent as predictive rules are used to re-check areas and objects that do not meet the expectations of those rules.

5.1.1 Request-based Model

Each specialist registers (with the coordinator) a list of request names that it can process. This means that specialists do not need to know anything about each other, as long as they use the same names for generating and receiving requests. For example, the Primitive Identification specialist registers that it can handle any requests called ‘find_primitive_of_type’, ‘identify_unknown_objects’ and ‘reclassify_objects’. Any other specialist which determines that there is a missing primitive of a particular type, or creates new unidentified objects, or decides that objects are classified incorrectly, can generate a request with the respective name—along with identifying details such as a list of relevant objects, or a relevant area of the image—and the coordinating process will give the Primitive Identification specialist the request to process.

5.1.2 The Coordinating Process

In the designed system, the coordinator is not responsible for testing hypotheses; the coordinator is responsible for the flow of execution. Execution stops when none of the specialist knowledge sources wants more information, or when there are no specialists willing or able (based on resource limits such as elapsed time) to satisfy a request for more information. An example configuration file specifying these resources limits is given in Appendix [A.1](#).

Also in this implementation, the specialists have little overlap between them in terms of functionality. The fitness of the information is judged by the specialist modules that follow it. For example, the basic geometric shapes that are detected by the Primitive Identification specialist must be joined together in certain ways only. These assembled objects can be checked for correctness to see if they follow a proper music notation.

5.2 Musical Semantics and Notation-Related Issues

The methods used to model the semantic structure of scores in Western Music Notation are now described, including some of the problems faced by this process in practice. These problems include the vagueness of the notation, innovative use of musical symbols by composers, and typesetting errors made by publishers. This is followed by a discussion on the system’s use of feedback from the musical semantics

specialist.

5.2.1 Implementation Details

The musical *syntax* of the objects found during OMR is calculated and checked by the Musical Semantics specialist. For practical reasons the bulk of the work is performed by a program written in the `Perl` programming language, which has several strengths such as object-orientation and pattern-matching that make it suitable for parsing and scanning. The musical semantics phase of the OMR process is much quicker than the low-level graphical tasks, so any potential increase in time resulting from the use of the `Perl` interpreter is negligible.

Because different music notations can have vastly different rule-sets describing how objects interact, separate rules must be encoded into the program for each music notation that the OMR system is to recognise. The remainder of this discussion is based on the rules written for Western Music Notation scores.

To understand the structure of the score, a two-dimensional graph—a *lattice*—is created from the recognised objects. As well as nodes existing for each object that has a duration—that is, noteheads and rests—nodes are created in several other circumstances:

1. where a node with a duration does not have a matching node in another stave in the same staff system, “empty” nodes are created in that timeslice;
2. nodes are created for objects that do not themselves have a duration, but affect those objects that do;
3. nodes are created for ease of navigating the lattice; and
4. barlines cause nodes to be created in each stave in the staff system, mostly for practical purposes.

These different classes are named; nodes that are for objects with a duration are called *primary* nodes. The first type listed above (called “timefill” nodes) is a special type of primary node as it has a duration but no corresponding object. The second type consists of nodes for objects such as clefs, time signatures and accidentals, and these are called *secondary* nodes. Barlines are also considered to be in this class, even though they do not directly affect the performance in the way that the other object types do. In the CANTOR system, Bainbridge [5] refers to these two classes

as *primary atomic* and *secondary atomic* node types. The third type, for navigation and processing purposes, is for nodes referred to as *divider* nodes and these are used to link staff systems and staves to each other.

The lattice is built by first joining the objects of primary node types (notes and rests) on each staff into chains, followed by the insertion of the secondary objects on each staff into the appropriate chain. Each system then vertically joins the primary nodes across the staves, inserting empty timefill nodes and divider nodes between the staves as required.

The Musical Semantics specialist also examines the set of vertical lines that have not been assembled into larger objects to see which have the characteristics of bar-lines, and re-classifies them and inserts them into the lattice. Vertical lines which are not assembled into other objects and are not re-classified as barlines are flagged for re-examination as the rules do not syntactically allow isolated vertical lines. This eventually results in requests being generated for each of these objects to be examined for a different classification. In practice, this may result in a vertical line being added back into a larger object that it was originally extracted from; this sometimes happens with flat (b) symbols that have vertical lines identified and removed from them but later have the line put back in before it is eventually recognised as a flat.

The lattice now contains all the relevant objects, and it is parsed to calculate duration information, such as the offset between consecutive timeslices, and to check that the total duration of each bar is consistent with the time signature in effect.

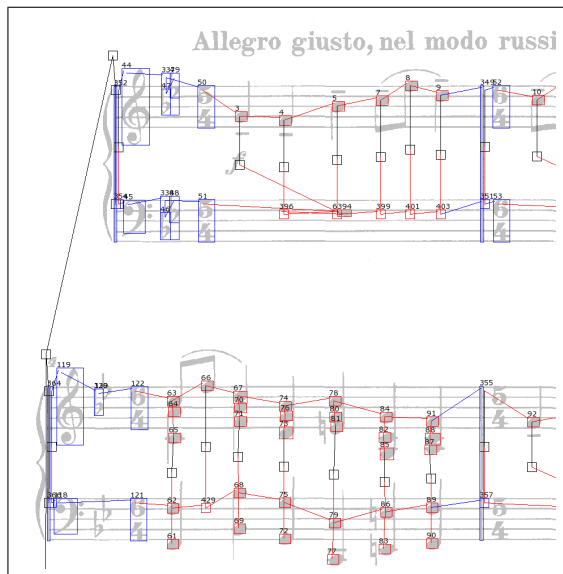
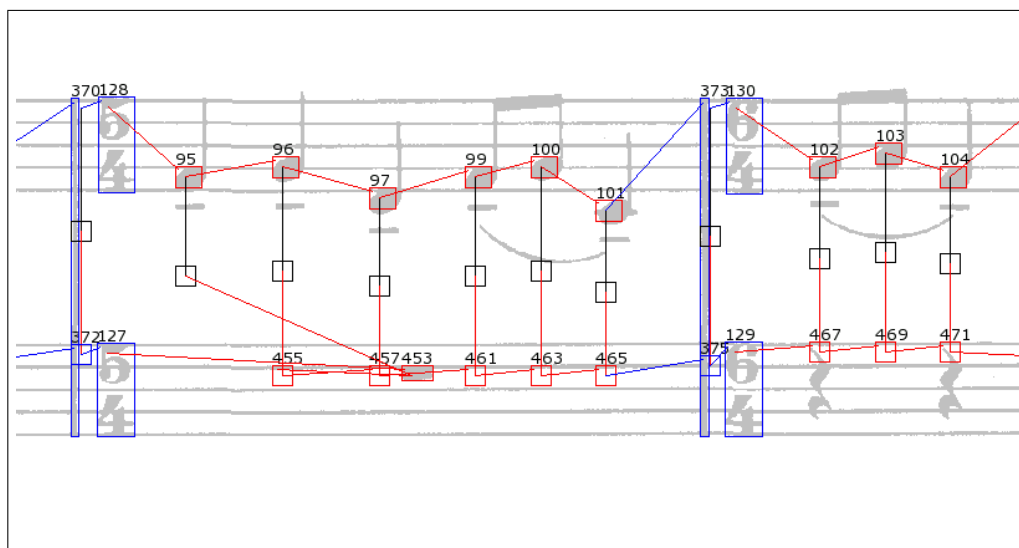


Figure 5.2: Extract showing semantic node structure

Figure 5.2 shows a graphical representation of the semantic-level understanding

by the OMR system. This shows the primary nodes for the notes, rests and their associated timefill nodes for each timeslice—linked vertically—that has no note or rest in another stave. The secondary nodes link the clefs and signatures to the primary nodes in the lattice, and divider nodes are used to vertically separate notes between the staves for each timeslice, link the beginning of each stave to its system, and to link systems together. This figure shows that the full bar rest in the lower stave of the first system semantically occurs on the first beat—at the same instance as the first note in the upper stave—despite being typeset in the middle of the bar.



(a) An Extract From a Semantic Lattice

- 1 new timesig (node 128) is the same as the old one (system 2 stave 1)
- 2 warning: staves have conflicting timesigs! (this id=127 has 5/4, other has 6/4)
- 3 moving rest (node 98) to bar start node (453)
- 4 should remove timeslice: 98 458
- 5 Bad/missing object between node 453 (calculated 4/1) and timefill node 463
- 6 possible missing object for node 463
- 7 could be crotchet_rest,full_notehead
- 8 Bad/missing object between node 453 (calculated 4/1) and timefill node 465
- 9 possible missing object for node 465
- 10 could be crotchet_rest,full_notehead
- 11 (ee) bar starting with 95: has 6/1, timesig = 5/4
- 12 bar ended by node 373
- 13 missing object for node 467
- 14 could be crotchet_rest,full_notehead,hollow_notehead,quaver_rest
- 15 missing object for node 469
- 16 could be crotchet_rest.,full_notehead.,quaver_rest
- 17 missing object for node 471
- 18 could be crotchet_rest,full_notehead

(b) Diagnostic Output relevant to the nodes in Figure 5.3(a)

Figure 5.3: Semantic diagnostics

Figure 5.3(b) shows an extract of the diagnosed problems during the system’s construction of the syntactic lattice for the score extract shown in Figure 5.3(a). These are not necessarily errors, but are caused by regions of the lattice where the calculated structure does not reflect the semantic rule set for a particular notation—in this case, Western Music Notation. These messages are now categorised and described.

Durational errors

In the time signature at node 128, the “digit_5” object was mis-recognised as a “digit_6”, and this is the cause of two diagnostic warnings: Line 1 warns that the time signature is the same as the time signature in effect for the previous bar time signature, which is probably (but not necessarily) a mistake, while line 2 warns that the two time signatures in the bar (nodes 128 and 127) on the separate staves disagree. When a time signature changes, it normally (in traditional scores, almost always) changes to an identical value in each staff in the staff system. These two observations combined present enough evidence to ignore the $\frac{6}{4}$ signature in the top staff and use the other staff’s $\frac{5}{4}$ time signature for the whole system from this point until the next time signature.

Another durational error causes the messages on lines 5–10; the semibreve rest is determined to be worth four beats rather than stretching out for the entire bar, so the system thinks there are missing objects on the remaining beats. The following assembly error results in an extra beat, which is why an object is thought to be missing from both nodes 463 and 465.

Assembly errors

The beam in the quaver group is not detected correctly, so nodes 99 and 100 are seen as two crotchets, rather than two quavers. This causes the diagnostic warning on line 11 that the bar (with six beats) does not match the time signature (five crotchet beats).

Undetected Object errors

In the figure, neither crotchet rest is recognised; node 467 is determined to be missing an object with a duration of either $\frac{1}{2}$, 1, or 2 beats, based on the durations of the detected objects in the upper staff. This is shown on lines 13 and 14 of the

diagnostic output, followed by similar warnings for nodes 469 and 471.

Other messages

Lines 3 and 4 are informational messages only, related to rearranging the nodes so that a full bar rest is moved to the front of the bar, and any timeslice created for it is removed if the timeslice now has no objects in it.

5.2.2 Unconventional Use of Notation

The Musical Semantics part of Chapter 3 discussed the problem of finding a set of formal rules to represent a notation such as Western Music Notation that is defined by common usage rather than with clear-cut rules. It is impossible to program an OMR system that can represent every possible use of a notation. Often, professional musicians would not notice anything out of the ordinary while reading some instances of unusual notation.

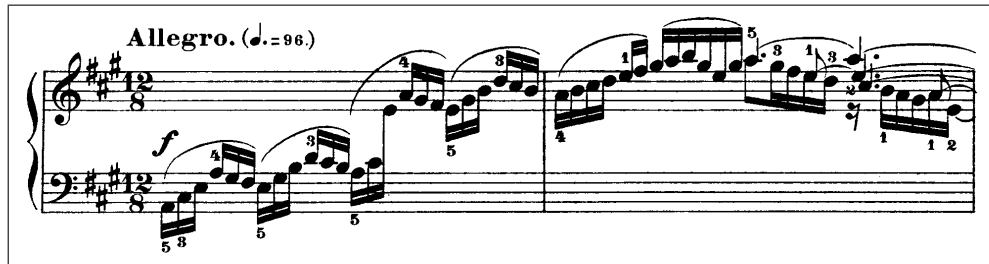
Byrd [20, 21] has detailed odd or convention-breaking use of Western Music Notation, as well as cataloging many extremities that do not break conventions but are never-the-less atypical. One example given of a common rule being broken is a time-signature change part-way through a bar that only affects one staff in the system; this is done by J. S. Bach in his “Goldberg Variations”.

Figure 5.4 shows some examples of slightly unorthodox use of Western Music Notation that may pose problems for OMR. Figure 5.4(a) shows a passage of piano music to be played using both hands and drawn over both staves. Consequently, no rests are given in the staff with no notes, even though the lower staff in the whole second bar contains no notes.

The passage shown in Figure 5.4(b) has three beats per bar. In the upper staff, the first notegroup has durational dots for two of the hollow noteheads (making their duration the full three beats of the bar), while the un-dotted third hollow notehead has only two beats, leading into the beat provided by the following crotchet note. Such a construct would typically be notated as ‘split voice’ by convention, with two separate notes—one with the stem up, and one with the stem down. The composer notated similar groups both with and without split voices through-out the score, as can be seen at the beginning of the following bar.

In Figure 5.4(c), the chords at the start of the bar occur simultaneously with the following starting note of each beamed group, despite appearing to be a completely

separate group on a separate beat. To handle this, an OMR system would need rules for this situation, using some contextual semantic knowledge such as the time signature in effect to determine that the currently calculated semantics are incorrect.



(a) A passage without rests on the unused staff.



(b) Bad voice grouping.



(c) Notes played simultaneously at four different horizontal positions.

Figure 5.4: Examples of unusual and difficult notation

5.2.3 Syntactic Feedback for Coordination

As mentioned earlier, determining the semantics of a score is difficult because there is no set of well-defined rules that can be used to determine what is or is not valid Western Music Notation syntax. The previous section discussed some recognition difficulties due to notation misuses and complexity. One practical step to take that can minimise the effect of some of these problems—as well as limitations of other parts of the OMR system—is to have a separate set of rules to specifically pick up and correct some commonly occurring errors. Baumann [8] says that: "...typical preprocessing errors can be handled by special rules."

Earlier in the section, Figure 5.3(b) showed some of the errors and warnings that the Musical Semantics specialist gave when the set of recognised objects did not fully match the rules created for understanding Western Music Notation. These observations are used to provide contextual feedback to the other specialists in the system, so that they may be able to identify a missing object or correct a misclassification and improve the system state. Four specific types of errors recorded and used for system requests are:

- incorrect bar duration;
- incorrect object duration;
- missing object; and
- incorrect object classification.

These requests also contain extra information where relevant, such as the approximate location of any missing or incorrect object, the calculated and expected attribute values (such as duration), and possible alternate classifications for existing or missing objects.

Another attribute of each request is a tolerance value that suggests how loosely thresholds and settings should be applied to tests when servicing this request. The Musical Semantics specialist increases the tolerance level on its requests, using the system's default tolerance increment step, each time it is called. This is to help resolve persistent errors that remain after several calls to the specialist, although the setting would also apply to any generated requests caused by a new internal state.

Durational Errors in Primary Nodes

For the first staff system of Mussorgsky's "Promenade" (shown in Figure 5.5), the Musical Semantics module detects five errors in the lattice created from the objects found by the earlier modules, although two of these errors are really part of the same problem.

1. The first bar has a detected time signature in the lower stave but not the upper.
2. The second bar contains 7 (detected) beats, but the time signature says it should only have 6.
3. The third bar has a (detected) time signature in the lower stave but not the upper.
4. The second-to-last beat in the lower stave of the third bar corresponds to two beats in the upper stave.
5. The third bar contains 6 beats, but the time signature says it should only have 5.

Points 4 and 5 are both really different symptoms of the same cause—the beam was not successfully detected and/or attached to the stems for the quaver notegroup in the third bar. This figure shows the detected notes and notegroups; the group in the first bar was detected correctly, but not the group in each of the second and third bars. While an incorrect bar duration is easily detected in isolation, the corrective action to take is difficult to determine. Often, further information such as that in point 4 can be taken in context to help narrow down where in a bar an error is (possibly) likely to be resolved.

The image shows a musical score for 'Promenade' by Vladimir Wassiljewitsch Stassow. The title is 'Bilder einer Ausstellung Erinnerung an Viktor Hartmann 1874'. The tempo/mood is 'Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto'. The score is in 5/4 time and features a piano (f) dynamic. The first three bars are shown, with detected notes and note groups highlighted in blue. The first bar is correctly detected, but the groups in the second and third bars are not.

Figure 5.5: Notes and note groups found by the earlier modules

Secondary Node Errors

Figure 5.6 shows the digits (and the digits assembled into time signatures) found by the previously executed specialists. The rules encapsulated in the semantic parsing code check that whenever a time signature occurs, there is one at the same time in every stave in the system. If any are missing, this is recorded and the detected time signature is used. The Musical Semantics specialist then issues a request stating that there is an object missing in that particular vicinity, of type “timesig” or either of the digits making up the found time signature. It is necessary to explicitly state the low-level primitive types because this specialist does not have a distinction between primitive objects and the assembled higher-order objects (such as “time signature”).

In this particular instance, the modules that perform the lower-level object recognition routines use this extra contextual information to look for “4”s and “5”s in the approximate area given by the Musical Semantics specialist. “timesig” was also given as a type to look for, but the Primitive Identification module does not have

Wladimir Wassiljewitsch Stassow gewidmet

Bilder einer Ausstellung
Erinnerung an Viktor Hartmann
1874

Promenade

Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto

(a) Found digits (4 in yellow, 5 in black, 6 in dark grey)

Wladimir Wassiljewitsch Stassow gewidmet

Bilder einer Ausstellung
Erinnerung an Viktor Hartmann
1874

Promenade

Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto

(b) digits paired into time signatures

Figure 5.6: Objects as first encountered by semantics module

Wladimir Wassiljewitsch Stassow gewidmet

Bilder einer Ausstellung
Erinnerung an Viktor Hartmann
1874

Promenade

Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto

The musical score for 'Promenade' is shown in two systems. The first system contains the first two measures, and the second system contains the last two measures. The time signature is 5/4. The first two measures have a 5/4 time signature, and the last two measures have a 6/4 time signature. The digits 4, 5, and 6 are highlighted in yellow, black, and dark grey respectively.

(a) Found digits (4 in yellow, 5 in black, 6 in dark grey)

Wladimir Wassiljewitsch Stassow gewidmet

Bilder einer Ausstellung
Erinnerung an Viktor Hartmann
1874

Promenade

Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto

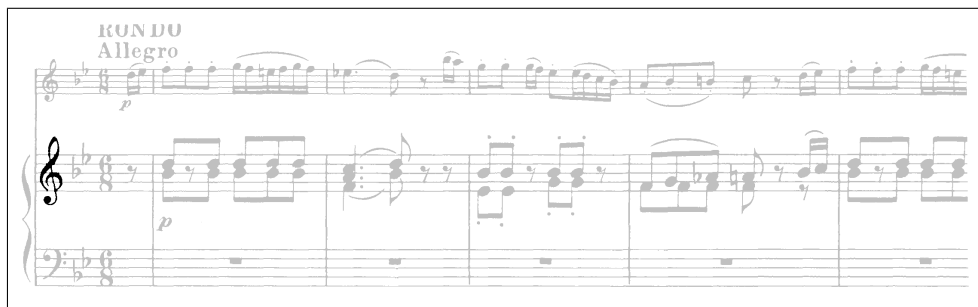
The musical score for 'Promenade' is shown in two systems. The first system contains the first two measures, and the second system contains the last two measures. The time signature is 5/4. The first two measures have a 5/4 time signature, and the last two measures have a 6/4 time signature. The digits 4, 5, and 6 are highlighted in yellow, black, and dark grey respectively.

(b) digits paired into time signatures

Figure 5.7: Objects encountered after feedback requests were processed

a rule for objects of that type. Because this context was provided, the patterns that look for “4”s and “5” were re-run but with lower thresholds for matching. For these two examples, the pattern’s template graphical data was allowed to be scaled by more than normally permitted to match the size of the object’s graphical data. Figure 5.7 shows the corrected data after the contextual feedback was successfully used.

As another example, Figure 5.8(a) shows the treble clefs found in an extract of a Mozart Clarinet Concerto. The clef on the second staff was found, but not on the first. The undetected clef is then broken up as erroneous objects (a notehead and a horizontal beam) are found within it. However, one of the specialist modules—the Segmentation specialist—joins unrecognised objects together to create new objects and joins the leftover parts of the clef back into a single object.



(a) Detected treble clefs (first time)



(b) Detected treble clefs (after use of feedback)

Figure 5.8: Treble clefs found in clarinet concerto extract

The Musical Semantics specialist has a rule that says every staff must begin with a clef, and issues a request to look for all clef types that it knows about in that general vicinity. Given the extra context that there might be a clef there, the Identification specialist relaxes the rules for all the specified clef patterns, and successfully identifies the object created from the leftover pieces.

The objects removed from the original clef are not determined to be invalid, as can be seen on inspection of Figure 5.8(b). Further development of the semantic

rules may lead to these classifications being removed and the objects added back into the clef object.

Pitch Errors

It is difficult for an OMR system to reliably detect if a note has been assigned an incorrect pitch; while a person listening to music may hear notes that sound wrong, it is hard to quantify this when working with symbolic data rather than the frequencies in audio data. It may be possible to find notes that are discordant based on the pitch frequency counts and other chord progressions in the score, but may be subjective and almost certainly composer- and/or score-specific.

The system can detect when a note has an accidental that is not required due to the key signature already in effect, but this is no guarantee that either the accidental or the key signature is incorrect. However, this information can be used to request that the accidental be double-checked.

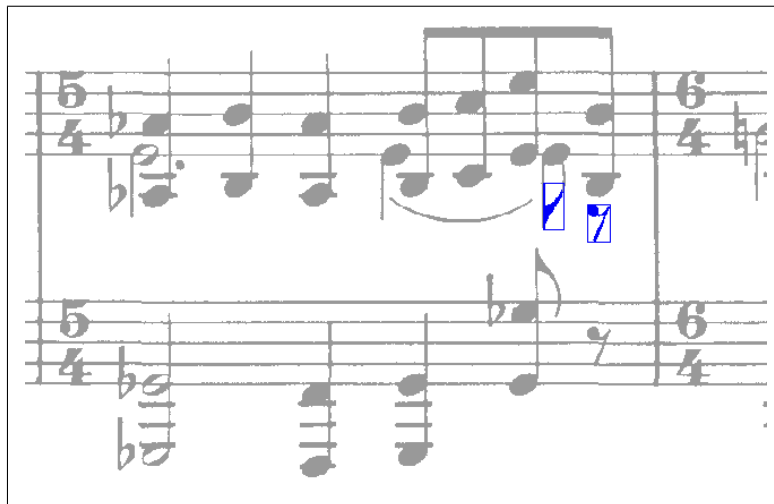
Other Errors

As mentioned above, the Musical Semantics specialist becomes more tolerant in its requests with subsequent executions. Although the aim of this is to help recognise objects that did not quite match their idealised description and remain unclassified, this can lead to problems in some circumstances. For example, in Figure 5.9(a), due to some minor issue, the quaver rest in the lower stave could not be matched. The semantic analysis detected a missing object with a duration of $\frac{1}{2}$ a beat, causing the primitive identification specialist to look for a quaver rest. Each time that this was unsuccessful, a new request would eventually be generated, resulting in a larger radius being searched, and at a lower tolerance threshold.

