

12-12-2014

# Utilizing Computer Programming to Analyze Post-Tonal Music: A Segmentation and Contour Analysis of Twentieth-Century Music for Solo Flute

Kate Sekula

*University of Connecticut - Storrs*, [katesekula@gmail.com](mailto:katesekula@gmail.com)

Follow this and additional works at: <https://opencommons.uconn.edu/dissertations>

---

## Recommended Citation

Sekula, Kate, "Utilizing Computer Programming to Analyze Post-Tonal Music: A Segmentation and Contour Analysis of Twentieth-Century Music for Solo Flute" (2014). *Doctoral Dissertations*. 628.  
<https://opencommons.uconn.edu/dissertations/628>

**Utilizing Computer Programming to Analyze  
Post-Tonal Music:  
A Segmentation and Contour Analysis  
of Twentieth-Century Music for Solo Flute**

Kate Sekula, Ph.D.

University of Connecticut, 2014

Two concepts will be synthesized in this dissertation: 1) the creation of accessible computer applications for melodic segmentation and contour reduction and 2) the application of segmentation and contour reduction to analyze twentieth-century post-tonal works for unaccompanied flute. Two analytical methodologies have been chosen: James Tenney and Larry Polanski's Gestalt segmentation theory and Robert Schultz's refinement of Robert Morris's contour reduction algorithm. The investigation also utilizes Robert Schultz's concept of diachronic-transformational analysis in conjunction with contour reduction. While both segmentation and contour reduction are invaluable analytical tools, they are meticulous and time-consuming processes. Computer implementation of these algorithmic procedures produces quick and accurate results while reducing analyst fatigue and human error. Microsoft Excel is used to complete melodic segmentation. Java programming language is used to create a contour reduction application. Each implementation greatly reduces the time needed to segment and analyze a melody. Computer programming is combined with pitch class set analysis to produce informed and expressive musical interpretations.





Utilizing Computer Programming to  
Analyze Post-Tonal Music:  
A Segmentation and Contour Analysis  
of Twentieth-Century Music for Solo  
Flute

Kate Sekula

B.S. Lebanon Valley College, 2001

B.A. Lebanon Valley College, 2002

M.M. George Mason University, 2003

A Dissertation  
Submitted in Partial Fulfillment of the  
Requirements for the Degree of  
Doctor of Philosophy  
at the  
University of Connecticut

2014

Copyright by

Kate Sekula

2014

APPROVAL PAGE

Doctor of Philosophy Dissertation

**Utilizing Computer Programming to Analyze  
Post-Tonal Music: A Segmentation and Contour  
Analysis of Twentieth-Century Music for Solo  
Flute**

Presented by

Kate Sekula, B.S., B.A., M.M.

Major Advisor

---

Ronald Squibbs

Associate Advisor

---

Richard Bass

Associate Advisor

---

Alain Frogley

University of Connecticut

2014

*To my husband,  
Jerome C. Sanders,  
without his love and support  
none of this would have been possible.*

# ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Ronald Squibbs, for his help and guidance during the writing of my dissertation. He introduced me to these topics many years ago in his course on advanced post-tonal analysis, creating the impetus for my entire research. Thank you to the members of my dissertation committee, Richard Bass, Alain Frogley, Peter Kaminsky and Jeffrey Renshaw, for the role they have played in my education.

My success at the University of Connecticut could not have been possible without the aid of the Department of Music staff members, especially Deb Trahan and Russell Ficarra. In addition, I am forever thankful to the staff of the University of Connecticut Department of Physics for all of their support and guidance especially: Alan Chasse, Kim Giard, Dave Perry, Doug Hamilton, Dawn Rawlinson, Lorraine Smurra, Cecile Stanzione, and Barbara Styrczula.