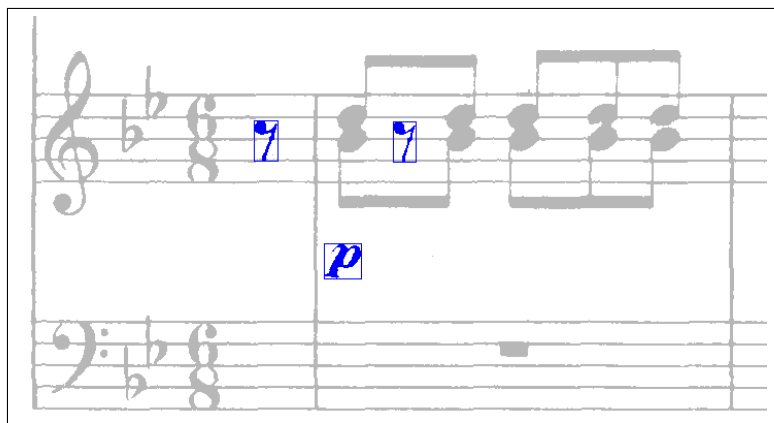
Figure 5.9(b) shows a similar situation—in this case, the manuscript publisher has accidentally omitted a rest from the lead-in bar. Each time that the semantics specialist complains that the lower-levels did not find an object of the correct duration in that general vicinity, it suggests that there might be a missing quaver rest (or object of similar duration). If each subsequent request is given greater importance and scope, the result is that the lower level specialists look in a larger and larger area, with lower and lower thresholds, until eventually the *p* dynamic marking is identified as the object that the system assumed to be missing.

In both of these cases, the recognition error was made by the Primitive Iden-

tification specialist, but was caused in part by the Musical Semantics specialist persistently suggesting a missing object with increasing tolerance levels.



(a) Promenade



(b) Mozart

Figure 5.9: Looking for quaver rests with a too low threshold

5.2.4 High-level semantics

Examining the musical meaning of the lattice created by the Musical Semantics specialist may reveal information that can be used to improve the accuracy of the recognition. For example, the distribution of the note pitches and accidentals might be useful for predicting the key signature in effect.

Detecting common *motifs* in a score, for example on the basis of relative pitch changes and/or rhythms, might be useful for detecting mistakes, although care must be taken not to be too over-zealous as variations of a repeated theme are commonly used by composers. Repeated rhythms can be found in several ways; one such method is to generate an auto-correlation of the score's notes to calculate regions of

self-similarity [69].

Pitch and duration distribution can also be used to predict—with various rates of accuracy—the musical *genre* (such as pop, rock, classical, baroque, and so on) [49, 13, 74], although genre identification is a more abstract piece of metadata, and less useful for identification purposes. However, further investigation may show that allowing genre-specific rules for identifying patterns and describing object characteristics may improve recognition quality.

5.3 Primitive Assembly

Primitive Assembly is the process of joining graphical primitives together into musical objects. Some primitive types are also musical objects without needing assembly, while other simple shapes must be built up into larger groups before becoming a music object. Of course, what constitutes a musical object is going to be notation specific. In this OMR system, Primitive Assembly is a distinct specialist knowledge source from Primitive Identification, rather than having one knowledge source responsible for identifying musical features directly from the image. The major reason for this is extensibility; having separate low-level graphical shapes and rules for combining those shapes into objects is more expressive, and should facilitate creation of rules to represent multiple music notations.

5.3.1 Implementation

Primitive Assembly is implemented in a fairly basic manner in the OMR system. In line with the goal of being easily extensible, assembly is controlled by a modifiable file with a straightforward syntax:

- objects of two different types can be joined together into a new type.
- objects of two different types can be joined, but the new object retains the dimensions and graphical shape of the first object.
- a temporary type, consisting of all objects of multiple types, can be set up.

The joining operators take a list of clauses that the objects must meet for the rule to be applied to them.

For example:

```
1 # set up a temporary type made up of a set of other types
2 digits E {digit_4, digit_5, digit_6}
3
4 timesig = digits + digit_4
5     where $1 is_above(5,15) $2
6     and $1 lhs_difference(-10,10) $2
```

A “temporary type” is one that is only used while performing assembly, and these types are forgotten when the specialist has finished while the other rules result in permanent objects being created from the assembled primitives that are used by the remainder of the OMR system. In the assembly rules given here, `digit_4`, `digit_5` and `digit_6` refer to object types generated by the Primitive Identification specialist, but temporary type rules may also refer to object types created earlier by previous assembly rules; this relies on the fact that the order that rules appear is significant. The rule on lines 4–6 create a new object type, called “timesig”. Any object created by this rule will consist of one object of type `digit_4`, `digit_5` or `digit_6`, and one object of type `digit_4`, where:

- both objects have not already been assembled,
- the first object is above the second by some specified number of pixels, and
- they are horizontally aligned, within some tolerance.

The same rules are also used in a predictive manner if one of the object types in the rule is found and nearby unknown objects would fulfill the rule as a different object type. After applying the rules to assemble the primitives, the rules are tested using unknown objects and requests are made with suitable suggestions. Using the example rules above, an unknown object that is directly above a `digit_4` not already assembled could be any of the primitive types made up by the temporary digits, and a request would be generated asking for that unknown object to be re-examined to see if it is a `digit_4`, `digit_5`, or `digit_6` object.

Complicated structures must be built up by a sequence of these joining rules, since each rule only joins objects of two types. Appendix A.3 shows the assembly rules used by the OMR system for Western Music Notation. Figure 5.10 shows a notegroup and the corresponding object that would be built-up by applying these rules.

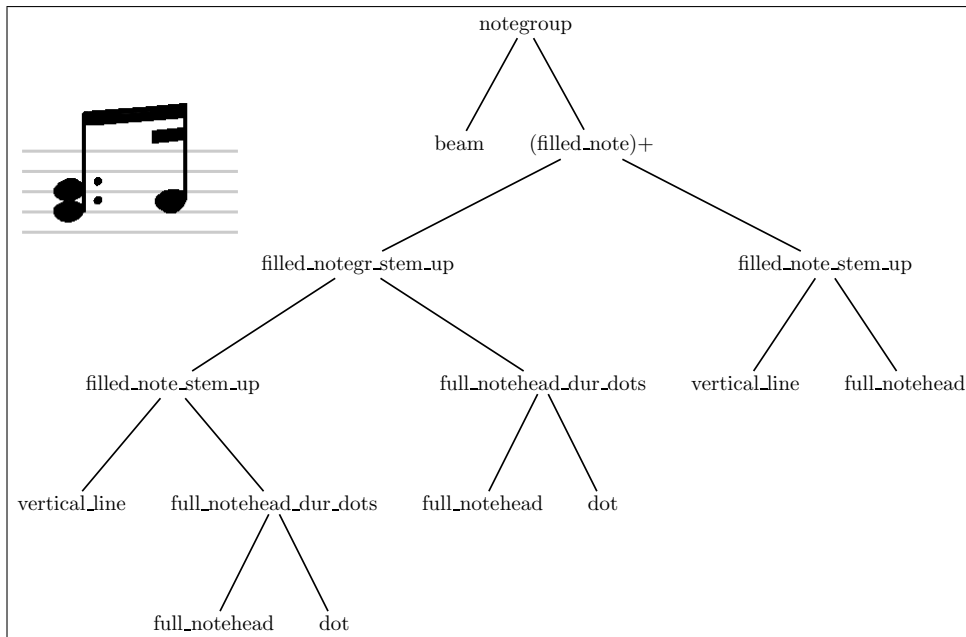


Figure 5.10: Tree-graph showing assembly

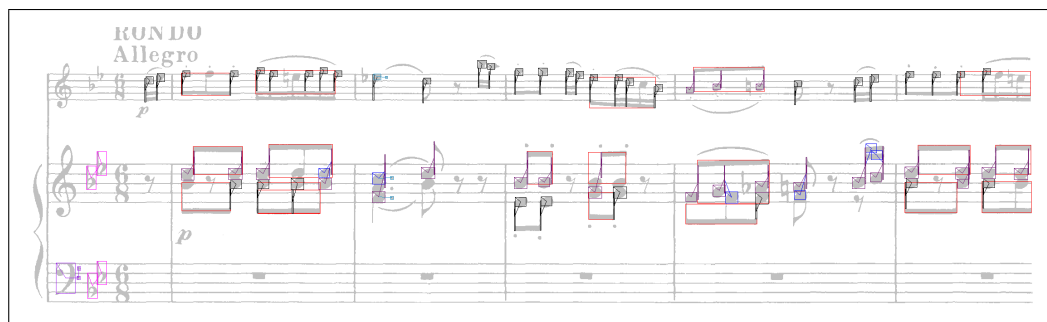


Figure 5.11: Assembly of primitives into musical objects

Figure 5.11 shows some of the results after primitive objects are assembled into larger musical objects for the first staff system of Mussorgsky’s *Promenade*. Some of the assembly rules shown in this example include:

- a “bass clef curl” is joined by two dots (in the appropriate positions) to create a “bass clef” object.
- durational dots are added to full noteheads.
- a full notehead is added to a vertical line to create a note (stem up) or a note (stem down).
- a note (either stem up or stem down) is joined with other noteheads to create a chord (stem up, or stem down, respectively).
- a beam and multiple chords (and notes) will be joined to create a notegroup

Key signature assembly is actually done by the musical semantics specialist, since that specialist performs pitch calculations, and accidentals (bs and #s) are only part of a key signature in certain places. Incidentally, this explains why the top staff's leading flats were not assembled into a key signature; a recognition error resulted in a notehead being found in the treble clef object, and the syntax rules only allow key signatures at the start of a staff, before any notes.

5.3.2 Assembly Difficulties and Examples

The assembly rules join objects together into new objects based on conditions such as their relative positions and distances. So far, these distances have been calculated heuristically, but what works well for some scores may not work very well for other scores. This is partly mitigated by using tolerance thresholds so that these distance conditions are relaxed as the system spends more time looking for objects, but this behaviour will not correct objects that were incorrectly joined together.

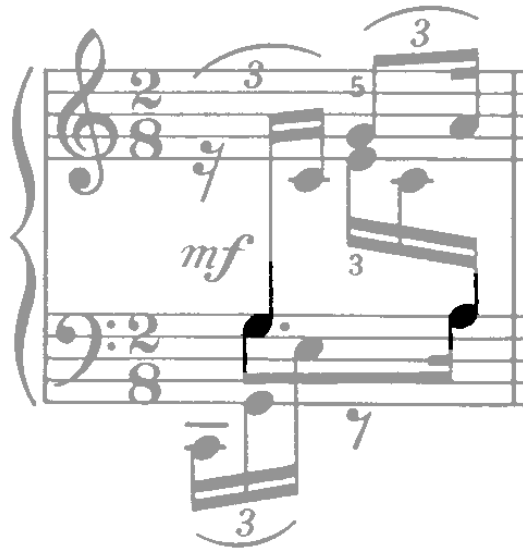


Figure 5.12: Notes shared by two parts

Figure 5.12 shows a challenging extract of double-staved piano music that has two voices on each staff: two of the notes are shared between voices, and ideally an OMR system's internal model should be able to represent this. The assembly rules used in the implemented system do not allow for this possibility; either the model should be flexible enough to allow one notehead to belong to two notes, or a work-around for the problem could be used by creating a duplicate notehead for each note. Because the system only assigns the notehead to one notegroup, the semantics will be wrong if the output is used for graphical or score-related purposes,

although any audio interpretation will be unaffected.

5.4 Primitive Identification

The Primitive Identification specialist classifies all the unknown objects on the page after the staff lines and suspected text have been removed from the image, as detailed in Chapter 3. This specialist uses several methods for identifying objects. These are now described.

5.4.1 Patterns and Template matching

The main method for identifying objects is to compare each unknown object in turn to pattern description rules. These patterns allow a set of rules to describe a primitive type both by the primitive's bounding box and with a graphical template. Other settings are used for controlling the scaling of the object (so that the pattern can be used on arbitrary-sized staff systems), and whether the pattern describes a primitive type that needs to be extracted from a larger shape or if the primitive type is normally detached from other primitives. Appendix A.2 has several examples of these pattern descriptions. There can be different patterns for the same primitive type, allowing a set of patterns to cover a variety of symbol shapes. This can help, for example, recognise symbols that differ between publisher, or have alternate forms.

When comparing primitives to the graphical template, the OMR system uses “XOR” bitmap template matching, which is a simple method for comparing each pixel in relative positions of two bitmaps. This can be thought of as superimposing one bitmap over the other, and counting how many identical pixels there are. The pattern description includes a threshold level for what proportion of pixels must match. Because this is a slow process, a common optimisation is to stop the comparison if there are too many pixel mismatches after a certain percentage (normally between 20% and 50%) of the bitmap has been tested. Wijaya [76] discusses several other optimisations that can be used with better accuracy, such as *sampling* (so that not all pixels need to be tested at first) and calculating weighted errors (where some pixel mismatches are considered worse than others).

5.4.2 Compression-based template matching

Compression-based template matching [43] compares two bitmaps by calculating how well one predicts the other. In the field of compression, the term entropy is

used to describe how much data would need to be transmitted to completely recreate an object. For an image composed of only two colours, each pixel can be encoded with 1 bit, as one of two choices. Many standard image formats do this — for example, the Portable Bitmap (PBM) image format uses a 0 to indicate a black pixel, and a 1 to indicate a white pixel. However, by taking into account the relative proportions of black pixels and white pixels, one pixel can be encoded, on average, by less than one bit.

As a rudimentary example, if an image had lots of black pixels, a ‘1’ could be used to represent three black pixels in a row, ‘00’ to represent a single white pixel, and ‘01’ used to represent a single black pixel. A single ‘1’ bit encodes 3 pixels, so on average, it takes less than one bit to represent one pixel.

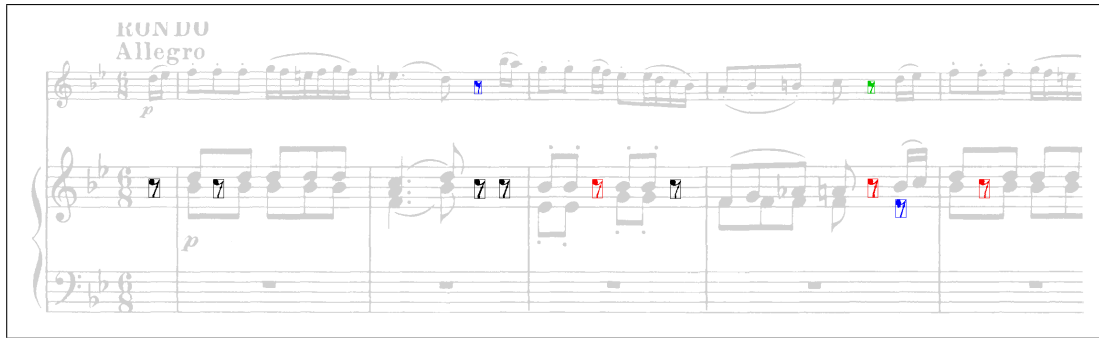
The entropy of a pixel is calculated by the equation

$$\text{Information} = -\log_2 \textit{probability}(\textit{colour})$$

For example, if a pixel is just as likely to be either black or white—50% for both—then we need $-\log_{base2} 0.50 = 1$ bit to represent either case. If say 90% of the pixels seen have been black, and we assume that the probability that the next pixel is black is also 90%, then it will take $-\log_{base2} 0.90 \approx 0.152$ bits to confirm that, or $-\log_{base2} 0.10 \approx 3.322$ bits to represent the fact that this pixel is actually white.

For the purposes of bitmap template matching, each pixel in one bitmap is predicted based on the proportion of white and black pixels seen in the ‘context’ of the surrounding pixels for the same pixel position in the other bitmap. If the two bitmaps are very similar in size and shape then these predictions are more likely to be correct, and can be encoded with relatively few bits. On the other hand, if the bitmaps are dis-similar then the predictions will be incorrect more frequently, requiring more bits to represent each incorrectly predicted pixel. The quality of the match is measured by the average number of bits per pixel needed over the entire bitmap.

Compression-based template matching as implemented in the OMR system is used to compare unknown objects against already classified primitives. This is in addition to the pattern description rules mentioned above. This acts as a form of adaptation; the set of templates used to identify unknown objects changes dynamically as the system progresses. An example of compression-based template matching being used adaptively is shown in Figure 5.13. The quaver rests shown in black were



(a) Objects matched against original 'crotchet_rest' objects using compression-based template matching. (Original objects in black)

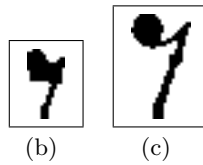


Figure 5.13: Quaver rests found by compression-based template matching. (b) shows a quaver rest from the smaller top staff (with dimensions of 14×26 pixels); (c) shows a quaver rest from the larger staff (22×39 pixels)

identified using the standard pattern descriptions. The rests shown in red were matched against one of these 'original' rests, the rests shown in blue matched one of the red rests, and the rest shown in green matched against one of the blue rests. Due to the smaller staff height of the top staff, some of its objects vary in shape and proportion to the same objects on the larger staves. These shapes are different enough that the naïve XOR bitmap matching fails to match them against the pattern template at the same threshold that the larger objects match against. The two top staff rests were found on a subsequent run of the identification specialist, presumably because they were checked against existing objects before the objects that they matched against were identified.

Adaption is a desirable feature for any recognition system, as the shapes of objects used in the current page are the best indicator of similar objects in the same document, as opposed to matching shapes against a set of pattern descriptions created *a priori*. Compression-based template matching is used for two purposes in the system: for comparing unknown bitmaps to bitmaps from classified objects, and for comparing unknown bitmaps to other unknown bitmaps. Both of these purposes are described below.

Comparing to Known Objects

As implemented in the OMR system, an unknown bitmap is classified as being the same type as a known bitmap if there is relatively little entropy required to calculate each of them from the other. Experimentally, an average of 0.4 bits per pixel or less was determined to be a good cut-off point for matches, as suggested by Inglis and Witten [43].

Comparing to Other Unknown Objects

It is advantageous to realise that several unknown objects appear to be graphically similar or identical, even if they are (as yet) unclassified. By storing this information, a knowledge specialist may be able to use this extra context to make an educated inference that could not be otherwise calculated. This is now explored further.

5.4.3 Identifying Objects with Semantic Feedback

Higher-level specialists, such as Primitive Assembly and Musical Semantics, can provide a list of possible types for unknown objects based on the known surrounding objects, using the specialist's internal rules for how objects relate to each other. This information can then be used when trying to classify objects.

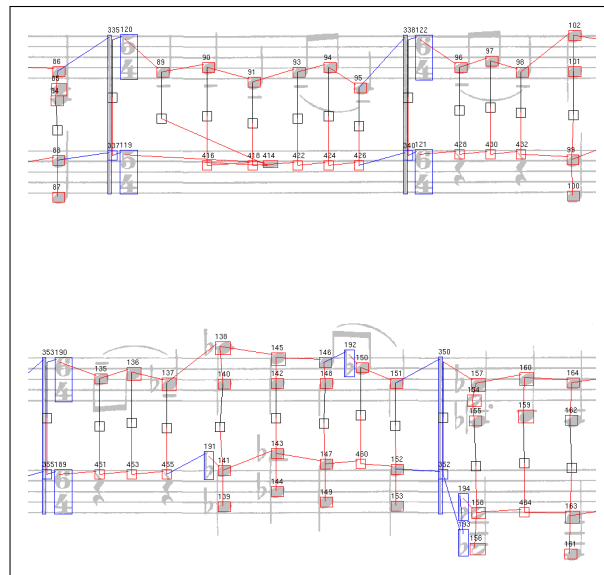
The OMR system uses compression-based template matching to group unknown objects into *clusters* of similar shapes, and then examines each cluster to see the list possible types suggested for each object by other specialists. Based on those suggestions, the clustering algorithm decides which, if any, of the suggested types will be assigned to all the objects in that cluster. Factors that influence whether or not to assign the type include:

- the number of unknown objects in the cluster,
- the proportion of objects in the cluster which have that type as a suggestion,
- the number of (and certainty assigned to) different suggested types, and
- whether there is a rule describing the suggested type, or if it is a previously unknown type that the Primitive Identification specialist has no knowledge of.

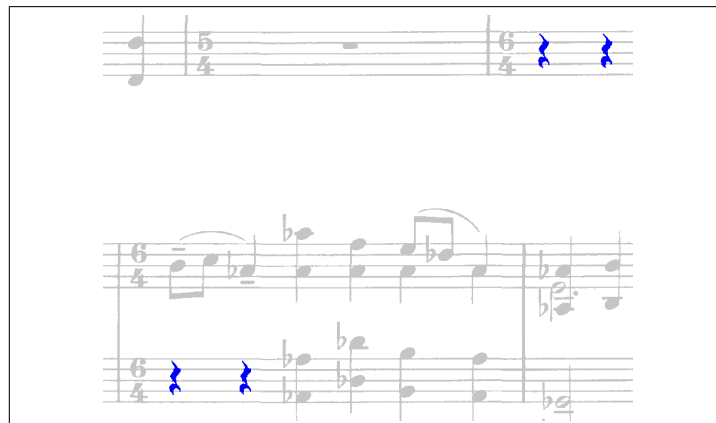
The idea behind the fourth point is to bias decisions in favour of suggested types that are not currently recognised. If there are recognised objects assigned with that type, then the assumption is that the Primitive Identification specialist should be

able to match other unknowns of that type, for example by using compression-based template matching to compare them to the existing classified objects. Perhaps more importantly, this also means that objects can be classified as types that exist in the musical semantic rules but do not have primitive description patterns written for them.

For example, if there are six objects in a cluster of similarly shaped unknown primitives, and they all have the same suggested classification (and no other offered alternate types), then it makes sense to accept the suggested type for those objects. If there are several competing suggested types, or types that are only suggested for a minority of the objects in the cluster, then the above factors must be weighed up to decide which (if any) suggested type should be given to the objects.



(a) Missed crotchet rests shown in semantic lattice (near nodes 428, 432, 451 and 455)



(b) Classified crotchet rests after compression-based template matching clustering

Figure 5.14: Clustered objects identified by use of semantic information

Figure 5.14 gives a practical example of this. The Musical Semantics specialist detects that there are objects missing from its lattice, shown in Figure 5.14(a), and calculates a list of possible objects based on the durations in the relevant time segments. (This was described in more detail in Section 5.2.3.) When the Primitive Identification specialist receives a request to identify unknown objects, the possible types for each object, based on the semantic feedback, is taken into account. This specialist detects that there are four unknown objects that are very similar in size and shape, and that all have “crotchet_rest” as a suggested possible classification. It then makes a decision to classify them as this type, as shown in Figure 5.14(b).

It is important to note that, in the example given here, the Primitive Identification specialist has no knowledge of the ‘crotchet_rest’ primitive type. There is no pattern description rule describing the graphical appearance; the knowledge that there is a musical object type called a ‘crotchet_rest’ that has a base duration of one beat is represented by the syntactic rules that describe Western Music Notation. But once this information was attached to objects, even as an uncertain classification, another specialist was able to use it to make a decision with the benefit of this extra context.

5.4.4 Segmentation of Objects

An OMR system needs to correctly account for objects that are either

- deformed because they overlap or touch other objects; or
- broken up into separate, non-touching pieces.

Both of these problems can be caused by poor typesetting or be introduced during image re-production (such as photocopying or scanning). Computer-typeset scores are generally much worse than hand-crafted manuscripts for typesetting problems, since experienced engravers try very hard to ensure pages are readable. To recognise touching and overlapping objects, there are two approaches that could be used:

1. use patterns and algorithms that are designed to match against only part of an object. (In this implementation, notehead recognition uses this technique to extract shapes from larger objects, rather than only comparing against isolated objects.)
2. perform pre- and post-processing on unrecognised objects to determine where

they may be overlapped; thinning algorithms can find ‘weak’ points to break up large objects.

The first method has the advantage of being more robust but is generally much slower, as it requires every pattern to be matched against many possible offsets inside the larger object. The second method has the disadvantage of not having any context for deciding which weak points are due to image or typesetting quality and which are inherent in the graphical shape.



Figure 5.15: A barline partially removed by the typesetter.

Sometimes segmentation can be deliberate—Figure 5.15 shows an example where a barline has been partially removed by the typesetter to make room for a performance instruction. While this does not pose a problem for a musician reading a score, segmentation such as this may confuse an OMR system. In this particular case, two vertical lines may be found instead of one, but they may not be recognised as barlines if barlines are classified as vertical lines that begin and end on the appropriate staff lines.

Segmentation of objects can be due to poor image quality, but is sometimes also caused by some of the image processing steps during OMR. The OMR system developed for this research has a Segmentation specialist tasked with modifying unrecognised objects so that the Identification specialist has a better chance of recognising objects that were previously deformed. The main task of this specialist is to join two nearby unknown objects into a new larger object, and then generate a request asking for any newly created objects to be processed by the Identification specialist. From observation, segmentation of objects due to staff line removal is almost always vertical and not horizontal. This knowledge could be used to improve the process of joining objects together again, by using different thresholds for

distances in each direction.

5.4.5 Practical Issues in Object Identification

If a specialist suggests that an object is classified incorrectly then the object may be reset back to being classified as “unknown”. For a Western Music Notation example, vertical lines that do not make syntactic sense—that is, they are neither a barline or part of a note—are returned to an “unknown” classification, due to a request by the Musical Semantics specialist indicating that the current classification of those particular objects is incorrect. Sometimes vertical lines are extracted from objects that are not initially identified, such as from the stem of flats (b).

When an object was mis-identified and extracted from a larger object (as some patterns are, notably full- and hollow-noteheads), then it should be inserted back into the larger object. This can cause difficulty in putting objects back together again if some of the leftover fragmented components were themselves joined together into new, larger unknown objects. The system amalgamates previously fragmented objects when one fragment is re-classified to “unknown”: this is only done if the parent object and sibling objects (that is, the leftovers after the object was removed from the parent) are flagged as “unknown”, even if the siblings have since been joined into larger objects by the segmentation specialist. This is only possible if the larger object that a sibling is now part of has not itself been classified.

Noise versus Dots

Many objects are found that meet the pattern description rules’ criteria to be classified as a “dot”. Dots are used for different purposes in Western Music Notation, so it may require semantic information to determine the meaning (if any) of a dot. For example, dots are used as part of the bass clef object, they can follow a notehead as a durational (augmentation) dot, and dots can also be used above or below noteheads as a *staccato* marking.

Many dots are also found in the leftover parts of the image after the extraction process for other primitives. Figure 5.16 shows several areas where the extraction of identified objects such as noteheads and vertical lines has left behind small fragments of images, and these may be mistaken for small dots. Although these fragments appear different to the legitimate dots in the figure, there are often a variety of shapes and sizes of legitimate dots that look similar to some of these noise fragments.

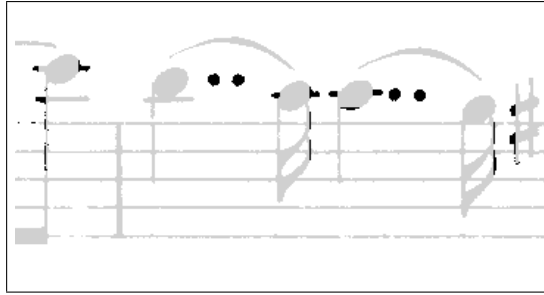


Figure 5.16: Extract showing dots and noise remaining after object extraction

A simple technique to overcome this problem is to modify the primitive identification extraction process so that small objects left over that would meet the criteria to be dots are instead marked as noise, so that only completely disjoint dots are further processed to be identified by the dot pattern rules.

Droettboom [28] deals with the problem of extraneous dots affecting note duration when examining bars that have a duration inconsistent with the time signature. One of the methods used for correcting the bar is to remove a durational dot from a note that has one, and test if that corrects the bar’s duration.

5.5 Text Recognition and OCR

The Page Layout specialist uses a flood-fill [62]—sometimes called “connected components” analysis—to group the black pixels on the page into objects, and marks large objects as possible staff systems. The Text Processing specialist looks at the remaining objects marked as “unknown”—initially, all objects that are not large enough to potentially be staff systems as determined by the Page Layout specialist—and uses some basic heuristics such as minimum size and minimum aspect ratio to determine which objects are likely to be textual. In addition, suspected text characters are assumed to be more likely to be text if they are horizontally close to other suspected text characters. Fujinaga [36] makes a distinction between suspected text objects that are near others and those that are not. This increases the chance that characters that are part of a text region but unrecognised are still flagged as “unknown text”, rather than reverted to being “unknown” objects and then being incorrectly identified by the musical object primitive recognition routines. The reason for this is that some objects that are not on a staff system might appear to be text, but are not. Examples of text that might not be near other text includes bar numbers, fingering directions, and dynamic markings, while examples of objects

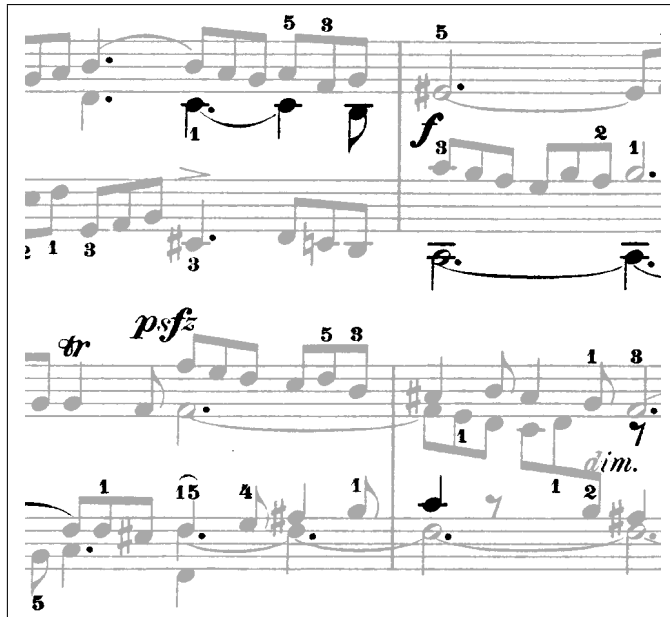


Figure 5.17: Sample score showing both text and musical objects off the staff

that might be mistaken for text (based on the loose filtering by position and size) include short phrase markings such as ties and pauses, accents, ledger lines, and notes that are not physically joined to the staff system (for example, due to broken stems, or split voice with stems pointing away from the staff). Figure 5.17 shows this problem for objects located off the staff system, with both text objects (digits, dynamic markings such as *f*, *psfz* and “*dim.*”, and the “trill” mark) and musical objects (including dots, notes, rests, ledger lines, and tie/slur markings).

Performing OCR on the suspected text objects should more accurately identify which are text and which happen to be non-text objects that are outside the staff system areas, leaving these to be processed by the music primitive recognition stages. This may still not be enough if some musical objects in a particular notation look like text. For example, in Western Music Notation, the flat symbol \flat can look similar to a lowercase “b”. In other notations, text may actually be used for notational purposes—recall the Indian Bhatkhande notation example in Figure 1.3 (page 3). It may be possible to use semantics to find falsely classified objects; an object identified as a text character that seems to be out of place could be given to the musical Primitive Identification specialist to test whether it also looks like a musical object. However, this line of investigation was not pursued.

It is also expected that performing text-removal will result in a net performance gain for the whole OMR process, as there are fewer unknown objects for the later stages to do multiple passes over for making suggestions.

The lack of a high performing open source OCR implementation prevented further efforts into exploring the use of textual semantics in this OMR system, although this situation may improve in the future. While preliminary tests show that the primitive recognition routines could be adapted for recognising text glyphs, a text-specific process, taking advantage of structure and properties unique to text, should give better recognition results. Examples of text properties for Western alphabets include the baseline, the *x-height*, the *caps-height*, and the descender-depth for each font, as well as language detection. For Latin-based alphabets, the language can often be inferred from the character frequencies and combinations, and this can be used for spell-checking and limiting the range of valid characters (such as accented characters).

5.5.1 Musical Semantics of Text

Any text on a score often affects the semantics of the music. This means that not correctly identifying text or understanding the text’s meaning affects the music. Several examples are:

- Identifying which digits are for grouping and triplets, and which are performance suggestions such as fingering numbers. The first bar of Figure 5.18 has both triplets (marked with a “3” enclosed by a slur) and fingering suggestions (marked with a “3” and a “5”).
- Implied phrasing, where the musician understands that an instruction continues to take effect. This is often explicitly marked by “*simile*” or “...”. In Figure 5.18, the notes in the first bar have triplet markings, and a musician understands that the remainder of the piece is performed with triplets despite the lack of explicitly drawn markings or instructions to this effect. This means that each group of three notes is performed in the space of two notes. Because tuplet markings affect the duration of the notes, not identifying triplets leads to bar durations not agreeing with the time signature.
- Instrument names for staves in orchestral or group music scores. This can be important for aligning staves between sheets when some staves are inserted or removed from pages. Typically, the first page of a conductor’s score lists all the instruments, and staves are dropped for instruments that do not play a passage or movement in the subsequent pages.

- Lyrics need to be correctly associated with the appropriate note for timing and duration information.

5.5.2 Example of Text Detection

Figure 5.18 shows the result of the text region detection on a piece by Chopin. The suspected staff systems are greyed out, and the objects in black are suspected text objects. Most of the non-system objects pass the above heuristics for classification as possible text; the objects (coloured medium grey) that are excluded by these rules are either too small—such as some dashes and dots—or have an unacceptable aspect ratio, such as some slurs and system braces.

This demonstrates some obvious problems with the simplistic text detection. Firstly, some musical objects that are not touching part of a system are flagged as potential text because they have the similar size or shape as real text. Rests and notes in split voice are particularly likely to be detached from the system. Secondly, some small objects that are excluded may be needed for OCR; text includes punctuation and accented and dotted letters. Fujinaga [36] includes small objects as text if they are very close to other detected text. Thirdly, it is not uncommon for consecutive characters to be touching, resulting in a wider bounding box. This may mean that they form one large object that has an aspect ratio or size that excludes it from being flagged as possible text. Lastly, text on different parts of the page may be in different sized fonts, so any size thresholds may be too limiting. Carter used a size relative to the inter staff-line gap for lyrics in sacred harp notation [23]. As mentioned earlier in this section, the “x-height” of a line of text is easy to calculate, so this could be used to identify lines of text at once rather than individual characters.

5.6 Staff Processing/Page Layout

To find staff lines, the system performs a horizontal projection (as described in Section 3.3) and then calculates which peaks in the projection correspond to staff lines. The algorithm used for removing staff lines while leaving the super-imposed objects intact is based on Bainbridge’s “track_wobble” method from the CANTOR system [5]. To remain flexible so that different music notations may be recognised, the system does not assume that staves may only have five staff lines (as in Western Music Notation). Instead, the number of staff lines is calculated based on the gaps

24 PRÉLUDES

OPUS 28

A son ami Camille Pleyel · 1839

Agitato

I.

vif

cresc.

stretto

7

16

23

28

Figure 5.18: Objects suspected of being text (in black). Excluded objects in grey.



Figure 5.19: Obvious peaks in staff system projection

between them; small gaps are assumed to be between lines on the same staff, while large gaps are assumed to be gaps between staves on the same staff system.

The OMR system identifies the staff lines by first finding the longest peak in the projection, and then finding all the peaks that are longer than some threshold relative to the longest peak. If this threshold is too high, then some staff lines will not be found, especially if the image is very slightly skewed. If the threshold is too low, then erroneous staff lines will be found where peaks are caused by other objects. Also, the length of the longest peak in the projection is used as a measure of the skew; if the longest peak is less than 65% of the staff system's width, then it is assumed that the system is skewed. If $\frac{1}{5}$ or more of the staff systems are determined to be skewed, then the staff processing specialist generates a request for the page to be rotated to remove skew.

Figure 5.19 shows a projection of a completely de-skewed staff system. Because there are many long beams at the same vertical position, there are several long peaks in the projection for the corresponding horizontal line. These may be mistaken for stave lines. Figure 5.20 shows an extract with a deformed staff system; it appears that the typesetter made an error in aligning the lower stave of the top staff system. Also, the image was scanned in with a small amount of skew, and the projection shows both of these problems. The short peaks of the projection compared to the system width are symptomatic of skew in the image, while the even shorter peaks in the lower stave of the first system are caused by the staff lines being non-parallel to the staff lines of the upper stave and lower staff system.

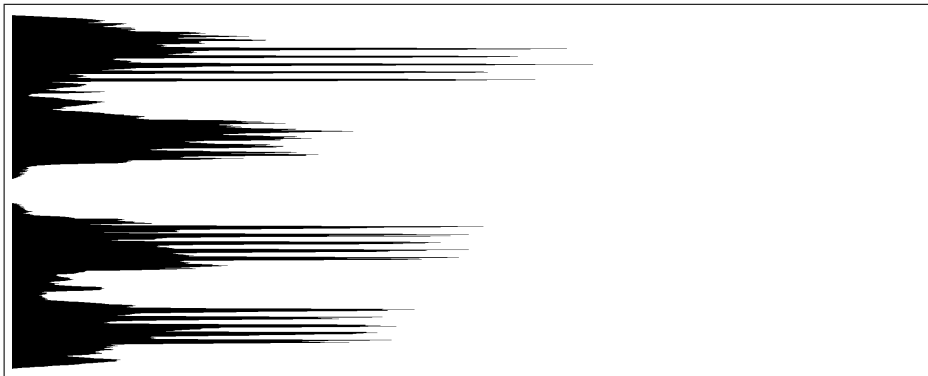
Like the other parts of the OMR system, the Staff Processing specialist tries to be more tolerant with its threshold settings as the system execution advances. To achieve this, the specialist gradually reduces the percentage of the longest peak required for other peaks to be identified as staff lines. On the most tolerant setting, peaks that are only 20% of the width of the longest peak are assumed to be staff lines, but this looseness is only tried after multiple attempts to identify suitable staff systems have failed.

5.6.1 Page Layout

To correct the image for any skew due to the image not being scanned in completely straight, a Hough Transform is used to find long lines. Each pixel in the input image can lie on many possible lines, and each potential line—at every angle between 0



(a) Non-parallel staff lines in the upper system



(b) Horizontal projection of staff systems

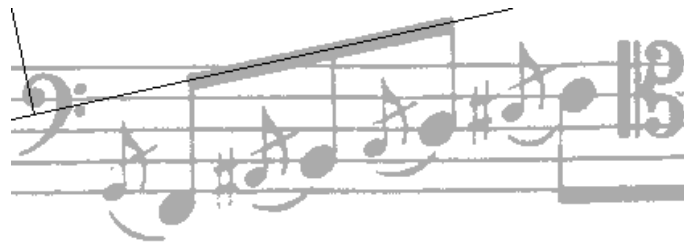
The staves are not parallel; the inter-stave gap is marked in the first staff system.

Figure 5.20: Obvious deformation in staff system affecting projection

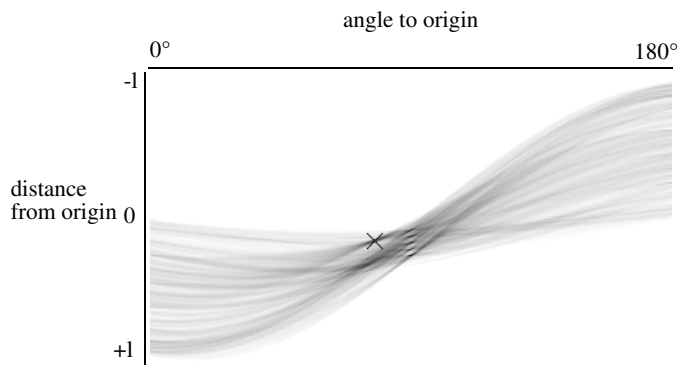
degrees and 180 degrees—through that point is recorded. If many pixels are on the same potential line then the transformed point for that line will have more votes, shown as a dark spot in Figure 5.21(b). The pixels that lie on the marked line in Figure 5.21(a) are represented by one position in the transformed data, marked with a cross in Figure 5.21(b). This marked line is identified by its distance from the origin, and the angle of the perpendicular line. There are a series of dark spots at 90 degrees in the transformed data, corresponding to the long horizontal staff lines in the image. It is this area of the transformed data that has the strongest points and these points indicate the skewed angle, if skew is present.

The Hough Transform is a robust method for calculating the angle of skew for the input score image. This can be processor-intensive, so several steps are taken:

1. The transform is only done on the largest single object found on the score, rather than on the whole score. The assumption here is that a staff system will be the largest found.
2. The algorithm was used in two stages—firstly, using a coarse resolution to find



(a) Sample Image for Hough Transform



(b) Graphical Representation of Hough Transform Data

Figure 5.21: Using the Hough Transform for calculating skew

the general skew angle, and then at a finer resolution of only twenty degrees.

5.7 OMR System and Coordination Issues

So far, this chapter has discussed implementation details and issues for the various specialists in the OMR system. However, there are some observations that are either not about any particular specialist, or affect all the specialists. These points—as well as issues that relate to the coordination of these specialists and their data as a whole—are now examined.

5.7.1 Boundaries between System Stages

Some processing or generation of information could arguably be the responsibility of more than one knowledge source. For example, in Western Music Notation, there are several cases where some objects are specialisations of other objects; a commonly occurring case is that some vertical lines are barlines. Which specialist should be responsible for determining which are barlines and making this assignment—is this semantic, syntactic, or primitive information? Similarly, alto clefs and tenor clefs have exactly the same appearance, as shown in Figure 5.22; the only difference is where they are located on the staff.

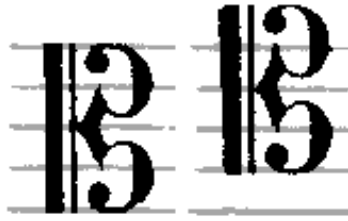


Figure 5.22: Alto (left) and Tenor (right) Clefs

Technically, it does not really matter to the system which specialist assigns a particular value to an object; an OMR system using a coordinated approach should be flexible enough that where individual pieces of information comes from does not affect the overall system recognition quality.

Although the specialists in the implemented system were designed to have minimal overlap in functionality, only minor modifications would be needed to support this. One possibility is to have multiple specialists able to handle the same types of requests, and have a strategy for choosing which specialist should handle each instance one a request-by-request basis. This is now further explored.

5.7.2 Possible Coordination Strategies

This system could be extended to allow competing knowledge sources (such as different pattern recognition modules), although this would require new coordination strategies. Possible methods for dealing with the different conclusions from each competing specialist include:

- the use of a voting mechanism between the different specialists. For example, each Primitive Identification specialist could be asked: “Do you think this object is a treble clef?”. Alternatively, the specialists could rank a list of likely classifications and the coordinating process combines these lists and weighs up each entry (possibly based on each specialist’s history of accuracy) to determine the most likely classification;
- trying each alternate hypothesis from each competing specialist, and testing which leads to the most accurate representation of music at the end (for example, based on the number of detected errors and unrecognised objects); or
- having a preferred order for competing specialists to be used, so an alternate one is only used if the preferred specialist made a detectable mistake or was unable to process a particular object.

- choosing the least-recently used specialist that provides the functionality, so that each is chosen in turn for subsequent requests of the same type.

5.7.3 System Progress and Tolerance Levels

The coordination module is responsible for passing the execution to each specialist as required to service requests. One potential problem with the current design of this system is the possibility of the system performing a loop, where a specialist generates a request which is serviced but does not change anything, so that the system's internal state does not change and the specialist generates an identical request when it is next called, and so on. There are several techniques used to minimise this effect. Firstly, the coordinator has resource limits on each specialist, and these can be configured on a per-specialist basis by the operator. For example, by default each specialist is only allowed to receive five requests while processing the score, and only five requests made by each specialist will be serviced by the coordinator. In addition, by default each specialist will not be given any requests for servicing if it has already used up sixty seconds of processor time. However, typically the Primitive Identification specialist is configured to allow many more requests—experimentally, two hundred was found to be adequate—since it is the only specialist that can service the majority of the requests. The second technique used to help prevent the system getting into a loop is for specialists to detect if no progress is being made, and to increase their tolerance level so that they are looser when applying thresholds.

Specialists can detect whether or not progress is being made by examining the objects that are within their area of expertise. For example, the Primitive Assembly specialist calculates its state based on the possible types it has calculated for unknown objects, and it keeps track of its state each time it is called. Similarly, the Musical Semantics specialist calculates its state from the warnings and errors it generated when parsing the semantic lattice. When one of these specialists detects that it is in an identical state to one that it has previously been in, it relaxes the tolerance levels on each request it generates.

When identical requests are repeatedly made, something needs to be done differently to change the system's state. Some specialists perform different actions depending on how often they are called. For example, the Staff Processing specialist becomes more tolerant each time it is asked to find staff systems, because it only

performs this task if there are no suitable systems found. Alternatively, a request that has been generated many times should have a lower chance of being selected, since little progress is being made, and processing should be given to other requests that may make changes to the current state. The implemented system only counts identical states rather than identifying previously generated requests.

The general behaviour of this system is to start with restrictive threshold values, and become more tolerant over time as more information about objects is calculated. When suggested classifications for an unknown object are tested, the thresholds are loosened since those classifications did not adequately describe the object's shape at the current tolerance levels. When an object is detected as being incorrectly classified, it is re-examined by the Primitive Identification specialist with that classification blacklisted from consideration. A possible alternate strategy that could be investigated is to make the testing use more restrictive thresholds, rather than excluding types from being tested.

5.8 Other Practical Difficulties

Various difficulties with some scores were encountered during the development and testing of the system. These include problems due to accidental omissions or mistakes by the composer, carelessness by the typesetter, and complex or uncommon notational constructs. Several of these are now described, illustrated with example input score fragments to demonstrate the problems.

5.8.1 Typographical and Editorial Errors

It is not uncommon for published scores to have minor typographical errors in them—often a seasoned performer or music reader will not even notice a minor omission. Figure 5.23 shows one of these simple omissions that was encountered earlier in this chapter: the lowest stave is missing a quaver rest in the lead-in bar. Ideally, an OMR system would be able to detect situations like this; in this case, a missing rest will not affect the audio semantics as rests are generally added where there is no obvious alternative. However, this situation could cause an error if the system is too broad in its search and decides that another object is the missing object, as demonstrated in Figure 5.9(b).



Figure 5.23: A typographical error (missing rest)

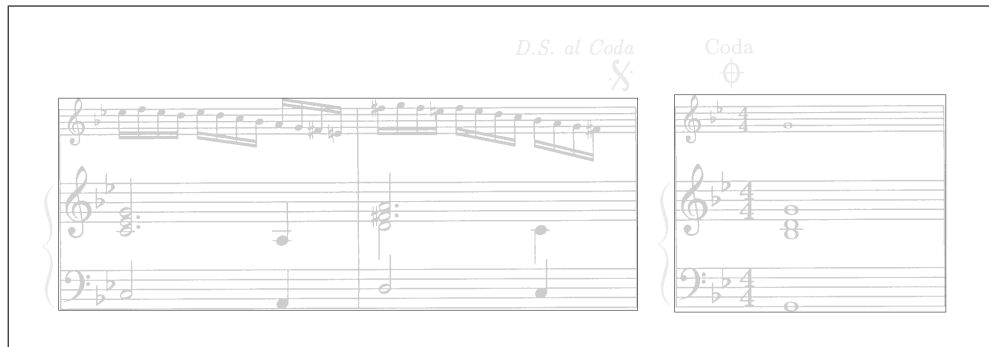


Figure 5.24: Two staff systems side-by-side. The added outline shows that the second is slightly higher than the first.

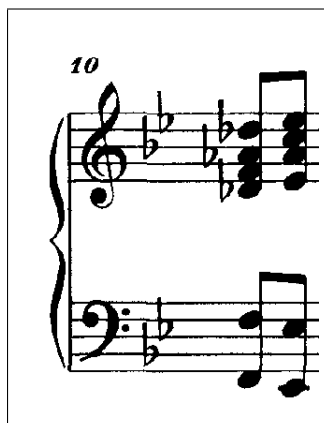
5.8.2 Staff System Locations

A page of music may have staff systems horizontally side-by-side. Obviously, it is important that systems are processed in the correct order when determining the musical syntax. Figure 5.24 shows an example of two systems next to each other, where the second system is marginally closer to the top of the page than the first. A naïve algorithm that processes systems in order from the top of the page will get this wrong. Both the vertical and horizontal positions of systems need to be examined to calculate the correct order.

5.8.3 Complex or Poor Layouts

Key signatures are composed of multiple accidentals, although all the accidentals are of the same type. The pitch of these accidentals must be in a particular order: Figure 5.25(a) shows a bar that has a key signature with two flats ($B\flat$ major),

followed by a chord that has accidentals. The upper two accidentals in this chord happen to be at the pitches that would be in the correct order for a key signature with four flats (A^b major), and with relaxed tolerances in effect, the lack of any object between the key signature and these flats can result in the flats being assigned to the key signature rather than to the notes. In this particular case, the system notices that the key signatures do not match between the staves, although that is not necessarily an error if one staff is for a transposing instrument. A musician reading this score uses the horizontal space between the key signature and the accidentals as a visual cue, as well as the context provided by the key signatures in the other staff and previous staff systems, and an OMR system should use the same information to interpret these symbols correctly. The Musical Semantics specialist should err on the side of caution when constructing the key signature, and only become more tolerant of distances if it improves the internal semantic state.



(a) Key signature requiring context to unambiguously classify



(b) Transposing Clef

Figure 5.25: Examples of difficult notation

Figure 5.25(b) shows a transposing treble clef; in this case, the figure 8 means that the notes should be played one octave lower than they are typeset in the score. The figure sometimes appears on top of the clef instead, which means that the notes should be transposed upwards instead of downwards. This needs to be correctly accounted for, or the pitch calculation for the notes will be wrong.

Chapter 6

Evaluation

The previous chapters have demonstrated how an OMR system can be designed to detect and correct some errors, and have shown that some types of errors are easier to correct than others. Given extra information, it is relatively straightforward to determine a classification for a previously unrecognised object because it is obvious that an unclassified object is wrong. It is harder to detect and correct misclassified objects. The main way this is achieved is via feedback from the higher-level specialists—Primitive Assembly and Musical Semantics where they have detected inconsistencies—to the lower-level Primitive Identification specialist, which makes use of the classifications suggested by them. In some cases, objects can be successfully identified based only on suggested types from the Semantics processing, even for types that are completely unknown to the Identification specialist.

This thesis has described many of the practical decisions and considerations that were made when implementing the various stages of an OMR system. The behaviour of the specialist modules, their interactions, and the effect of the different types of knowledge introduced by them is now measured. The tasks are demonstrated by studying the system’s behaviour in-depth for several input scores.

6.1 Measuring System Performance

For musical scores, there are different levels of re-construction that are acceptable end goals, resulting in various trade-offs to be considered. Some people may be happy with just the pitch and duration information required for audio playback (disregarding score notation details and performance markings), while others may be after just textual metadata and lyrics (such as for a traditional music library).

Someone compiling a large corpus may be satisfied with less accurate reconstructions of the scores if the processing time can be significantly reduced.

While it is fairly straightforward to test the accuracy of the operations involving purely symbolic data (that is, low-level graphical manipulations and primitive assembly), the evaluation and comparison of high-level information “is still a largely unsolved problem” [29]. As with other document image analysis domains (such as mathematical equation recognition, topological map recognition, and electronic circuit diagrams), part of the problem is that modelling the semantics of the domain is complex enough that multiple representations may be semantically equivalent. The level of human perception required to evaluate the semantics makes it difficult to formalise rules describing them for automation, or for comparing the semantic similarity of two different states.

Figure 6.1 shows an assembly error where notes are assigned to the wrong stems, forming chords in the upper voice instead of assigning the lower note to the lower voice (except for the first note, shared between voices). In such a case the audio semantics would not be changed since the correct notes would still be played, but the notational semantics are incorrect since the notes are located in the wrong voice.

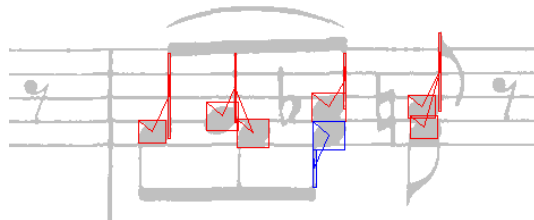


Figure 6.1: Notes assembled incorrectly (first, second and fourth)

6.1.1 Measuring Effectiveness

The OMR system described here is composed of individual specialists that are controlled by a coordinating process. The performance of the specialists’ actions needs to be measured so that the coordinating process can calculate which specialists should be run next, as well as determining when an acceptable state has been reached for processing to finish. To determine whether or not some action is improving the system’s internal representation of the musical score, the following qualities should be measured:

- the effect that processing a request (or set of requests) makes.

- a “goodness” or “completeness” rating for the output from every specialist.
- the difference in “completeness” between two outputs from the same specialist.
- finding good thresholds and tolerance levels for various algorithms.
- the amount of time used by each specialist, including time spent processing their requests.

Unfortunately, the completeness of a particular internal system state can be difficult to quantify. Turning the problem around, an acceptable indicator may be to determine the number of obviously incomplete items, by quantifying:

- the number of unknown or unaccounted for objects.
- how much extra processing each specialist would do if it were given the current state as input.
- some measure of the total certainty of all the detected objects.

6.1.2 Issues in OMR Evaluation

Evaluating the calculated semantics of music is difficult; there can be multiple ways to represent the same music, meaning that there is no single correct representation. Comparing an audio representation to a notational representation requires the interpretation of either one or both instances. Even comparing two instances in the same form raises problems. For audio representations, it is difficult to compare two sounds and quantify how similar they are: comparing the timbre of different instruments, changing volume levels, and timing variations are subjective qualities. Comparing the semantics of visual notation faces similar problems in deciding how closely two scores match; although they may represent identical audio and performance instructions, there are many ways for them to differ visually. Examples of this include:

- the direction of note stems;
- the location of secondary objects such as slurs or other performance directions (above or below the note);
- the use of different glyphs or symbols (for example, using “ $\sharp\sharp$ ” instead of the “ \times ” symbol for double-sharps).

- transposition of notes, such as changing clef rather than using many ledger lines above or below a stave;
- the horizontal spacing of objects, which is important for page layout features such as the bars on a system, and page turns.

Measuring the severity of different errors can be problematic; for example, secondary semantic objects (such as key signatures) can affect a large number of primary objects (notes and rests). Getting a time signature wrong will result in an incorrect number of beats in each bar that follows, causing erroneous additions or subtractions of time events. A wrong key signature will result in incorrect semantics for every note at any of the signature’s mis-detected pitches, even when the note was identified on the correct stave or ledger line. An incorrect clef will result in every note on that stave being assigned a wrong pitch. However, despite the audio semantics being wrong for many notes, this is only due to a single error, and the semantics are arguably “more correct” than if the key signature and clef were identified correctly but each note was identified incorrectly.

There have been several proposals for visual evaluation of notation by people. One method that has been proposed [5] for evaluating the notation visually is to measure the number of changes to the primitives in a hypothetical music notation editor. Different errors could be measured—for example, Table 6.1 shows some typical errors that could be corrected, and an associated “cost” for each correction. This means that a missing accidental or incorrect key signature is only counted as one mistake, rather than as one mistake for every subsequent note that was assigned an incorrect pitch due to the mistake.

Type of error	hypothetical edit	Cost
Note was assigned wrong pitch	modify	0.5
Note was assigned wrong duration	modify (tails/dots)	0.5 each
Missed time event	insert and assign time	1
Erroneous time event	remove	1
Accidental not correctly assigned	insert	1
Key signature mis-detected	insert or remove accidentals	1 each
Clef missed	insert	2
Time signature unrecognised	insert	2

Table 6.1: Hypothetical Cost of Different Recognition Errors

Evaluating the results of different OMR systems introduces extra difficulties; these generally use incompatible and proprietary file formats, and interface directly with a music notation program. While many of these systems can export scores

to the MIDI format, this format is only capable of representing pitch, volume and durational data, and of little use for evaluating notational information. The Interactive Music Network compared several OMR systems by getting volunteers to visually compare the notation of the reconstructed scores [18, 56], as mentioned in Chapter 2 (Section 2.1.10). This was done by measuring fourteen points evaluating symbol identification and symbol relationships. These are:

- notes with correct pitch and duration;
- association of accidentals with notes;
- recognition of rests;
- grouping beamed notes;
- time signatures;
- key signatures;
- symbols (such as accents, trills, and performance markings);
- grace notes;
- slurs, ties and bends;
- durational (augmentation) dots;
- clefs;
- recognition of irregular note groups (tuplets);
- number of barlines/bars; and
- number of staves.

6.2 Evaluation of System Specialists

Evaluating individual specialists is problematic, as often the information calculated by one specialist is used for decision-making by another specialist. As discussed earlier, it is also difficult to quantitatively measure semantics. For the lower-level processing specialists, it is straightforward to measure the accuracy of the graphical processing and identification routines. For the higher-level specialists, the effect of their feedback requests on the identification routines can be measured.

Strengths and weaknesses of the system can be found by examining the effect of various settings on the amount and quality of feedback, both on a per-specialist basis, and a total system basis. One important quality to measure is robustness: how much degradation of the input image can be recovered from? This helps to determine the limits of the system.

The system has a setting that defines how much to loosen tolerance values when a specialist is told to ‘be more tolerant’ in its tests; a smaller value will result in the tolerance value changing slowly, while a larger value may lead to more drastic changes in rule matches. The effect on system performance of changing exactly how much to loosen the tolerance each time can be measured.

Another important setting controls the amount of feedback allowed (both per-specialist and system-wide). The behaviour and performance of the specialists are now examined in detail, in the general order that they are first called.

6.2.1 Preprocessing/Page Layout

The system uses a Hough Transform to detect any page skew if the score was scanned in at an angle. In some instances, it was difficult to measure the true skew angle for input scores because the staff lines and systems on the image were not completely parallel. In most cases during testing, the algorithm calculated the angle to within one degree of what was perceived to be “straight”.

In several instances, the de-skew algorithm completely failed. For one image, the input score image was ninety degrees from upright, and the de-skew algorithm rotated the image ninety degrees clockwise instead of ninety degrees counter-clockwise, resulting in an upside-down score. For situations like this, more document analysis would be required, perhaps by studying any text on the page. For another score, the input image had a large border around the sheet of music, so the transform was performed on that instead of a staff. The longest lines found in this object were running down the page, so these were assumed to be staff lines and the image was rotated to ninety degrees, causing the rest of the OMR system to fail to find the staff systems.

6.2.2 Staff Processing

The major cause of the Staff Processing specialist failing to locate all staff systems and extract the staff lines is too much skew or deformation of the input image. As

discussed in Chapter 3 (Section 3.3), a horizontal projection of the staff system is used to locate the long horizontal stave lines. For deciding which peaks in the projection correspond to stave lines, the threshold is 50% of the system’s width, although this threshold will be made more tolerant if the specialist is called repeatedly (which would indicate that there is a problem finding error-free staves or staff systems).

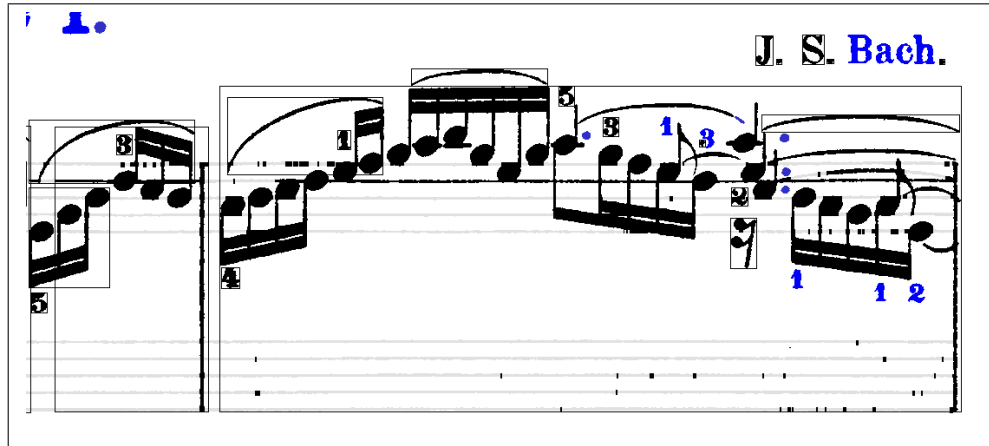


Figure 6.2: An error during staff-line extraction

Figure 6.2 shows an example of the Staff Processing specialist failing to completely remove all the staff lines in a system, leaving one staff line behind. This results in a single, large graphical object linked by this line, rather than a set of isolated graphical objects. The detected staff lines are greyed out, text objects are coloured, and the unknown objects are outlined and in black. While this is not a critical failure, it results in degraded performance for the Primitive Identification specialist in several ways:

- primitive patterns that are defined as ‘isolated’ types, such as rests and accidentals that should not be joined to other primitives, will not be matched (unless or until the object is broken up at a later stage by other primitives being removed, or explicitly broken up by another specialist).
- primitive patterns that are allowed to be joined to other primitives, such as note heads and stems, will take much longer to process the object because of its size—the graphical routines will take time to scan over the mostly empty space.

6.2.3 Object Identification

The OMR system's object identification routines are rather naïve. For example, the template matching used for comparing objects to the primitive pattern descriptions does a straightforward pixel to pixel comparison, when more sophisticated matching routines could be used, such as Compression-based Template Matching [78] or a non-template method such as a decision tree of multiple features.

The reason for this design decision is because the focus is on the coordinated systemic approach correcting mistakes. However, it would be a good idea to test results against a better identification specialist and see if the coordination still has a positive effect or if the effect is merely due to the poor recognition in the first place.

When given semantic feedback about possible types in an area, there are several actions that can occur to match unknown objects to the possibly missing objects:

- naïvely assign this possible type to all the nearby unknown objects, with the certainty inversely proportional to the number of nearby objects and the number of possible types.
- cluster unknown objects of the suggested type, and check if they have the same physical features. For example, if ten different objects have been given the same suggested type, and those ten objects are very similar in appearance, then this could be used as evidence in favour of the suggested classification.

For an example of the first possible action, if there is only one suggested missing object type in a region and there is only one nearby candidate unknown object then it seems more beneficial to assign this classification to the object, as opposed to (say) having three or four different possible types that have been suggested by the semantic analysis for an area that has several unknown objects. The suggested classifications can then be tested by the Primitive Identification specialist for types that have primitive descriptions, or they could even be accepted outright, depending on how conservative the system's settings are.

Compression-based Template Matching

As well as being used for clustering unknown objects, Compression-Based Template Matching (CBTM) is used for comparing unknown objects that do not match any of the pattern descriptions to already-classified objects.

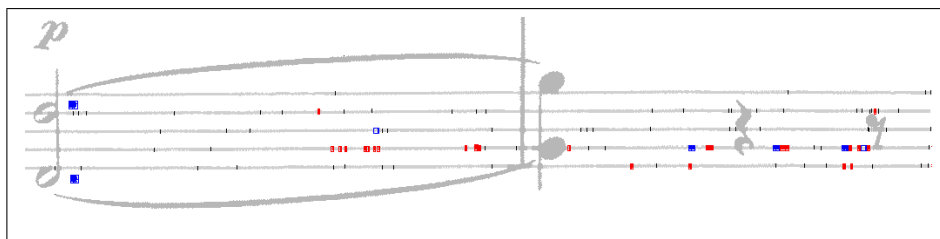
CBTM, as implemented in this system, provides a form of adaptation in the patterns. Since each run of matching unknowns can test against objects found in the previous runs, the set of identified objects may become more and more divergent from the original templates described in the user-supplied patterns. Figure 6.3 shows that the majority of objects matched by CBTM were correct; most of the false positives were parts of objects fragmented by staff line processing that were similar enough to “dots” to match an existing object of that type. For the column labelled “Correct” in Figure 6.3(a), this means that the matched (previously unknown) object has the same graphical shape as the template (previously recognised) object. That is, the evaluation is on whether the two bitmaps were successfully identified as being the same type of object, not whether the object type was identified correctly, for instances where the known object was mis-classified. The majority of errors in the evaluated scores are for objects that look like dots. These results suggest that CBTM should either not be used for very small objects, or should use different threshold settings based on the object’s size and shape.

Score	Matches	Correct	Common Errors
<i>Promenade</i>	55	49 (89.1%)	5×dot (see Figure 6.3(b))
<i>Clarinet Concerto</i>	129	80 (62.0%)	37×dot (see Figure 6.3(c))

(a) CBTM bitmap statistics



(b) Some of the erroneous matches (left: dots, right: rectangular rest)



(c) Dots found due to bad staff line removal

Figure 6.3: Evaluation of Compression-based Template Matching

Recognition Rates

An example of Primitive Identification performance is now given. The performance is affected by the other specialists, but this section gives an indication of how the Primitive Identification specialist fits into the system. More comprehensive data on

the number and types of identified primitives is provided later in this chapter.

Table 6.2 shows the number of classified objects for several primitive types at different points in time. This shows the changing recognition rates as the system progresses through the different specialists and re-evaluates decisions based on new contextual information. The system’s behaviour depends significantly on various configuration settings; these results are for a configuration set that was heuristically found to give acceptable performance. The following definitions are used as column headings in the table:

Time is how many times any specialist has performed an action since the processing started. This value increases each time control passes to another specialist, so an arbitrary number of feedback requests may have been processed between two listed time points; only actions that resulted in a change to the state at the previous time are listed.

Wrong means an object of a different type was incorrectly identified as this type.

Irrelevant means a minor mistake was made (such as a leftover sliver from a vertical line is recognised as another vertical line), but did not modify an object enough to prevent it from being correctly recognised.

Missed means an object of this type was recognised as a different type, or was unidentified.

Ignored means that an object of this type was correctly identified, but later (incorrectly) re-classified—for example, based on syntactic information suggesting that this classification was wrong.

The table shows that the number of recognised vertical lines fluctuates over time. This is due to conflicting information generated by different specialists; because vertical lines should either be part of a note, or a barline, the Musical Semantics specialist says that any vertical lines that are not part of a note (nor a barline) were incorrectly identified and should be re-recognised as “something else”. Figure 6.1 (on page 110) shows one of the causes of this—an assembly error has assigned a notehead to the wrong vertical line, creating a chord and leaving a vertical line unassigned when it is clearly a stem. Because this vertical line is not part of a note, and is not a barline, the Primitive Identification specialist is asked to re-classify it, as anything other than a vertical line. If it is re-classified back to “unknown”, it

Hollow Note Head	Time	Identified		Unidentified		
		Correct	Wrong	Missed	Ignored	
	4	0	0	11	0	
	6	7	0	4	0	
	13	9	0	2	0	
	16 final	10	0	1	0	
Flat	Time	Identified		Unidentified		
		Correct	Wrong	Missed	Ignored	
	4	8	1	35	0	
	8	9	1	34	0	
	23	29	1	14	0	
	47 final	30	1	13	0	
Vertical Line	Time	Identified			Unidentified	
		Correct	Wrong	Irrel.	Missed	Ignored‡
	4	160	42	92	0	0
	6	160	42	94	0	0
	23	136†	0	15	0	8
	47	140	11	37	0	4
	71	135	0	20	0	9
	90	135	0	20	5	4
	105	140	20	27	0	4
	128	136	0	21	4	4

† At this point, 16 vertical lines were (correctly) renamed by the syntax processing to a new ‘barline’ type.

‡ The majority of ‘ignored’ vertical lines are the result of syntactic rules not allowing note heads to belong to two voices, so the second (correctly identified) vertical line was thought to be erroneous.

Table 6.2: Primitive Identification results for *Promenade*

may then later be joined to nearby unknown objects by the Segmentation specialist to create a new object, and this new object may be partially or wholly matched against one of the vertical line pattern descriptions.

The score images used while testing and evaluating the OMR system were generally acquired at a resolution of 300 DPI. While the primitive pattern descriptions were designed to be scaled to an arbitrary staff height, the system performed poorly on a low resolution image. It was tested with an image that had a resolution of 122 DPI, and because of their small size, many noteheads were identified as large dots and many note beams were classified as rectangular rests.

6.2.4 Primitive Assembly

The Primitive Assembly specialist uses a configured ruleset to join some primitive types into musical objects. As described in Chapter 5, these rules are also used in a predictive manner to suggest types for unknown objects that would pass any of the rules if they had the appropriate classification. This can result in hundreds of suggestions, mostly for the large number of small unknown objects that are merely fragments left over after legitimate objects have been identified and extracted from the image. A closer inspection of these suggestions and the effect they have on system performance is now given.

Figure 6.4 shows the objects identified after the Primitive Assembly specialist has made suggested classifications for unknown objects based on the assembly rules. These are the results for the first time that this specialist is called while the system is processing the first page of *Promenade* by Mussorgsky; the following times Primitive Assembly is called, there are fewer suggestions because there are fewer suitable objects still left unidentified. In this figure, four objects—three hollow noteheads and one full notehead—were tested and identified as one of the types suggested by the assembly (shown in black), and seven unknown objects were correctly identified as types that were not suggested for them, shown in blue. These objects were identified despite not having the correct type suggested in the request because they were matched to objects that had not been found when the Primitive Identification specialist was previously called; for example, the hollow notehead shown in blue was matched to one of the hollow noteheads that was just classified earlier in this same processing run, based on suggested types. There are also several objects that classified incorrectly. There are five full noteheads, shown in red, that were identified

Wladimir Wassiljewitsch Stassow gewidmet

Bilder einer Ausstellung
Erinnerung an Viktor Hartmann
1874

Promenade

Allegro giusto, nel modo russo, senza allegrezza, ma poco sostenuto

The musical score is presented in four systems, each with a treble and bass staff. The first system begins with a forte (f) dynamic. The score includes various musical notations such as notes, rests, and dynamic markings. There are several red and blue annotations on the score, including a red 'x' above a note in the first system, a red 'x' above a note in the second system, and blue markings (a '2' and a 'b') in the third system.

© 1984 by Wiener Urtext Edition, Musikverlag Ges. m. b. H. & Co., K. G., Wien
Urtext Edition No. 50076

Figure 6.4: Objects identified after Primitive Assembly suggestions

as the type suggested by the Primitive Assembly specialist, but the suggestions were wrong. This is due to the beams not being identified, and the primitive assembly rules suggesting that an unknown object that is touching a vertical line could be a notehead.

6.2.5 Segmentation

The Segmentation specialist modifies the set of objects that are persistently unrecognised by joining nearby objects together to create new objects that will hopefully be recognised by the Primitive Identification specialist. There must be a limit on the number of times that the specialist joins nearby unknown objects together, otherwise it would be possible to chain objects together to create new objects that are unlikely to be legitimate; the main reason for this specialist's creation was to repair segmentation caused by staff line processing. There are two actions that the Segmentation specialist can perform after it has created new objects by joining old objects:

1. Immediately generate a request for the new objects to be recognised; or
2. Leave the items as “unknown” objects and leave them for other specialists to use semantics or other information to identify them in context.

The tunable settings for this specialist are the maximum number of times that the Segmentation specialist performs its default action (of joining objects together), and the maximum number of times that the specialist can request that the new unknown objects are immediately processed for identification. By requesting immediate processing of the new objects, execution passes to the Primitive Identification specialist before control is given to the Segmentation specialist again. Restricting this behaviour means that control will pass on to the higher-level specialists, such as Primitive Assembly.

Table 6.3 shows the effect of increasing the number of times that the Segmentation specialist will merge nearby objects together into new objects. At first, increasing the maximum execution count results in several of the newly formed objects being successfully identified. Further increases in the maximum execution count result in decreased accuracy, with more objects being mis-identified. In this table, the following column headings may need more explanation:

Allowed means the maximum number of times that the Primitive Segmentation

specialist was allowed to perform its default action, which is to join nearby unknown objects into a new, larger unknown object.

Destroyed means that the newly combined object subsequently either: i) was itself joined to another object to create another new object, ii) had a matched object extracted from it, so new objects were created from the remnants, or iii) had one of the object’s components removed and reverted back into its original object.

For example, the third row shows that when the specialist was allowed to be called twice and allowed to generate four requests (to the Primitive Identification specialist), it created thirty eight new objects. Of those, twenty nine could not be recognised, four were broken up or themselves joined into new objects, four were correctly identified (for objects that had previously been segmented), and one of those new objects was wrongly identified. With this setting, the system finished processing with 247 unknown objects, which includes the 29 newly joined objects that were unrecognised. This table shows that the Segmentation specialist can reduce the number of unknown objects by joining nearby objects together and correctly recognise some fragmented objects. Increasing the number of times that objects may be joined together leads to more correctly identified primitives. However, this comes at the expense of finding more incorrectly identified primitives.

Allowed Calls	Final Unknown	Joined Objects				
		Created	Destroyed	Identified	Unidentified	Mis-identified
0	267	0	–	–	–	–
1	249	32	3	3	25	1
2	247	38	4	4	29	1
3	246	39	4	4	30	1
4	245	40	4	4	31	1
5	244	41	4	4	32	1
6	235	82	28	5	46	3
7	235	92	31	5	53	3
8	230	119	44	5	67	3
9	218	131	45	5	77	4
10	198	151	45	5	97	4

Results for processing the first page of *Promenade*. The Segmentation specialist is allowed to generate four requests, while the other specialists are not allowed to generate any.

Table 6.3: Modifying the number of actions allowed by the Segmentation specialist

Figure 6.5(a) show some examples of unknown objects joined together by the segmentation specialist being correctly reconstructed and identified (although sev-

eral other flats fragmented by the staff line removal were joined back together but not successfully recognised). Figure 6.5(b) shows some objects that were formed by the Segmentation specialist and remained unknown. The two small lines in the top right were joined into a new object that was identified as a ‘small dot’. By allowing the Segmentation specialist to run multiple times, such objects may themselves be joined to form new unknown objects; several of the objects in this figure (including one of the successfully identified flats) consist of at least three distinct unknown objects.

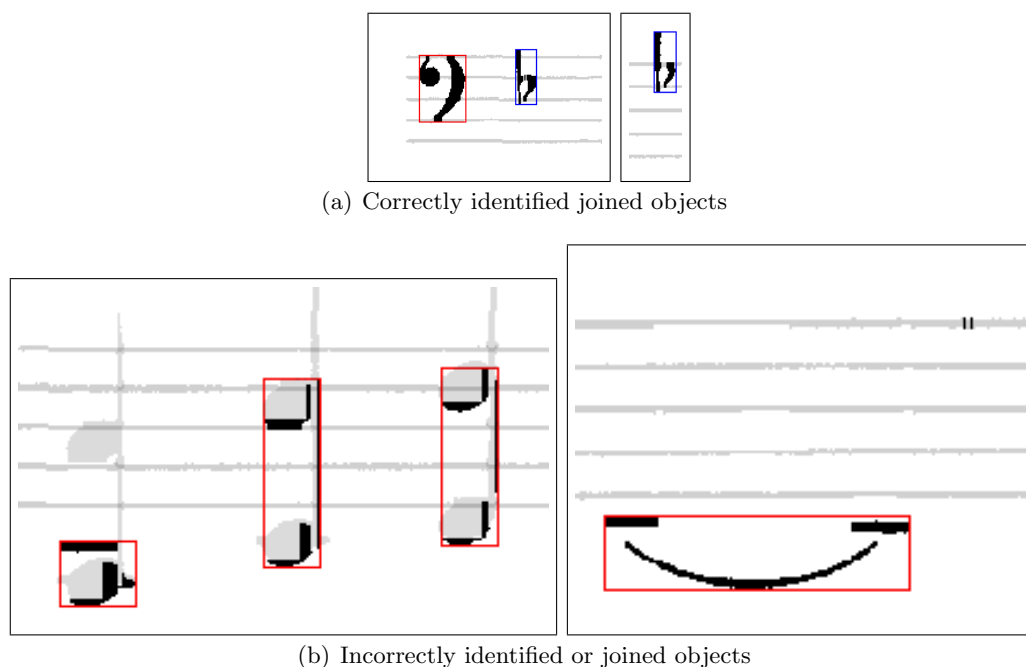


Figure 6.5: Example objects created by Segmentation specialist

6.2.6 Text Removal

There is a measurable performance increase to the OMR system due to the removal of characters by the Text Processing specialist. This performance increase is due to less processing time being spent performing Primitive Identification on the set of unknown objects. When Text Processing is not performed, any text in the image is treated as unknown objects. As discussed earlier, the Text Processing specialist does not go as far as to perform Optical Character Recognition to identify individual characters, but classifies objects as ‘text’.

Table 6.4 shows that identifying textual regions results in a nearly 6% reduction in the total processing time (including the time spent performing text removal). These results are for the first page of *Promenade* by Mussorgsky, running on a

1100MHz Athlon CPU, and all CPU processing times are in seconds.

Without Text Processing						
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Prim. Identification	31.32	31.29	31.29	31.29	31.30	31.298s
Total processing time	44.72	44.56	44.57	44.61	44.58	44.608s
With Text Processing						
Text processing	0.25	0.25	0.24	0.24	0.24	0.244s
Prim. Identification	28.41	28.42	28.4	28.42	28.4	28.410s
Total processing time	42.01	42.05	41.97	42.08	42.03	42.028s

Table 6.4: Performance considerations of Text Processing

Score	Correct [†]	Missed Objects	Incorrect
<i>Promenade</i>	210 (95.02%)	11 (4.98%)	4
Mozart <i>Clarinet Concerto</i>	33 (89.19%)	4 (10.81%)	5

[†] Touching characters that only formed one single graphical shape are counted as one.

Table 6.5: Classification of Text Objects

The algorithm used for deciding if an object is textual is fairly robust, despite being rather simplistic. Table 6.5 shows the accuracy of this algorithm for several scores. “Missed” text objects are text characters that the Text Processing specialist did not classify as text, while those counted as “Incorrect” are non-text objects that were incorrectly flagged as text (such as musical objects that are not physically touching a staff system). These errors might be corrected with further processing by a full Optical Character Recognition implementation. For example, the objects falsely labelled as possible characters would not be recognised as text by OCR so the system could re-classify them as unknown musical objects for the Primitive Recognition specialist, while all unknown objects could also be run through OCR to see if they are really text or lyrics.

Figure 6.6 shows the detected (in black) and missed (in red and outlined) text objects for an extract of the first page of Mozart’s *Clarinet Concerto*. The two noteheads were detected as text because they are not attached to the staff system (partly due to the de-skew algorithm and the thinness of the stem line in the original image) and because being near another suspected text object (each other) increased the certainty that they were textual objects. The *p* dynamic markings were detected as suspected text objects but because they were not near other suspected text objects, they remained classified as “unknown”.

Incorporating OCR as another knowledge source means that these possible classifications could be checked, resulting in higher accuracy for confirming or rejecting

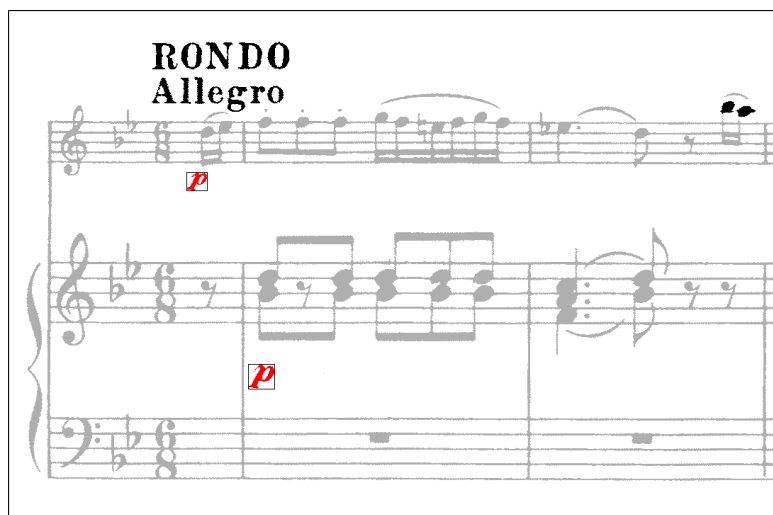


Figure 6.6: Example of detected, mis-detected and missed text objects

objects on the page as being text.

6.2.7 Musical Semantics

Although the OMR system was designed to be extensible, with flexibility in expressing Musical Semantic rules to allow arbitrary music notations to be represented, only rules for Western Music Notation have been implemented. Consequently, all the evaluation of the Musical Semantics specialist and its interaction with the rest of the OMR system described here is for this notation.

For Western Music Notation, the Musical Semantics specialist calculates the musical attributes of the (possibly assembled) musical primitive objects, and then creates a two-dimensional lattice linking the objects to represent their interactions. This can introduce different types of errors:

- Calculating incorrect attributes for the primitives (such as pitch or base duration).
- Lattice-creation errors (for instance, when deciding which notes should occur simultaneously).
- Incorrectly identified or missed primitives leading to inconsistencies or syntax errors in the lattice (such as the wrong number of notes in a bar).

Duration Calculation

The Musical Semantics specialist calculates the duration of primitives and assembled objects by first assigning a base duration for an object type, and then applying any

modifiers such as durational dots, or beams and quaver tails.

In the OMR system, the Primitive Assembly specialist is responsible for assigning these modifiers to the objects (for example, by creating a “hollow_notehead_dur_dots” object composed of the relevant objects). The semantic rules then examine the base objects that have been assigned durations, and determines whether any of the modifying types have been joined to the object in this manner. This means that any errors in the calculation of an object’s duration is almost certainly an error in Primitive Assembly or Primitive Identification.

The Musical Semantics specialist joins the individual objects together and checks the inter-relationships for consistency; for example, to find simultaneously occurring notes, and to ensure that bars have the same number of beats as dictated by the time signature.

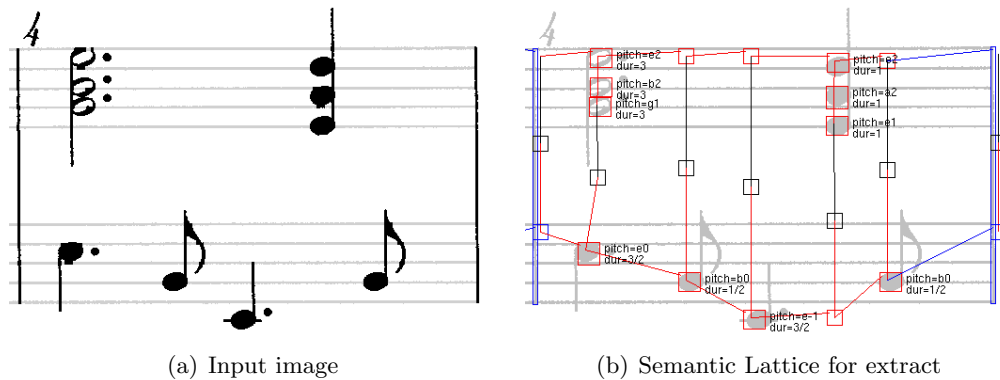


Figure 6.7: Extract of a score with syncopated rhythm

The extract in Figure 6.7 shows a bar with a syncopated rhythm. Moving from left-to-right, each position that has a node for a time event (either a note or a rest), empty “time-fill” nodes are inserted into the other staves that do not have a corresponding event, and the vertical set of nodes at that horizontal position (for all staves in the system) is referred to as a “timeslice”. All events in the same timeslice occur simultaneously.

To check consistency between staves (for multi-stave scores), the offset from the start of the bar is calculated for each non-empty node. The duration between two consecutive timeslices is not merely the shortest duration that occurs in the first timeslice; in the example above, the fourth timeslice (the second chord in the upper staff) occurs one beat after the third timeslice (the lowest note in the lower staff), not one and a half beats. For each node, the ‘potential bar offset’ is the offset of the staff’s previous non-empty time event plus its duration. The bar offset assigned to

the whole timeslice is the minimum ‘potential bar offset’ of each node in the timeslice. Table 6.6 walks through this calculation for the example figure. For each timeslice, the possible offset from the previous nodes in each of the upper and lower staves is calculated, and the lesser of the two—underlined—is the correct assignment. The sum of the offset and duration for the last non-empty node should be the same in every stave—this is used to check the consistency of the calculated object durations in the bar. Figure 6.8 shows these calculated offsets for each timeslice.

Timeslice	(upper stave offset, lower stave offset)	assigned offset
1		0
2	$\text{minimum}\{ \underline{0 + 3}, 0 + \frac{3}{2} \}$	$1\frac{1}{2}$
3	$\text{minimum}\{ 0 + 3, \underline{\frac{3}{2} + \frac{1}{2}} \}$	2
4	$\text{minimum}\{ \underline{0 + 3}, 2 + \frac{3}{2} \}$	3
5	$\text{minimum}\{ 3 + 1, \underline{2 + \frac{3}{2}} \}$	$3\frac{1}{2}$
(end of bar)	$\text{minimum}\{ 3 + 1, 3\frac{1}{2} + \frac{1}{2} \}$	4

(The offset for an event is calculated by the sum of the previous event’s offset and duration).

Table 6.6: Calculation of time-slice offsets and durations

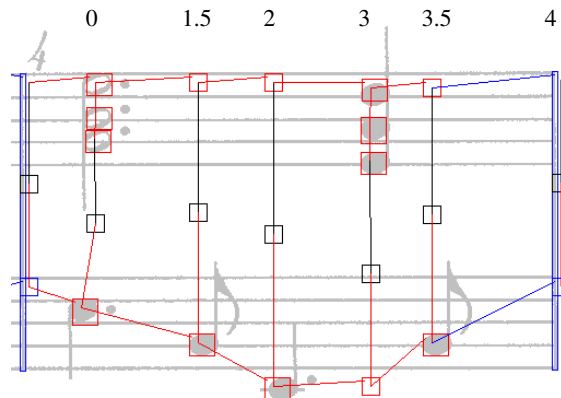


Figure 6.8: The time-slice offsets for the example figure

Pitch Calculation

Pitch calculation of notes is performed by a relatively simple algorithm: the average gap between staff lines is used to determine how far the middle of the notehead is from the lowest staff line in the stave, and it is assumed that pitch is only dependent on this. Any key signatures and accidentals are then applied to this ‘base’ pitch to get the final pitch.

After correcting mis-detected clef primitives (and excluding any unrecognised notes), Table 6.7 shows that most notes have the correct base pitch calculated (that is, ignoring the effect of any key signatures and accidentals).

Input Score	False Noteheads	Correct Pitches	Incorrect Pitches
<i>Clarinet Concerto</i>	23	298 (97.8%)	10†
<i>Promenade</i>	2	256 (100%)	0

† excludes a badly detected note (see the second-to-last bar in Figure 6.11)

Table 6.7: Pitch Calculation algorithm accuracy

Reasons that this algorithm can calculate pitches incorrectly include:

1. primitive identification errors, leading to noteheads being given the wrong dimensions.
2. the image being slightly skewed or distorted, so that towards the edge of the page, the calculated position of the bottom staff line in the stave becomes further away from the actual position.
3. ledger lines being badly typeset or positioned, resulting in a badly positioned note head.

Figure 6.9 gives a closeup of a note that has been assigned the wrong pitch due to the first reason listed above; the pattern has also matched some of the space under the note, resulting in a larger perimeter and moving the centre of the detected notehead downwards. Since the algorithm uses the position of the note’s centre, it has been judged to be closer to the gap between stave lines rather than being on the middle stave line. This could be improved by using a better method for calculating the centre of the notehead rather than merely using the halfway points of the width and height; for example, the average position of all the black pixels, or “first-order bitmap moment” mentioned in Chapter 3.

As well as the failures in the algorithm listed above, another reason for pitches to be calculated incorrectly is the use of notation-specific symbols that modify pitches in some way. Figure 6.10 shows a construct that transposes notes. The notes directly under the “8” and the horizontal line are meant to be played one octave (eight notes) higher, but were drawn on the stave for space and readability reasons. The transposition remains in effect until the *loco* marking (which literally means “in place”). Constructs such as this will require extra semantic rules and processing (including Optical Character Recognition to understand the semantics of the text) to correctly calculate the pitches of the affected notes.

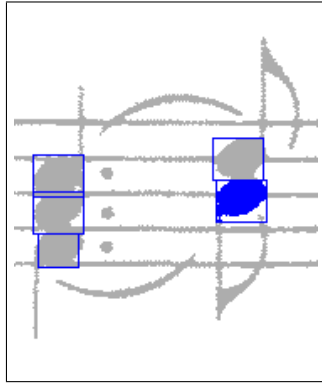


Figure 6.9: A note with incorrectly calculated pitch



Figure 6.10: An octave marking

6

RONDO
Allegro

The musical score is for a piece titled "RONDO Allegro" in 6/8 time and B-flat major. It is divided into four systems of staves. The first system shows the beginning with a piano (*p*) dynamic. The second system includes dynamics *p*, *cresc.*, *f*, and *p*, with a measure number "10" above the staff. The third system includes *cresc.*, *f*, and *p* dynamics. The fourth system includes a *p* dynamic. Notes are color-coded: incorrect notes are solid blue, and correct notes are outlined in blue. Some notes are also highlighted in red.

H. 15737

Figure 6.11: Pitch calculation: Incorrect notes coloured, correct notes outlined

Figure 6.11 graphically shows the calculated pitches for Mozart’s *Clarinet and Piano Concerto*. All detected noteheads are outlined and incorrectly found noteheads are filled in red, while noteheads with an incorrectly calculated pitch are filled in blue. The cluster of incorrect notes in the top right appears to be caused by slight page deformation and skew. Most of the other pitch calculation errors are due to the primitive identification recognising a notehead in a slightly offset or larger (or smaller) area than the notehead physically occupies.

Lattice Creation Errors

The lattice is created by first creating and linking nodes for objects on a per-stave basis, and then linking the nodes between each system’s staves based on alignment (as discussed in Section 3.1). This process will introduce errors in several circumstances:

1. notes that should be played simultaneously are not aligned (as shown in Figure 3.1(b) on page 41);
2. some barlines were not recognised, leading to multi-part staff systems having an inconsistent number and duration of bars between parts.

The graph structure needs to be modified to correct such mistakes; this requires explicit support from the semantic rules created for each music notation.

For problems of the first type, the side-by-side notes often belong to different voices (for example, one voice with stems drawn upwards, and one voice with stems drawn downwards) so this may be a useful indicator for any semantic rules attempting to detect this error. Figure 6.12 contains another example of this—the chord at the start of the bar occurs at the same time as the first quaver in the notegroup. This is one of a set of difficult examples given by Byrd [20].

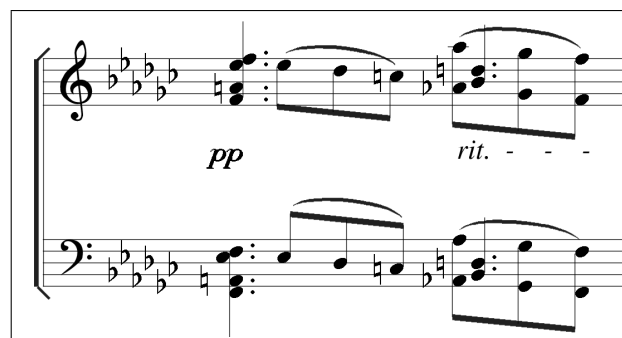


Figure 6.12: An extract from Intermezzo Op. 117 number 1, by Johannes Brahms

Problems of the second type could be eased by assuming that there should be a barline at the appropriate position in the staves that do not have a matching barline. However, a better ‘overall’ approach would be for the system to use the extra knowledge that there should be barlines in those positions to re-examine that area, and alternately to double-check that the unmatched barline really is a barline and not merely a wayward or otherwise mis-detected vertical line.

Other Semantics

Some preliminary work on using the distribution of base pitches (that is, the calculated pitch disregarding accidentals and key signatures) to predict the key signature showed some promise. However, it was not accurate enough to warrant continued development, and did not differentiate between ‘major’ and ‘minor’ key signatures. If the key could be indicated by the pitch distribution, this information could be exploited to look for missing accidentals in any incorrect key signatures. The weightings used to calculate the key signature from the cumulative durations for each pitch appear to be genre-specific, as most classical pieces were classified correctly, although this may be caused by the weighting algorithm being developed using the classical scores as the test images.

6.3 Evaluation of System Interaction

The previous sections have evaluated the effect of each specialist in relative isolation. This section explores the interaction between the specialists and measures various characteristics of the coordinating process.

6.3.1 Request Handling

After a specialist has performed an action, it may have created requests for several reasons: to suggest alternative classifications for objects, or for any detected problems that it is not directly responsible for or capable of handling. The coordinating process is responsible for deciding which, if any, requests are to be accepted for further processing, and determining their order of acceptance.

Order of Service

Requests are assigned a *priority* on creation, based on the severity of the problem that the generating specialist has encountered.

Normally, the coordinator will choose outstanding requests for service in order of priority, until either all requests have been served, or each specialist has reached its maximum allowed requests served. Table 6.8 shows that when the maximum total number of requests performed system-wide is limited to 200, the number of primitives and assembled primitives were identical, regardless of whether the coordinator always chose the request with the highest priority, or always chose requests randomly. The semantics of both processing runs were also identical.

Objects Eventually Found	Coordinator Request Strategy:	
	Highest Priority	Choose Randomly
barline	16	16
bass_clef†	3	3
bass_clef_curl	3	3
composite	454	454
crotchet_rest	4	4
digit_4	24	24
digit_5	12	12
digit_6	12	12
dot	61	61
filled_note_stem_down†	36	36
filled_note_stem_up†	109	109
filled_notegr_stem_down†	15	15
filled_notegr_stem_up†	51	51
flat	31	31
full_notehead	244	244
hollow_note_stem_down†	5	5
hollow_note_stem_up†	3	3
hollow_notehead	9	9
hollow_notehead_dur_dots†	5	5
hough_horizontal	21	21
keysig†	6	6
noise	1060	1060
notegroup†	11	11
quaver_rest	1	1
rect_rest	5	5
staff_line	40	40
staff_system	4	4
text	214	214
timesig†	24	24
treble_clef	4	4
unknown	385	385
vertical_line	155	155

† denotes objects created by Primitive Assembly. Results for the first page of *Promenade*, by Mussorgsky.

Table 6.8: (Lack of) effect of request processing order

This seems counter-intuitive. The explanation for this is that after a specialist has performed its default action and generated a set of requests, all of those requests will be processed until the global maximum number of requests have been serviced. It seems likely that in general, each request targets a specific area or problem and does not overlap with other requests so their order does not matter, and that multiple requests for the same area or problem cause the same resolution, so again order does not matter. This table also seems to demonstrate that after 200 requests have been processed, any further requests have little effect on the state.

This may also indicate a weakness with the allocation of request priorities. A request is assigned a priority when generated, based on how much the specialist thinks that the system's internal state will be improved if the request is serviced. For example, if the Staff Processing specialist finds several skewed staff systems, it will generate a "de-skew" request with a priority of 50 (out of 100), but will give the request a higher priority of 90 if $\frac{2}{3}$ or more of the systems are skewed. If no staff systems at all are found, a request is generated with a priority of 100. However, the numbers given to particular requests are somewhat arbitrary, and a more careful consideration of these priorities may affect the behaviour of the system when choosing and servicing requests.

Maximum Number of Requests

The coordinator will not pass on requests to individual specialists that have reached their maximum number of actions performed. (However, specialists may still perform more actions than this limit if it is the result of execution continuing from the previous specialist.) Similarly, specialists also have a maximum number of generated requests that will be considered. Finally, requests may also be rejected if a specialist has passed a limit on the amount of CPU time the specialist had already used.

Table 6.9 shows the limits and resource usage of the OMR process for several input scores. The third table (for Chopin's *Scherzo*) shows that the OMR system performed a CPU-intensive function in the Page Layout specialist (that did not occur for the other listed score). This was the de-skew algorithm determining the skew angle, rotating the page, and deleting all previously found objects from the system's internal state; a de-skew on the input page was requested by the Staff Processing specialist, which determined that several of the detected staff systems were skewed, leading to poor recognition of the staff lines. (See Section 6.3.3 for a more detailed

analysis of the order of requests for this score.)

Promenade page 1—maximum of 50 requests accepted by coordinator:

Specialist	Called (Max)	CPU (Max)	Reqs Made	Accepted (Max)
Page Layout	1 (5)	2.04 (20)	0	0 (3)
Text Processing	1 (5)	0.12 (20)	0	0 (3)
Staff Processing	1 (5)	0.52 (20)	0	0 (3)
Prim Identification	51 (200)	25.21 (300)	0	0 (3)
Prim Segmentation	9 (5)	0.05 (20)	7	3 (3)
Prim Assembly	6 (5)	1.34 (20)	6	3 (3)
Musical Semantics	3 (5)	3.96 (20)	84	44 (100)
File Output	1 (5)	0.03 (20)	0	0 (3)

Promenade page 1—maximum of 150 requests accepted by coordinator:

Specialist	Called (Max)	CPU (Max)	Reqs Made	Accepted (Max)
Page Layout	1 (5)	1.89 (20)	0	0 (3)
Text Processing	1 (5)	0.11 (20)	0	0 (3)
Staff Processing	1 (5)	0.50 (20)	0	0 (3)
Prim Identification	107 (200)	30.84 (300)	0	0 (3)
Prim Segmentation	13 (5)	0.08 (20)	10	3 (3)
Prim Assembly	10 (5)	2.45 (20)	10	3 (3)
Musical Semantics	7 (5)	10.41 (20)	184	100 (100)
File Output	1 (5)	0.03 (20)	0	0 (3)

Chopin's *Scherzo*—no limit on number of requests accepted by coordinator:

Specialist	Called (Max)	CPU (Max)	Reqs Made	Accepted (Max)
Page Layout	2 (5)	114.16 (90)	0	0 (3)
Text Processing	2 (5)	0.32 (90)	0	0 (3)
Staff Processing	2 (5)	2.06 (90)	2	1 (3)
Prim Identification	108 (200)	128.84 (300)	0	0 (3)
Prim Segmentation	9 (5)	0.15 (90)	4	4 (4)
Prim Assembly	5 (5)	2.90 (90)	5	3 (3)
Musical Semantics	2 (5)	27.36 (90)	348	100 (100)
File Output	1 (5)	0.06 (90)	0	0 (3)

(Time is CPU seconds on an Athlon 1800MP.)

Table 6.9: OMR Resource usage and limits for several scores

Table 6.10 shows the effect of changing the maximum number of feedback requests that the coordinator will allow to be serviced. The results of modifying the maximum number of requests is difficult to quantify, because the final state is also dependent on the limits of the individual specialists, which each have their own maximum number of requests created and received.

Often there will be a series of requests that each make a small (or no) modification, followed by a single request that has a much larger impact (illustrated in the following subsection). For example, some specialists generate a request for each fairly trivial suggestion for an alternative object type, and some specialists generate a request that causes another specialist to process or modify its complete internal

Max. requests:	0	5	10	15	20	25	30	35	40	...	200
Object Type	Number Identified										
bass_clef†	0	3	3	3	3	3	3	3	3	3	3
bass_clef_curl	0	3	3	3	3	3	3	3	3	3	3
crotchet_rest	0	0	0	0	0	0	4	4	4	4	4
digit_4	24	24	24	24	24	24	24	24	24	24	24
digit_5	12	12	12	12	12	12	12	12	12	12	12
digit_6	12	12	12	12	12	12	12	12	12	12	12
filled_note_stem_down†	36	36	36	36	36	36	36	36	36	36	36
filled_note_stem_up†	108	108	108	108	108	108	108	108	108	108	109
flat	9	10	30	30	30	30	31	31	31	31	31
full_notehead	242	243	243	243	243	243	243	243	243	243	244
hollow_notehead	0	10	10	10	10	10	10	10	10	10	10
keysig†	2	2	7	7	7	7	8	8	8	8	8
quaver_rest	1	1	1	1	1	1	1	1	1	1	1
timesig†	24	24	24	24	24	24	24	24	24	24	24
treble_clef	4	4	4	4	4	4	4	4	4	4	4
vertical_line / barline	279	281	154	154	154	154	189	189	189	189	155
Processing Time (secs)	19.0	24.1	27.3	27.2	27.3	27.5	32.1	31.9	32.3	...	46.3

† denotes objects created by assembly.

Table 6.10: Effect of changing the system’s maximum number of feedback requests allowed—*Promenade*

state (such as a request to re-process all unknown objects, or to perform a de-skew on the input image).

Limiting the system’s maximum number of requests also favours requests generated by the earlier specialists, since they have the first chance to have all of their requests serviced.

6.3.2 Tolerance Levels

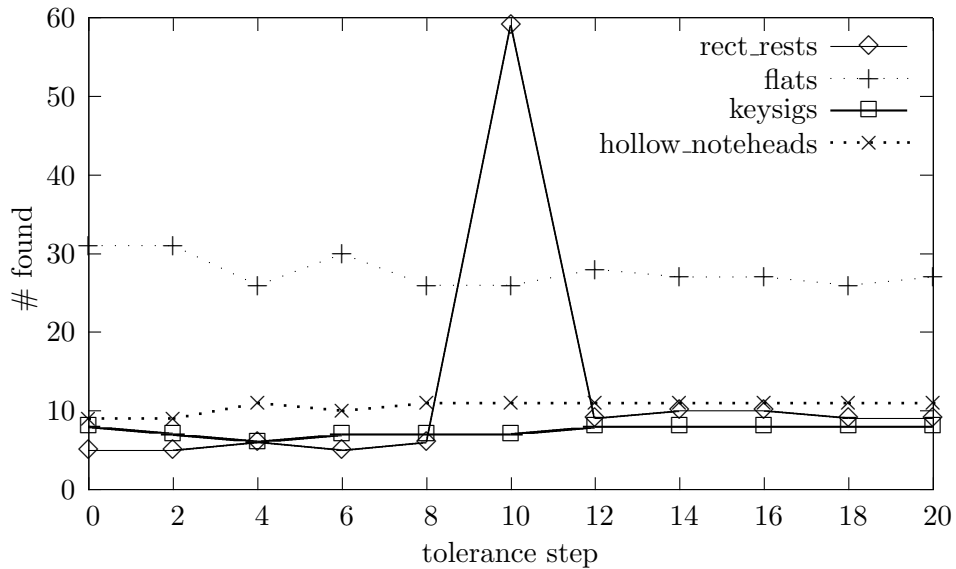
The *tolerance step* is a coordinator setting describing how much the specialists’ thresholds for various tests should be loosened. The idea is that if a task is repeatedly requested, the threshold for that test should be lowered. The tolerance step is merely how much the threshold is lowered by each time. Specialists use a tolerance between 0 and 100, with 100 being most tolerant. For example, the Primitive Identification specialist will classify more unknown objects as primitive types if the limits in the pattern descriptions are not applied so tightly, and the Primitive Assembly specialist will join primitives together even if they are slightly further apart than the rules normally allow.

A larger increase in the threshold tolerances means that an object that is unrecognised because it did not match the relevant pattern description may be more

likely to match an incorrect pattern rather than the sensible one. For example, an unrecognised flat that is just under the thresholds when compared to the pattern for flat objects may be matched as a natural (h) rather than a flat (b) if the thresholds are greatly increased the next time the classification algorithms are run. Smaller increases may help match such an unrecognised object against the correct pattern template without unduly increasing the chance of matching some other pattern type, especially if it was previously just under the required threshold to match against the correct type. On the other hand, if the rate of change in the tolerance is too small, then the object may not be recognised before the Primitive Identification specialist has reached its maximum resource limits and is not run again.

Experimentally, the best results were achieved with a tolerance step of 5%. How each specialist modifies its behaviour to account for the tolerance is defined on a per-specialist basis—it does not automatically mean that maximum distances or pixel counts are increased by that percentage, for example. However, the effect of this setting on the system behaviour is inter-dependent on the other system settings, such as the per-specialist and global request limits, so a different configuration of these factors may mean that a different tolerance step setting leads to better performance.

Table 6.11 shows the effect of changing the default step between 2 and 20 on primitive and assembled objects. Increasing the rate of change in the tolerance level results in more objects being recognised, at the expense of making more classification errors. This table is summarised graphically for several object types in Figure 6.13(a). As the default step increases, recognition performance becomes more volatile; for example, several flats that were unrecognised when the system runs with low tolerance steps are recognised when using a higher step, but several flats that were previously recognised are now unidentified or misidentified as other object types. The obvious stand-out feature of this graph is that the number of *rectangle_rests* found jumps dramatically when the tolerance step is 10. This is because most of the short, horizontal lines (such as ledger lines and accent markings) were determined to match a *rectangle_rest*. Figure 6.13(b) shows the *rectangle_rests* found on the first staff system when using this setting. The three larger matches (shown in red) matched the pattern template, while those in other colours were matched to identified rests by compression bitmap template matching on subsequent calls to the Primitive Identification specialist. This peculiarity does not happen at the other tolerance step levels because it is a side-effect of another object being recognised and



(a) The tolerance step's effect on recognition of several object types



(b) Rectangular rests found with too low tolerance thresholds

Figure 6.13: A graphical representation of the tolerance step's effect

removed from a larger object, and the leftover piece of the bitmap that first matches a `rectangle_rest` object is a different size depending on the tolerance step. After evaluation, this particular problem was solved by creating a new pattern that would recognise horizontal bars such as ledger lines and the stress accents shown in the figure instead of leaving them as unrecognised objects that might be mis-classified as another object type.

6.3.3 Flow of Execution

The order that specialists are processed in is now examined. Of course, the behaviour of the OMR system for any particular input score is determined by the configuration settings specifying global and per-specialist resource limits and variables. These settings control things such as the maximum number of times a specialist may process requests from other specialists and the maximum number of times the specialist will perform its default action when control is passed to it.

Figure 6.14 shows a graph of the order that the coordinator passes control to the

Tolerance Step	0	2	4	6	8	10	12	14	16	18	20
Object Type	Number Identified										
bass_clef†	3	3	3	3	4	4	4	4	4	4	4
bass_clef_curl	3	3	3	3	4	4	4	4	4	4	4
composite‡	453	455	443	457	440	444	452	456	458	459	468
crotchet_rest	4	4	4	4	4	4	4	4	4	4	0
digit_4	24	24	24	24	24	24	24	24	24	24	24
digit_5	12	12	12	12	12	12	12	12	12	12	12
digit_6	12	12	12	12	12	12	12	12	12	12	12
dot	60	60	62	61	62	62	62	63	65	63	67
filled_note_stem_down†	36	36	36	36	36	36	37	37	37	37	37
filled_note_stem_up†	108	108	108	109	109	110	108	109	108	108	108
filled_notegr_stem_down†	15	15	16	15	16	16	15	15	16	16	16
filled_notegr_stem_up†	50	50	50	51	51	51	53	53	53	54	54
flat	31	31	26	30	26	26	28	27	27	26	27
full_notehead	242	244	247	245	246	248	249	249	250	251	267
hollow_note_stem_down†	5	5	5	5	5	5	5	5	5	5	5
hollow_note_stem_up†	3	3	4	4	4	4	4	4	4	4	4
hollow_notehead	9	9	11	10	11	11	11	11	11	11	11
hollow_notehead_dur_dots†	5	5	5	5	5	5	5	5	5	5	5
hough_horizontal	21	21	21	21	21	22	23	23	23	23	23
keysig†	8	7	6	7	7	7	8	8	8	8	8
natural	0	0	1	0	1	1	0	0	0	0	1
noise	1053	1054	1014	1058	1021	1021	1022	1027	1029	1031	1045
notegroup†	11	11	11	11	11	12	13	13	13	13	13
quaver_note_stem_up†	0	0	0	0	0	1	1	1	1	1	1
quaver_rest	1	1	1	1	1	1	1	1	1	1	1
quaver_tail_down	0	0	0	0	0	1	1	1	1	1	1
rect_rest	5	5	6	5	6	59	9	10	10	9	62
semi_quaver_rest	0	0	0	0	0	0	0	0	1	2	2
staff_line	40	40	40	40	40	40	40	40	40	40	40
staff_system	4	4	4	4	4	4	4	4	4	4	4
text	214	214	214	214	214	214	214	214	214	214	214
timesig†	24	24	24	24	24	24	24	24	24	24	24
treble_clef	4	4	4	4	4	4	4	4	4	4	4
unknown	393	394	327	386	318	267	328	333	334	335	271
vertical_line / barline	154	154	155	156	156	157	156	160	159	159	161
	16	16	16	16	16	16	16	16	16	16	16

† denotes objects created by assembly.

‡ “composite” objects are those objects left over when a partial match is made within it and the content becomes fragmented into new objects. The composite object itself no longer contains any graphical data.

Table 6.11: Effect of changing the coordinator’s “tolerance step” on *Promenade*

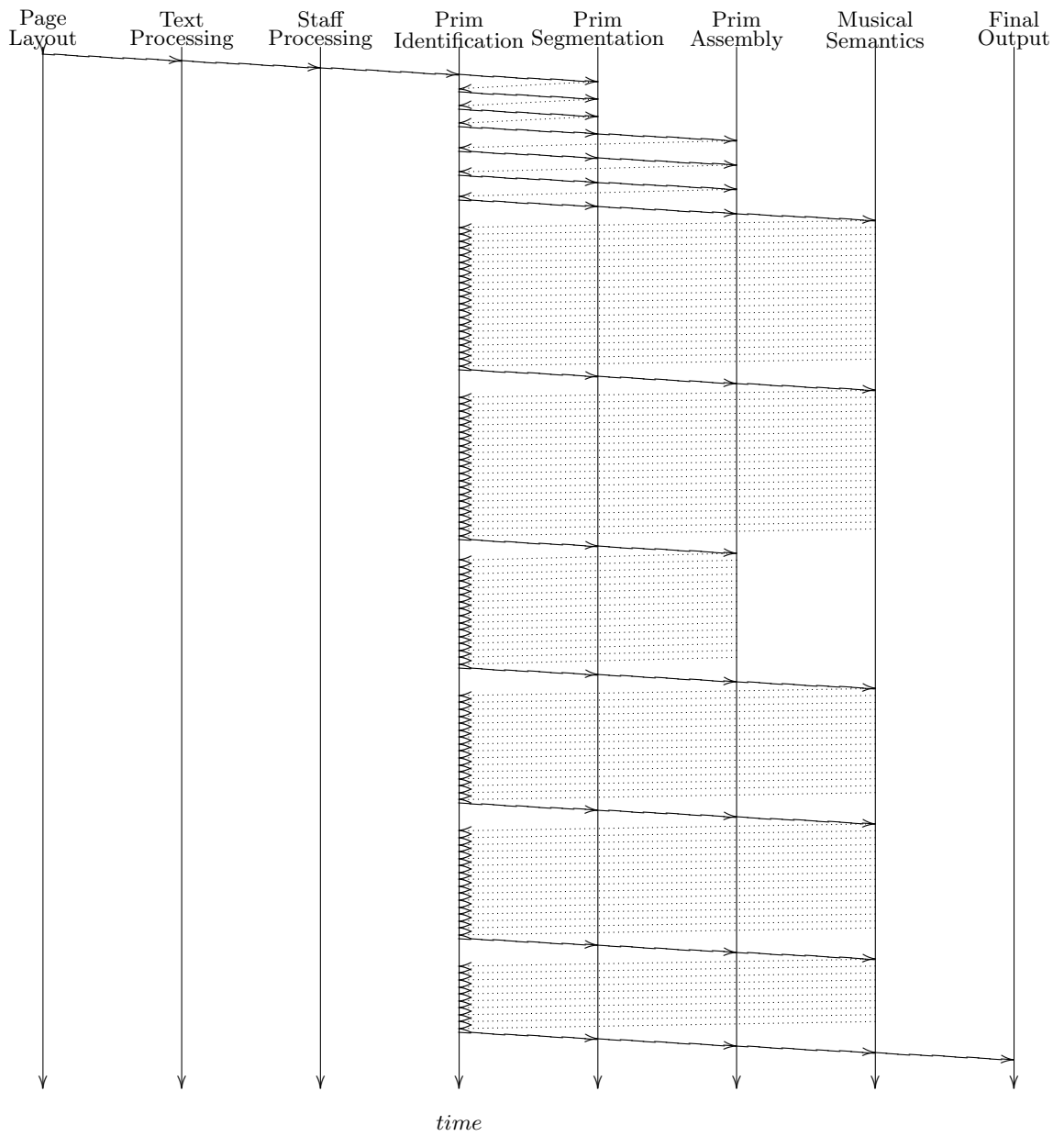


Figure 6.14: Graphical representation of program execution flow (*Promenade*, page 1)

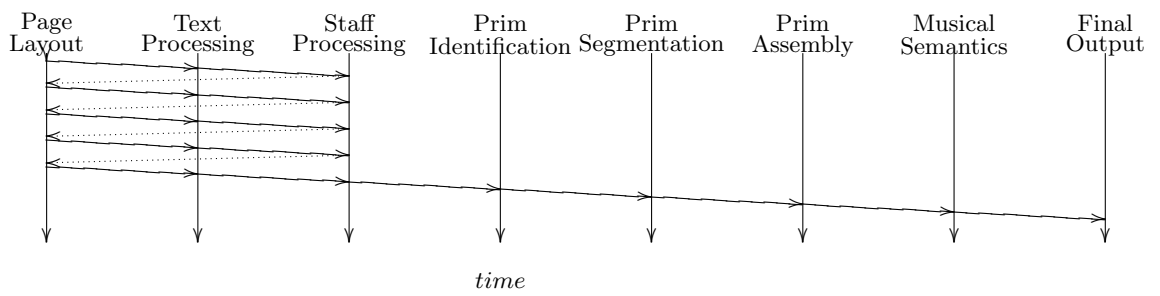


Figure 6.15: Graphical representation of program execution flow (for a blank image)

various specialists for the *Promenade* score. The solid arrows represent progression through the specialists’ natural order, while the dotted lines represent requests passing contextual information between specialists. This shows that generally, requests are performed in batches until either all the requests generated by a specialist’s action are completed or the specialist has reached its maximum generated requests count. For this particular processing run, every generated request was for providing contextual information about primitives to the Primitive Identification specialist. The first six requests, generated by the Primitive Segmentation and Primitive Assembly specialists, are generic requests to process all new objects created since the last time the specialist was run; the later requests are for processing specific objects or specific regions of interest.

When given a completely blank input image, the system behaves as shown in Figure 6.15. The Staff Processing specialist generates a request when it cannot detect any systems on the page. In contrast, the Text Processing specialist does not consider the lack of unknown objects for testing to be an error, and does not generate a request. This request is handled by the Page Layout specialist which performs various actions to try to correct any problems with the page—these actions include rotating the page to remove any skew, and joining objects together in case the image is poor quality and the systems are too fragmented to be detected. For a blank image, of course, none of these actions will make any difference, so there is a loop while the Staff Processing specialist keeps generating requests for “locate staves” and the Page Layout specialist keeps modifying the original image, until one of the limits are reached. In this particular processing run, the Page Layout specialist reached its default limit for number of processing calls (five) and the coordinating process rejected further requests that it would have handled.

Figure 6.16 shows a call trace graph for the OMR processing of *Scherzo 58* by Chopin, corresponding to the third summary displayed in Table 6.9. This input image is of a lower quality, and the graph shows requests made for functions such as “de-skew page” that are performed by the lower-level specialists. The de-skew function is based on the computationally expensive Hough Transform, and the Page Layout specialist exceeds its allowed CPU time after just one request, meaning any further requests to the Page Layout (for example, if the de-skew algorithm calculated the incorrect rotation angle) will be rejected.

These traces do not show any qualities about the requests, such as duration of

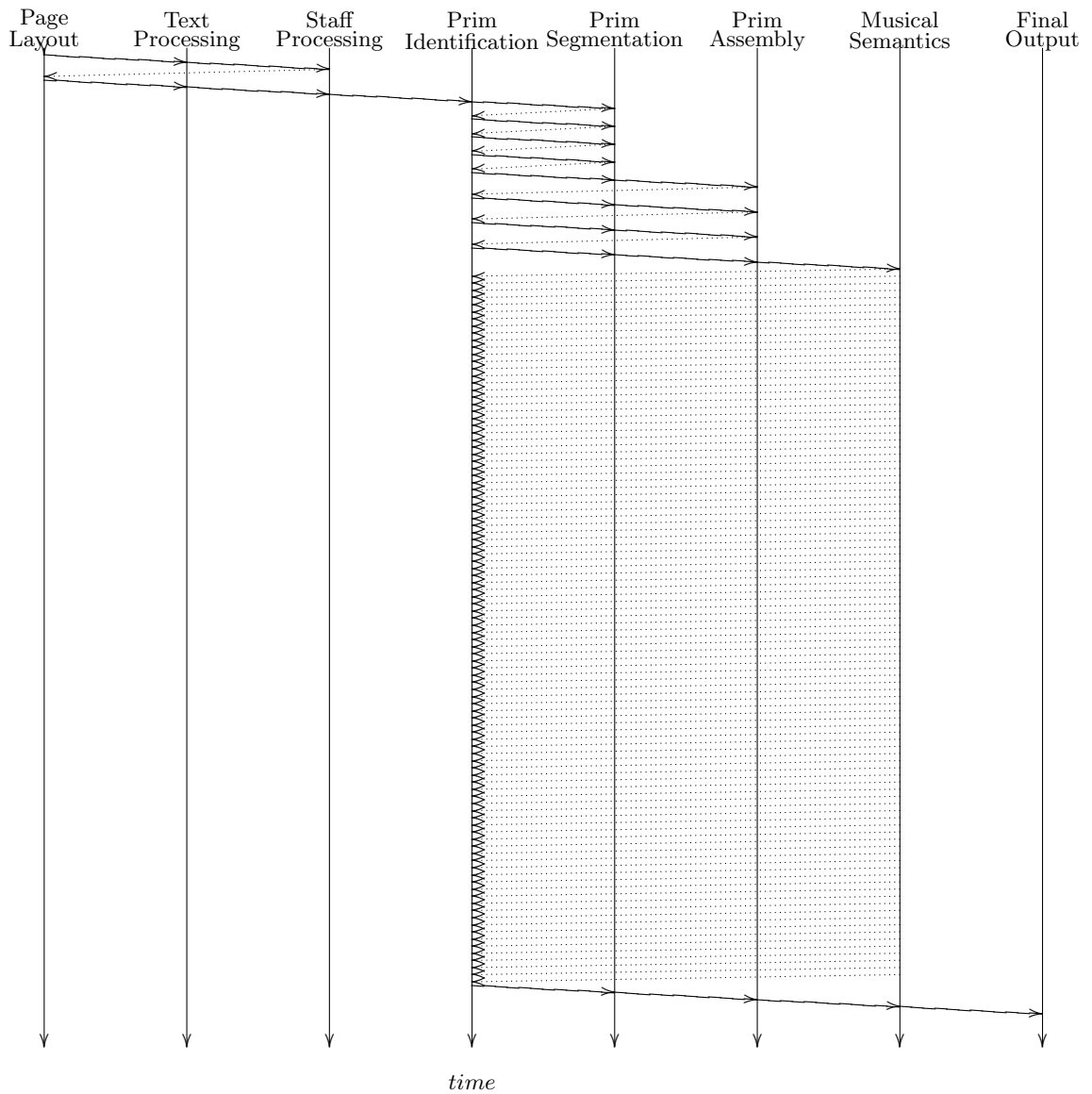


Figure 6.16: Graphical representation of program execution flow (Chopin)

execution or number of affected objects, other than the order that the specialists are called in. However, they demonstrate that the normal behaviour of the system is for the majority of requests to be from the higher level specialists to the Primitive Identification specialist. Requests to the lower level specialists are only made if objects or staff systems cannot be found, typically due to skew in the image.

Chapter 7

Conclusions

This research has explored methods for coordinating the various stages of the Optical Music Recognition process and demonstrated that contextual knowledge can be used to automatically correct mistakes and improve recognition accuracy. This chapter summarises the research performed, and outlines the main findings.

7.1 Original contributions of this thesis

An analysis of the literature has shown that while there have been proposals for—and several implementations of—the use of musical domain knowledge in some areas of OMR process, there has been little work using this knowledge throughout the entire system for decision-making and correcting mistakes.

In this thesis, methods for introducing contextual feedback into the OMR process were investigated, and several were implemented for further study. This raised some challenges, and offered insights that were further expanded for the implementation of a complete OMR system. The development of this system required the investigation and development of methods for the different knowledge sources (specialists) to pass messages containing information to each other. Each specialist needs different types of information, so the messaging sub-system needed to be flexible enough to represent diverse messages.

The usefulness of the feedback requests from the Primitive Assembly and Musical Semantics stages has been examined, as well as the effect that these have on the primitive identification rates. The previous chapters have discussed how the system's behaviour can be controlled by modifying the criteria used to decide when to stop processing feedback requests and by automatically modifying thresholds used when

performing tests on objects. This thesis has shown that using feedback in an OMR system to provide extra context for decision-making greatly increases the recognition rates.

7.1.1 Primitive Identification

Primitive Identification is the central component of the OMR process. As well as lowering thresholds for testing hypotheses about an object’s classification, this thesis has described several innovative aspects of musical primitive identification. The Primitive Identification specialist uses contextual information in different ways:

- preventing an object from being assigned particular classifications, or limiting matches to particular types, based on musical knowledge about surrounding objects;
- related to the previous point, re-testing and re-classifying objects’ assigned types based on conflicting information; and
- the ‘clustering’ algorithm for inferring the classification of similarly-shaped unknown objects.

Evaluation of the implemented OMR system showed that the Primitive Identification specialist is the main destination for contextual messages from other parts of the system. These requests contain information about the system’s current state, as determined by other specialists applying music-specific domain knowledge. Examples of this domain knowledge include information about how primitive shapes are joined into musical objects, and syntactic rules describing a particular music notation. These messages are used to provide extra information about possible classifications when testing and re-testing objects. In many cases, this process results in corrections and improvements to the set of classified objects, leading to a more accurate representation of the score.

7.1.2 Coordination of Knowledge Sources

The OMR process as implemented in systems such as CANTOR [5] and AOMR [36] is a linear step of distinct stages, with little scope for feedback between the various phases. While musical domain knowledge is often embedded in the recognition rules—such as looking for particular object types (such as clefs) in locations where they typically appear (the start of every staff normally has a clef)—and this improves

the recognition accuracy, coordinating the different phases in such a way as to allow feedback offers many advantages.

The research described in this thesis has led to the investigation and implementation of an OMR system designed around the idea of coordinating various knowledge sources. This system has some similarities in design and function to Kato and Inokuchi’s system, described in Chapter 2. Their blackboard approach uses layers of memory for storing hypotheses, with separate layers for various low-level (such as graphical information) and higher-level (such as musical note attributes) types of information, and their “top-down” methodology drives the recognition process on a bar-by-bar basis. In contrast, this thesis describes a system with a less structured separation between different parts of the global memory, meaning any specialist can access and modify any object. This uses a bottom-up approach, with the focus on the data as it moves through the various phases, although top-down methods based on assembly and syntax rules are also used for error correction and object prediction.

7.1.3 Identification based on Assembly Knowledge

The assembly rules used for joining graphical primitive shapes into larger musical objects are also used for suggesting classifications for unknown objects. Any unknown object that would satisfy a rule’s conditions if it were classified as the appropriate object is then flagged for further examination. For example, a rather crucial assembly rule is the one for joining noteheads and vertical lines into note objects (with a set of conditions describing the relative positions and distances between the primitive objects). In this example, an unknown object that would satisfy the rule if it were a notehead is marked as a possible notehead. Similarly, it might be in the correct location relative to a notehead to satisfy the rule if it were a vertical line, in which case it would be marked as a possible vertical line. This extra information is then taken into account—for example, by lowering pattern-matching thresholds for those possible classifications—at some later stage when the Primitive Identification module next processes this object.

7.1.4 Identification based on Clustering

The experimental OMR system has demonstrated that objects can be successfully classified solely on the basis of syntactic feedback, even for object types that are not previously known to the object identification routines. This is currently based on

syntactic knowledge of the interaction allowed between object types, and the system allows syntax rules for arbitrary types, not just for types known to the low-level graphical routines in earlier parts of the system.

The clustering algorithm was described in Chapter 5 (Section 5.4.3). It uses semantic information for object classification suggestions, and uses Compression-Based Template Matching (CBTM) for identifying similar shapes. While it has been shown that this can successfully identify some types of objects, this process needs more investigation to improve its accuracy and robustness.

7.1.5 Run-time Pattern Adaption

Because of the vast variation in symbol shapes used by different publishers, an OMR system should have a flexible recognition process that can adapt for each input score. The OMR system described here adapts its behaviour during object recognition/classification in several different ways.

Adaption of Pattern Conditions

Objects are compared against the pre-defined pattern templates with increasing tolerance as extra context is calculated for the possible classifications of remaining unknown objects. This means that primitive shapes that closely match one of the pattern templates are identified first, and the remaining unknown objects are later tested against the patterns with the template rules more relaxed.

For instance, a pre-defined pattern description might specify a minimum and maximum size, and specify a bitmap shape along with a minimum percentage of pixels that must match between an unknown object and this shape to be accepted. (Example pattern descriptions are provided in Appendix A.2.) The rules can be relaxed—for example, increasing and decreasing the maximum and minimum allowed dimensions, respectively—if the semantic processing has identified possible classifications for an unknown object, or if the system is trying to reduce the number of remaining unknown objects with the possible trade-off of having some objects incorrectly classified.

Adding New Templates

As well as the set of patterns initially loaded into the system when it starts up, unknown objects are also compared against recognised objects. This means that

the set of templates used for recognition adapts—at run-time—to better suit each individual input image, rather than (or as well as) static primitive descriptions that try to ‘match all’ objects of a particular type. This still requires a set of initial pattern descriptions so that examples of each type are found in the input score. Because of the variation in primitive shapes and symbols used by different score typesetters and publishers, it may be difficult to have a starting set of descriptions that works for all scores using the same notation.

The identification of previously unknown object types via clustering (mentioned above) is also helpful for introducing example templates for object types that do not have starting pattern descriptions, or are not similar enough to any existing pattern descriptions.

7.1.6 Re-classification of Objects

Few OMR systems will change the classification of an object, once assigned; Kato and Inokuchi’s system is a significant exception to this observation. For example, in their system, primitives that fail the consistency rules (such as a rule stating that a quaver tail must be attached to exactly one stem) are put back into the image. Similarly, semantic information can reject hypotheses (based on the durations of objects in the bar) for the objects in the symbol layer. The OMR system described in this thesis performs similar actions. Impossible or unlikely combinations (as allowed by the assembly and syntax/semantic rules) may result in an object being re-assigned to an alternative type or even set back to unclassified (“unknown”).

7.1.7 System Design

The system implemented as part of the research described here has shown the viability of a modular, message-passing design for the purpose of OMR. The modular design means that each specialist function of the system can be modified or even completely replaced by another module that uses different algorithms, without affecting the rest of the system. The specialists need only agree on the name and format of the messages that are passed between them for requests. An arbitrary number of specialists can be added to the system, so this could lead to the development of specialist modules that have very specific purposes, such as heuristically correcting a particular type of error that only occurs for a certain set of input scores. This model leads to reuse of code and can help shorten development times when developing and

testing new algorithms.

7.2 Shortcomings and Challenges

As the whole, the system behaves in a conservative manner, preferring to leave something unrecognised and assume a specialist using contextual feedback will find the correct classification later, rather than taking a “best guess” at a classification and assuming that any incorrect classifications will be discovered and fixed at a later stage. Similarly, this means that the system is better at using feedback to classify remaining unknown objects rather than using the feedback to re-evaluate incorrectly classified objects. This is partly because of the nature of the different errors; it is easy to pinpoint where an unrecognised object is, but difficult to pinpoint the exact location of a recognition error (for example) that causes a bar to have an incorrect number of beats.

7.2.1 Request Handling

The current model does not really allow forward-flow of information, or “what-if” scenarios. For example: “Would changing this classification improve the semantics accuracy?” In the current model, this can only be done by actually making the change, and hoping that the change will be reverted at a later stage if it is not the best decision.

Request Priority

There is difficulty in deciding which requests are more important than others. As implemented, the priority levels attached to requests for various problems and notices were chosen heuristically.

Better methods for determining importance and priority may be uncovered; however, recall that evaluation of the system demonstrated that with the currently assigned request priorities there was no difference in the final object classifications when requests were chosen for servicing randomly rather than in order of priority. It is possible that the order that requests are served in only matters if not all the requests will be eventually serviced, due to resource limitations.

7.2.2 Primitive Identification

There are several limitations of the primitive identification/classification routines as implemented in the OMR system. This is partly because the system originally implemented a naïve classification scheme, based on a simple “XOR” bitmap comparison, with the system’s focus on using knowledge from different parts of the recognition process. While this has been useful for identifying sources of musical knowledge and investigating methods for using this information for contextual decision-making, a more practical OMR system should expend more effort in the initial recognition stages to try to minimise classification errors.

Tolerance Threshold

There needs to be a balance in how tolerant to be when looking for matches. Being too tolerant can result in too many false matches. Conversely, being more strict while matching objects to pattern descriptions will result in fewer legitimate objects being correctly classified, depending on how much they vary from the described idealised object.

The approach taken by this OMR system is to start off with a conservative threshold, and become increasingly more tolerant with further processing when comparing unknown objects to the templates, especially when given extra context about possible classifications (for example, based on syntactic information). However, this will eventually become too tolerant and start incorrectly matching objects — for example, large objects of types that do not have pattern descriptions remain unrecognised until the other parts of the system offer enough suggestions for objects that would fit in that position, resulting in erroneous primitives being extracted from the large object. One possibility is to make the tolerance partially dependent on factors such as the size of the unknown object. Another strategy is to be less tolerant for patterns that match and extract primitives from larger objects, and only be more tolerant for objects that are ‘isolated’ and identified in their entirety.

Classification Techniques

Due to the developed system’s focus on the interactions between the specialists, rather than focusing on the best possible recognition by the Primitive Identification specialist, there is an over-reliance on bitmap matching in the primitive pattern description rules. Using more sophisticated features that can be calculated from

the primitive bitmaps will improve the flexibility and accuracy of the pattern descriptions. Fujinaga's GAMERA system [36] used over a dozen different features, and required the use of a genetic algorithm to calculate which features should be measured for different primitive types, based on training examples.

The image processing field offers many different types of qualities that can be calculated about a bitmap image. Further work could be done to see if any are particularly suited for music recognition (although this will be dependent on the music notation used in the input scores).

7.2.3 System Evaluation

The Primitive Identification stage uses the majority of the processing time, mostly due to the reliance on template bitmap matching, which is processor intensive. The system uses a modest amount of memory, so the development and use of techniques that use less processing time—possibly with the trade-off of using more memory—should be investigated.

Evaluating the performance of the system can be difficult. Messages involve two specialists, and the resolution of a request depends on the behaviour of the receiving specialist, so it is hard to give credit to an individual specialist for noticing an error and suggesting possible resolutions. While human evaluation has been proposed and used for comparing the similarity of two scores, automatically comparing and quantifying the differences between two semantic representations of the same score is a largely unsolved problem.

7.3 Future work

As always, ever-increasing computer power in the future will allow an OMR system to perform more tests, using more complex algorithms. Introducing more tests will provide more contextual information, and this extra information may require more complex procedures for decision-making.

The modular design means that the various specialists could be replaced or supplemented by new modules with similar functionality without requiring modification to the remainder of the system. For example, the low-level pattern recognition routines for locating graphical primitives could be replaced with a more sophisticated sub-system, such as that described by Fujinaga [36]. Another example is the possibility of adding specialists for very specific behaviour, such as correcting a problem

with the recognition that only occurs on input images with some particular characteristic (such as one composer or publisher who uses an innovative layout or graphical symbol).

Another possibility opened up by the modular design is to have multiple specialists that perform identical or overlapping functions, although using different algorithms. For example, there may be multiple specialists that can perform primitive identification tasks, or assembly tasks. This would require a coordinating process that could take advantage of the extra capabilities, such as using each specialist in some order for matters left unresolved or unrecognised by the previous specialist, or this could be more advanced and get the opinion of all the specialists for that task and somehow have methods for weighing up and resolving conflicting suggestions.

Automatic recognition of musical scores is a long way from approaching the accuracy of a musician reading a score—human perception is much more powerful than today’s computer algorithms. More methods of user input into an OMR system may lead to improved performance. Machine Learning techniques could allow a user to correct some mistakes with the system keeping track of the corrections and using that information during decision-making at a later stage (either within the same processing batch, or for completely different scores). Audio playback as well as graphical manipulation of a score would help the user locate some recognition errors.

A more robust model for calculating tolerances, thresholds, and object certainty values is needed. The use of genetic algorithms and other numerical techniques would require more accurate numbers, as these numbers are used by a fitness function to evaluate whether or not one set of possible classifications is better than another. While the system described here also uses the classification certainties to some extent, such as when re-classifying an object that is syntactically incorrect, they are not as critical to the decision-making as they are when used in these other algorithms.

7.3.1 Priority of Requests

As discussed in the previous section, evaluation has shown that in the current implementation, prioritising requests had little effect on the system’s performance or recognition accuracy. Further effort could be spent investigating either a better evaluation of how urgent each request is in terms of the seriousness of the observed

problem, or other methods for calculating priority of requests, such as assigning an expected cost of carrying out the request for example, in terms of processing time or other system resources.

7.3.2 OCR

Semantic meaning of text on scores is very important for document metadata, performance notes, lyrics, and multi-instrument scores. Document metadata (such as title, composer, date, publisher, and so on) is necessary for searching and retrieving digital scores from a document repository. Recognising this text automatically requires Optical Character Recognition (OCR) to convert the bitmaps of text into computer-understandable text codes.

Although there are many proprietary OCR systems, there remains a lack of high-quality open source OCR engines available that could be tailored for use or inclusion in an OMR system. Further work on this could involve improving or modifying one of the existing open source OCR programs for this purpose.

7.3.3 Musical Semantics

Work in the field of music information retrieval has demonstrated that genre can often be determined from an audio recording or (less accurately) from symbolic data. Calculating the genre of a score (whether from the symbolic musical data, or based on other features of the score such as title, instruments and composer/artist metadata) may mean that genre-specific knowledge can be exploited for that image.

Higher-level examination of the musical semantics may be useful for correcting some recognition and attribute errors. For example, common motifs and themes or durational patterns may be found and used to find similar passages that have a few differences; these differences could be inspected to carefully check for recognition, assembly, or pitch recognition errors. The effectiveness of this technique might be determined by the genre and style of the music in the score.

7.4 Key findings

The broad aim of this research was to determine how the data and methods used in an OMR system could be better coordinated to improve the system's accuracy. More specifically, does a coordinated approach offer advantages over currently used methods? How can extra information about a score be used to modify the system's

behaviour to increase performance? Which type of errors can be automatically detected, and how can information about them be used to correct mistakes?

7.4.1 Knowledge Coordination

The research described here has clearly shown that using a coordinated approach for representing and using knowledge can automatically correct many errors and improve the recognition process for OMR. This was achieved by designing and implementing a system that used a message-passing framework that allowed contextual information about musical objects to be passed between the specialist knowledge sources. These messages are in a structured form that allows specialists to use arbitrary request types. Requests will only be serviced if another specialist registers that type as one that it can process, and the coordinating process decides that the request message should be allowed.

As well as the messages, information is also attached to the objects, and the specialists can directly access and modify the image information, the musical object information, and the syntactic structure that describes the inter-relationships of the objects. Although all the information is accessible by any specialist, in practice each specialist only modifies information at the one or two levels involved in its data manipulation.

7.4.2 Adaptation

An OMR system can give more accurate results by adapting, at run-time, to the particular score it is processing. The approach found most flexible during this research is to use pattern templates (along with a coordinated approach providing contextual feedback) to find initial objects, and then use those found objects to help identify remaining unknown objects. For example, this means that the pattern descriptions only need to match one of the flats in the score and then that object can be used to match all the similarly-shaped flats, rather than the patterns being broad enough to match many slight (and not-so-slight) variations in the shape of glyphs used by different composers and score publishers.

7.4.3 Algorithm Settings and Thresholds

Decision-making throughout the OMR process involves performing many tests. For example, there may be multiple algorithms available for a particular task and the

specialist chooses one based on some criteria. The Primitive Identification specialist calculates how well a graphical shape compares to a primitive pattern description and whether or not it matches close enough to set a classification. The Assembly and Semantics specialists have thresholds for determining relationships between objects based on relative positions and proximity. Almost all these decisions involve using parameters to decide if something reaches a particular threshold or not. These settings should not be hard wired into the process, but they should be modifiable—either tunable by the person running the system, or dynamically modified by the system itself during processing.

This research has shown that an effective strategy for music recognition is to start the processing with conservative values for these variable settings and become more tolerant as more contextual information is provided. This means, for example, that atypical objects that are not correctly accounted for because they appear different to other objects of the same type are eventually classified—whether it is with the use of extra contextual information, or adaptation of the patterns to match the unknown object to one of the already classified objects.

7.5 In closing

The research presented in this thesis offers compelling evidence that many mistakes made during the OMR process can be automatically corrected by making more use of contextual knowledge during decision-making. The thesis has described and demonstrated methods that achieve improved accuracy: in the individual recognition methods that use the extra context, in the system that coordinates these individual methods, and in the representation of musical knowledge in a format useful to those methods.

Following this line of research will lead to better OMR systems that can infer much more information about the music being recognised. Adding more methods for calculating extra contextual knowledge about the musical data extracted from the score will mean improved understanding of the input document and better recognition accuracy. Advances in OMR will lead to better systems that are of practical use to people: publishers and composers wanting to edit or tidy up a composition, and performers who want to listen to or play along to scores that they have available as sheet music.

Bibliography

- [1] *Proceedings of the Third International Conference on Document Analysis and Recognition*, ICDAR '95, Montréal, Canada, August 1995. IEEE. [2.3.1](#), [3.4.2](#), [8](#), [26](#)
- [2] *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, ICDAR '99, Bangalore, India, 1999. IEEE. [2.3.1](#), [3.2.1](#), [3.4.2](#), [48](#), [72](#)
- [3] Jarmo T. Alander. Indexed bibliography of genetic algorithms in optics and image processing. Report 94-1-OPTICS, University of Vaasa, Department of Information Technology and Production Economics, 1995. [4.3.5](#)
- [4] A. Amin, S. Fischer, A. F. Parkinson, and R. Shiu. Comparative study of skew detection algorithms. *Journal of Electronic Imaging*, 5:443–451, oct 1996. [3.5.3](#)
- [5] David Bainbridge. *Extensible Optical Music Recognition*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1997. [2.1.5](#), [3.5.3](#), [4.3.4](#), [5.2.1](#), [5.6](#), [6.1.2](#), [7.1.2](#)
- [6] David Bainbridge and Stuart Inglis. Musical image compression. In *Proceedings of the IEEE Data Compression Conference*, pages 209–218, Snowbird, Utah, 1998. IEEE. [1.1](#)
- [7] Henry S. Baird, Horst Bunke, and Kazuhiko Yamamoto, editors. *Structured Document Image Analysis*. Springer-Verlag, 1992. [1.2](#), [15](#), [44](#)
- [8] S. Baumann. A simplified attributed graph grammar for high-level music recognition. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [[1](#)], pages 1080–1083. [4.3.4](#), [5.2.3](#)
- [9] P. Bellini and P. Nesi. Automatic justification and line-breaking of music sheets. *International Journal of Human-Computer Studies*, 61:104–137, 2004. [3.1.3](#)

- [10] P. Bellini, P. Nesi, and M. B. Spinu. Cooperative visual manipulation of music notation. *ACM Transactions on Computer-Human Interaction*, 9(3):194–237, sep 2002. [2.1.9](#)
- [11] Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. *Off-Line Optical Music Sheet Recognition*, chapter 2. IRM Press, July 2004. [2.1.9](#)
- [12] Juan Pablo Bello and Mark Sandler. Blackboard system and top-down processing for the transcription of simple polyphonic music. In *Proceedings of the COST G-6 Conference on Digital Audio Effects*, Verona, Italy, December 2000. [4.3.3](#), [4.4](#)
- [13] Adrian C. Bickerstaffe and Enes Makalic. MML classification of music genres. In *16th Australian Conference on AI*, volume 2903 of *Lecture Notes in Computer Science*, pages 1063–1071, Perth, Australia, 2003. Springer-Verlag. [5.2.4](#)
- [14] D. Blostein, E. Lank, and R. Zanibbi. Treatment of diagrams in document image analysis. In M. Anderson, P. Cheng, and V. Haarslev, editors, *Theory and Applications of Diagrams*, volume 1889 of *Lecture Notes in Computer Science*, pages 330–344. Springer Verlag, 2000. [4.3](#), [4.3.3](#)
- [15] Dorothea Blostein and Henry S. Baird. A critical survey of music image analysis. In Baird et al. [\[7\]](#), pages 405–434. [2.1](#)
- [16] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Practical use of graph rewriting. Technical Report 95-373, Department of Computing and Information Science, Queen’s University, Ontario, Canada, January 1995. [4.3.4](#)
- [17] Dorothea Blostein and Lippold Haken. Justification of printed music. *Communications of the ACM*, 34(3):88–99, March 1991. [3.1.3](#)
- [18] Ivan Bruno, P. Bellini, and P. Nesi. Assessing optical music recognition tools. Technical report, The Interactive-Music Network, 2nd MusicNetwork Open Workshop, July 2003. [2.1.10](#), [6.1.2](#)
- [19] H. Bunke and P. S. P. Wang, editors. *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997. [3.4.2](#), [3.5.2](#), [46](#), [68](#)
- [20] Donald Byrd. Music notation software and intelligence. *Computer Music Journal*, 18(1):17–20, 1994. [2.2.1](#), [5.2.2](#), [6.2.7](#)

- [21] Donald A. Byrd. *Music Notation by Computer*. PhD thesis, Indiana University, 1984. [2.2.1](#), [5.2.2](#)
- [22] Nicholas P. Carter. *Automatic Recognition of Printed Music in the Context of Electronic Publishing*. PhD thesis, University of Surrey, Surrey, United Kingdom, 1989. [2.1.1](#), [3.3](#)
- [23] Nicholas P. Carter. Segmentation and preliminary recognition of madrigals notated in white mensural notation. In O’Gorman and Kasturi [61]. [5.5.2](#)
- [24] Gerd Castan. Music notation formats. Internet Web Site, 2004. <http://www.music-notation.info/>. [2.2.3](#)
- [25] Niann-Tsuu Chiang. Optical music recognition: Processing the sacred harp. Master’s thesis, School of Computing and Mathematical Sciences, University of Waikato, New Zealand, July 1998. [2.1.5](#)
- [26] Bertrand Couïasnon and Jean Camillerapp. A way to separate knowledge from program in structured document analysis: Application to optical music recognition. In *Proceedings of the Third International Conference on Document Analysis and Recognition* [1]. [2.1.3](#)
- [27] Bertrand Couïasnon and Bernard Rétif. Using a grammar for a reliable full score recognition system. In *Proceedings of the International Computer Music Conference*, Banff, Canada, September 1995. [2.1.3](#)
- [28] Michael Droettboom. Selected research in computer music. Master’s thesis, The Peabody Institute of the John Hopkins University, Baltimore, Maryland, USA, April 2002. [2.1.6](#), [4.1.2](#), [5.4.5](#)
- [29] Michael Droettboom and Ichiro Fujinaga. Symbol-level groundtruthing environment for OMR. In *Fifth International Conference on Music Information Retrieval* [75], pages 497–500. [6.1](#)
- [30] Michael Droettboom, Karl MacMillan, and Ichiro Fujinaga. The gamera framework for building custom recognition systems. In *Symposium on Document Image Understanding Technologies*, pages 275–286, Greenbelt, Maryland, USA, April 2003. [2.1.6](#)

- [31] Jon W. Dunn and Constance A. Mayer. Variations: A digital music library system at Indiana University. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, Berkeley, California, 1999. ACM. [1.1](#)
- [32] Hoda Fahmy and Dorothea Blostein. A graph grammar for recognition of music notation. *Machine Vision and Applications*, 6(2):83–99, 1993. [4.3.4](#)
- [33] M. Ferrand, J. A. Leite, and A. Cardoso. Hypothetical reasoning: An application to Optical Music Recognition. In M. C. Meo and M. V. Ferro, editors, *APPIA-GULP-PRODE'99 Joint Conference on Declarative Programming*, pages 367–381, L'Aquila, Italy, September 1999. [2.1.7](#)
- [34] M. Ferrand, J. A. Leite, and A. Cardoso. Improving optical music recognition by means of abductive constraint logic programming. In P. Barahona and J. J. Alferes, editors, *Progress in Artificial Intelligence, 9th Portuguese International Conference on Artificial Intelligence*, volume 9, pages 342–356. Springer-Verlag, September 1999. [2.1.7](#), [2.1](#)
- [35] Miguel Ferrand. *Reasoning Under Uncertainty: Applications to Automatic Recognition and Interpretation of Printed Music*. M.Phil, Dept. Engenharia Informatica, Universidade de Coimbra, September 2000. [2.1.7](#)
- [36] Ichiro Fujinaga. *Adaptive Optical Music Recognition*. PhD thesis, McGill University, Canada, 1997. [2.1.6](#), [3.2.2](#), [5.5](#), [5.5.2](#), [7.1.2](#), [7.2.2](#), [7.3](#)
- [37] Christopher Graefe, Derek Wahila, Justin Maguire, and Orya Dasna. Designing the muse: A digital music stand for the symphony musician. In *Proceedings of the CHI '96 Conference on Human factors in computing systems*, page 436, Vancouver, Canada, 1996. ACM. [1.1](#)
- [38] Cindy Grande and Alan Belkin. The development of the notation interchange file format. *Computer Music Journal*, 20(4):33–43, 1997. [2.2.3](#)
- [39] Karl Hader. *Aus der Werkstatt eines Notenstechers*. Waldheim–Eberle Verlag, Vienna, Austria, 1948. [2.4](#)
- [40] Simon Haykin, editor. *Kalman Filtering and Neural Networks*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley & Sons, New York, USA, 2001. [2.1.3](#)

- [41] Holger H. Hoos and Keith A. Hamel. The guido music notation format version 1.0 specification part 1: Basic guido. Technical Report TI 20/97, Fachbereich Informatik, Technische Universität Darmstadt, 1997. [2.2.3](#)
- [42] IEEE. *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, ICDAR '97, Ulm, Germany, 1997. IEEE. [2.3.1](#), [3.4.2](#), [73](#)
- [43] Stuart Inglis and Ian H. Witten. Compression-based template matching. In J. A. Storer and M. Cohn, editors, *1994 Data Compression Conference*, pages 106–115, Utah, USA, 1994. IEEE, IEEE Press, Los Alamitos, CA, USA. [5.4.2](#), [5.4.2](#)
- [44] Hirokazu Kato and Seiji Inokuchi. A recognition system for printed piano music using musical knowledge and constraints. In Baird et al. [\[7\]](#), pages 435–455. [2.1.2](#)
- [45] Dennis Koelma and Arnold Smeulders. A blackboard infrastructure for object-based image interpretation. In E. Backer, editor, *Computing Science in The Netherlands*, pages 136–147, Amsterdam, 1994. CWI. [4.3.3](#)
- [46] A. Kundu. Handwritten word recognition using hidden markov model. In Bunke and Wang [\[19\]](#), pages 157–182. [2.3.1](#)
- [47] J. A. Leite, M. Ferrand, and A. Cardoso. Riem - a system for recognition and interpretation of music writing. Technical Report RI-DEI-001-98, Dept. Engenharia Informática, Universidade de Coimbra, 1998. [2.1.7](#)
- [48] S. Marinai and P. Nesi. Projection based segmentation of musical sheets. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition* [\[2\]](#), pages 515–518. [2.1.9](#), [3.2.1](#)
- [49] Cory McKay and Ichiro Fujinaga. Automatic genre classification using large high-level musical feature sets. In *Fifth International Conference on Music Information Retrieval* [\[75\]](#), pages 525–530. [4.1.2](#), [4.2](#), [5.2.4](#)
- [50] Rodger J. McNab, Lloyd A. Smith, David Bainbridge, and Ian H. Witten. The New Zealand Digital Library MELody inDEX. *D-Lib Magazine*, May 1997. [1.1](#)
- [51] J. R. McPherson. Page turning — score automation for musicians. B.Sc Honours thesis, University of Canterbury, New Zealand, 1999. [1.1](#)

- [52] K. Ng, R. Boyle, and D. Cooper. Domain knowledge enhancement of optical music score recognition. Technical Report ESRRS-1997-01, University of Leeds, 1997. [2.1.4](#), [4.1.2](#), [4.2](#)
- [53] K. C. Ng and R. D. Boyle. Recognition and reconstruction of primitives in music scores. *Image and Vision Computing*, 14(1):39–46, February 1996. [2.1.4](#)
- [54] K. C. Ng and D. Cooper. Enhancement of optical music recognition using metric analysis. In *Proceedings of Colloquium on Musical Informatics*, volume XIII, pages 189–192, L’Aquila, Italy, September 2000. [2.1.4](#), [4.1.2](#)
- [55] Kia Ng. Music manuscript tracing. In D. Blostein and Y.-B. Kwon, editors, *Graphics Recognition. Algorithms and Applications*, volume 2390 of *Lecture Notes in Computer Science*, pages 330–342. Springer-Verlag, 2002. [2.1.4](#), [3.2.2](#), [4.2](#)
- [56] Kia Ng, Jerome Barthelemy, Bee Ong, Ivan Bruno, and Paolo Nesi. Coding images of music sheets. Technical Report DE4.7.1, The Interactive-Music Network, February 2004. Project IST-2001-37168. [2.1.10](#), [2.1.12](#), [6.1.2](#)
- [57] Kia Ng, Roger Boyle, and David Cooper. Key signature detection using note distribution. Technical Report 95.23, School of Computer Studies, University of Leeds, UK, 1995. [2.1.4](#)
- [58] Kia C. Ng. *Automated Computer Recognition of Music Score*. PhD thesis, School of Computer Studies, University of Leeds, 1995. [2.1.4](#)
- [59] Han-Wen Nienhuys and Jan Nieuwenhuizen. Lilypond, a system for automated music engraving. In *Processings of the XIV Colloquium on Musical Informatics*, pages 167–172, Firenze, Italy, May 2003. [2.6](#), [3.1.3](#)
- [60] Han-Wen Nienhuys, Jan Nieuwenhuizen, et al. *GNU Lilypond: The music typesetter*, 2.4 edition, 2004. <http://www.lilypond.org>. [1.2.1](#), [2.2.1](#), [3.1.3](#)
- [61] Lawrence O’Gorman and Rangachar Kasturi, editors. *Document Image Analysis*. IEEE Press, 1995. [1.2](#), [2.3.1](#), [3.5.2](#), [23](#)
- [62] Theodosios Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1981. [3.5.3](#), [5.5](#)

- [63] Jeremy Pickens, Juan Pablo Bello, Giuliano Monti, et al. Polyphonic score retrieval using polyphonic audio queries: A harmonic modeling approach. *Journal of New Music Research*, 32(2):223–236, June 2003. [4.3.3](#)
- [64] João Caldas Pinto, Pedro Vieira, and João M. Sousa. A new graph-like classification method applied to ancient handwritten musical symbols. *International Journal on Document Analysis and Recognition*, 6(1):10–22, July 2003. [2.1.11](#)
- [65] Gary M. Rader. Creating printed music automatically. *IEEE Computer*, 29(6):61–68, June 1996. [3.1.3](#)
- [66] John C. Russ. *The Image Processing Handbook (3rd edition)*. CRC Press, 1999. [3.5.2](#)
- [67] Eleanor Selfridge-Field, editor. *Beyond MIDI: The handbook of musical codes*. MIT Press, 1997. [2.2.3](#)
- [68] M. Shridhar and F. Kimura. Segmentation-based cursive handwriting recognition. In Bunke and Wang [19], pages 123–156. [2.3.1](#)
- [69] Lloyd Smith and Richard Medina. Discovering themes by exact pattern matching. In *Proceedings of the Second Annual International Symposium on Music Information Retrieval*, pages 31–32, Bloomington, Indiana, USA, October 2001. [5.2.4](#)
- [70] Kurt Stone. *Music notation in the twentieth century: A practical guidebook*. W.W. Norton, New York, 1980. [1.1](#), [2.2.1](#)
- [71] Marc Vuilleumier Stückelberg and David Doermann. Model-based graphics recognition. In Chhabra and Dori, editors, *Graphics Recognition (GREC '99) Third International Workshop: Selected Papers*, volume 1941 of *Lecture Notes in Computer Science*, pages 121–132. Springer-Verlag, 1999. [2.1.8](#)
- [72] Marc Vuilleumier Stückelberg and David Doermann. On musical score recognition using probabilistic reasoning. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition* [2]. [2.1.8](#), [2.1.8](#)
- [73] Marc Vuilleumier Stückelberg, Christian Pellegrini, and Mélanie Hilario. An architecture for musical score recognition using high-level domain knowledge. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* [42]. [2.1.8](#), [2.2](#), [4.3.2](#), [4.3](#)

- [74] George Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signals*. PhD thesis, Princeton University, June 2002. [5.2.4](#)
- [75] Universitat Pompeu Fabra. *Fifth International Conference on Music Information Retrieval*, Barcelona, Spain, October 2004. [29](#), [49](#)
- [76] Kadir Wijaya. Extending cantor: An optical music recognition system. Master's thesis, School of Computing and Mathematical Sciences, University of Waikato, New Zealand, February 1999. [2.1.5](#), [5.4.1](#)
- [77] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition, June 2005. [3.2.2](#)
- [78] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 1999. [6.2.3](#)
- [79] Sue Wu and Adnan Amin. Automatic thresholding of gray-level using multi-stage approach. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, Edinburgh, Scotland, August 2003. IEEE. [3.5.1](#)
- [80] I. Yoda, K. Yamamoto, and H. Yamada. Automatic construction of recognition procedures for musical notes by genetic algorithm. In *Proceedings of IAPR Workshop on Document Analysis Systems*, pages 203–209, Kaiserslautern, Germany, October 1994. [4.3.5](#)

Appendix A

Practical Implementation

A.1 Resource Limits for Specialists

Configuration file controlling per-specialist and global coordinator settings:

```
[global]
##### coordinator settings
# mostly for experimentation purposes - we can load .mus files
# and skip some of the initial steps
# starting_specialist = 4

# how much should specialists change by when told to "try harder"?
default_tolerance_step = 5

max_total_reqs_processed = 200 # coordinator variable

##### default limits for specialists (can be overridden below)

# if this isn't specified, the global default is 5
#max_reqs_submitted = 3

# if this isn't specified, the global default is 5
#max_reqs_received = 12

# default global setting per specialist is 60 seconds of cpu time
max_cpu = 90 # in seconds
```

[Staff Processing]

[Prim Identification]

this specialist is the target of most requests...

max_reqs_received = 200

max_cpu = 300 # in seconds

[Prim Segmentation]

this limits the number of times prim seg. will do its default action.

max_reqs_created = 4

max_reqs_submitted = 4

[Musical Semantics]

max_reqs_submitted = 100

[Final Output]

A.2 Object Pattern Descriptions

This appendix gives a sample of some of the pattern descriptions used by the Primitive Identification specialist. The patterns that contain templates have the bitmaps shown in Figure A.1.

```
1  # ORDER IS IMPORTANT! Objects will be tested in the order that appears
   # in these files!

   primitive
5  name          ‘‘quaver_rest’’
   # this primitive description is relative to a staff of the following
   # height, and will be scaled during matching
   staff_height  84
   # is this object type expected to be part of a larger object and
10  # needs to be extracted, or are all these objects "stand-alone"?
   isolated_only yes
   pattern { bbox (20, 38), (25, 42) }
   # if a pattern is given an origin point, the pattern will only
   # match if the origin point is set to a black pixel.
15  pattern origin (5,5) {
       # template pattern is based on examples in ‘‘Promenade’’
       template threshold 85 size (22,39) {
           # main dot
           polygon filled (0,2) (2,0) (8,0) (11,4) (12,10) (8,13) (0,10) (0,2)
20           # stem
           polygon filled (14,11) (21,2) (21,8) (10,38) (6,37) (17,12) (14,11)
           }
       }
   primitive
25  name          ‘‘semi_quaver_rest’’
   staff_height  86
   isolated_only yes
   pattern origin (15, 2) {
       template threshold 80 size (30,57) {
30           polygon filled (11,0) (7,5) (10,11) (22,13) (18,28)
```

```

        (10,29) (12,26) (8,21) (4,21) (0,24) (0,30) (8,33)
        (15,31) (7,56) (12,56) (29,0) (20,10) (18,3) (11,0)
    }
}
35
primitive
name          ''rect_rest''
staff_height  72
isolated_only yes
40 pattern {bbox (20,10), (26,15) } # filter on size
pattern origin (12,6) {
    template threshold 85 size (23,11) {
        polygon filled (0,0) (22,0) (22,10) (0,10) (0,0)
    }
45 }

primitive
name          ''flat''
staff_height  86
50 isolated_only yes
pattern { bbox (20, 51), (23, 57) }
pattern {
    template size (20,51) {
        polygon filled closed (0,0) (0,50) (2,50) (2,0) # stem
55        polygon filled closed (3,49) (17,41) (19,33) (14,28)
            (3,30) (3,33) (9,31) (12,35) (11,40) (3,46)
    }
}
# slightly smaller/pointier curve
60 primitive
name          ''flat%sharper''
staff_height  86
isolated_only yes
pattern { bbox (20, 51), (23, 57) }

```

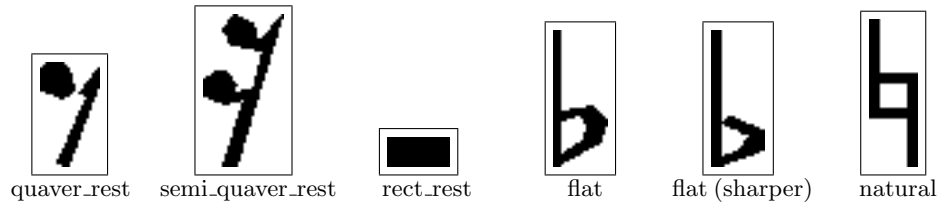


Figure A.1: Bitmaps described by sample pattern descriptions

```

65  pattern {
      template size(20,51) {
          polygon filled closed (0,0) (3,0) (3,50) (0,50)
          polygon filled closed (3,50) (19,44) (19,37) (7,32)
              (3,34) (8,36) (14,39) (9,45) (3,47) (3,50)
70      }
    }

    primitive
    name          ‘‘natural’’
75  staff_height  84
    isolated_only yes
    pattern { bbox (18, 55), (21, 68) }
    pattern {
      template size (18, 55) {
80      polygon filled closed (0,0) (0,36) (3,36) (3,0)
          polygon filled closed (14,20) (14,54) (17,54) (17,20)
          polygon filled closed (4,33) (4,36) (12,36) (11,33)
          }
    }
  }

```

A.3 Assembly Rules

The following listing is the default set of rules for performing primitive assembly of scores in the Western Music Notation. Lines starting with a '#' are comments, and are ignored by the OMR system.

```
1  # ORDER MATTERS!

    # set up a temporary type made up of a set of other types
    timesig_top E {digit_2, digit_3, digit_4, digit_5, digit_6, digit_7, digit_8}
5  timesig_bottom E {digit_2, digit_4, digit_8}

    # assignment
    #
    #  newobject = obj1 + obj2 [conditions]
10 #
    # creates a new object that is the two other objects combined
    #
    #
    #  newobject = obj1 <- obj2 [conditions]
15 #
    # creates a new object that has both as children, but only the
    # same dimensions and graphic data as the first object.

    timesig = timesig_top + timesig_bottom
20 # $1 refers to the object of the 1st type being tested, $2 for
    # the object of the 2nd type.
    # minimum and maximum distances (in pixels) - either minimum or both
    # values may be omitted
        where $1 is_above(5,15) $2
25 # this rule compares the left-hand-side edges of both objects
        and $1 lhs_difference(-10,10) $2

    full_notehead_dur_dots = full_notehead <- dot
        where $1 is_left_of(8,15) $2
30 # bottom of 1st object may not be above top of 2nd object
```

```

        and $1 !is_above $2
# top of 1st object may not be below bottom of 2nd object
        and $1 !is_below $2

35 hollow_notehead_dur_dots = hollow_notehead <- dot
        where $1 is_left_of(8,15) $2
        and $1 !is_above $2
        and $1 !is_below $2

40 full_notehead_types E {full_notehead, full_notehead_dur_dots}
hollow_notehead_types E {hollow_notehead, hollow_notehead_dur_dots}

# notes on left-hand-side of stem up
45 filled_note_stem_up = vertical_line <- full_notehead_types
        where $1 is_right_of(-5,5) $2
# within 20 pixels from top of notehead
        and $1 is_above(-20, 20) $2

50
# add notes on right-hand-side of stem up
# need to make sure noteheads aren't too close to top of stem
filled_notegr_stem_up = filled_note_stem_up + full_notehead_types+
        where $1 touches(-5,5) $2

55
# notes on right-hand-side of stem down
filled_note_stem_down = vertical_line <- full_notehead_types
        where $1 is_left_of(-5,5) $2
# within 20 pixels from top of notehead
60        and $1 is_below(-20,20) $2

# add notes on left-hand-side of stem down
filled_notegr_stem_down = filled_note_stem_down
        + full_notehead_types+

```



```

65         where $1 touches(-5,5) $2
#         where $1 is_right_of(-5,5) $2
#         and $1 !is_above(2) $2
#         and $1 !is_below(2) $2

70

hollow_note_stem_up = vertical_line + hollow_notehead_types+
        where $1 is_right_of(-5,5) $2
        and $1 !is_above(2) $2
75        and $1 !is_below(2) $2

hollow_note_stem_down = vertical_line + hollow_notehead_types+
        where $1 is_left_of(-5,5) $2
        and $1 !is_above(2) $2
80        and $1 !is_below(2) $2

# need to be able to join both types to horizontal with
# different rules...
filled_note E {filled_notegr_stem_up,
85        filled_notegr_stem_down,
        filled_note_stem_up, filled_note_stem_down}

notegroup = hough_horizontal + filled_note+
        where $1 touches(5) $2

90

## this could be tighter for vertical limits...
bass_clef = bass_clef_curl <- dot+
        where $1 is_left_of(3,10) $2
95        and $1 !is_above $2
        and $1 !is_below $2

```