During my time at UConn I have had many families, and my heartfelt thanks must go to all of them. Thank you to the community of Emanuel Lutheran Church in Manchester, CT for providing me with a musical community beyond reproach. To my musical mentors, Jonathan Reuning-Scherer and Woosug Kang, your instruction and guidance will serve me for the rest of my life. Thank you to the UConn cycling club for giving me mental and physical strength. Thank you to the faculty and staff of the University of Science and Arts of Oklahoma for the camaraderie I share with you and the unsurpassed teaching environment you create.

To my fellow music graduate students, especially T.J. Bourne, John Corbet, Heather DeSavage, Rebecca Renfro Grimes, Cameron Logan, Teresa Olin, Sacha Peiser, Jessica Portillo, Carissa Reddick, Leann Sanders, Mike Sperr, Tim Shaw, and Jennifer Wanner and to my friends, especially, Erin Carey, Anagha Sabnis-Sambo, Nolan Samboy, and Ting-Yu Huang: thank you for your love, humor, and shoulders to cry on.

Thanks must be given to the publishers of the musical examples used herein and to the authors of the Computer Music Encyclopedia. Examples of Tōru Takemitsu's works are used with permission of the European American Music Distributors Company, sole U.S. and Canadian agent of Schott Music Co. Ltd., Japan. Examples of Kazuo Fukushima's works are used with permission of Sufarmusic S.p.A. - Suvini Zerboni, Milano (Italy). Terminology from the Computer Music Encyclopedia is reprinted with permission from The Computer Language Company ([www.computerlanguage.com](http://www.computerlanguage.com)).

Finally, I would like to thank my family. I am eternally grateful for the love, support, and guidance of my parents Steve and Annetta. Thank you to my brother,

Stephen, and my sister-in-law, Jodi, for opening their home to me on so many occasions, for introducing me to good wine, and for offering the love and wisdom that can only be given by those who have also gone through this process. Thank you to my other family John, Jan, Jenny, Stefan, Julie, Cory, and Jess for welcoming me as a daughter and sibling. And finally, my whole-hearted thanks to my husband J.C. You are the greatest gift I have ever received. Thank you for loving me and challenging me to be a better person every day.

# Contents

Abstract . . . . .	i
List of Figures . . . . .	viii
<b>1 Introduction</b>	<b>1</b>
1.1 Principles and Definitions . . . . .	2
1.2 Methodology . . . . .	6
1.3 Scope and Caveats . . . . .	8
1.4 Chapter Summary . . . . .	8
<b>2 Literature Review</b>	<b>9</b>
2.1 Contour Theory . . . . .	9
2.2 Segmentation Theory . . . . .	36
2.3 Computer-Assisted Analysis . . . . .	55
<b>3 Computer Programming Process</b>	<b>61</b>
3.1 A Short Coding Example . . . . .	64
3.2 Computer Program Structure . . . . .	70
3.2.1 Web Pages . . . . .	72
3.2.2 Java Source Packages . . . . .	73
3.2.3 Libraries . . . . .	81
3.2.4 Configuration Files . . . . .	83
3.2.5 Summary . . . . .	83
3.3 Microsoft Excel and TG Segmentation . . . . .	84
3.3.1 How to Use the TG Segmentation Spreadsheet . . . . .	90
3.4 Typesetting . . . . .	90
3.4.1 Lilypond . . . . .	90
3.4.2 L <sup>A</sup> T <sub>E</sub> X . . . . .	95
3.4.3 Summary . . . . .	96



<b>4</b>	<b>Analysis</b>	<b>98</b>
4.1	Kazuo Fukushima: Mei (1962) . . . . .	98
4.2	Kazuo Fukushima: Requiem (1966) . . . . .	119
4.3	Tōru Takemitsu: Itinerant (1989) . . . . .	134
4.4	Tōru Takemitsu: Air (1995) . . . . .	153
<b>5</b>	<b>Conclusion</b>	<b>175</b>
	<b>Glossary</b>	<b>178</b>
	<b>Appendices</b>	<b>196</b>
<b>A</b>	<b>Source Code: Robert Morris CRA</b>	<b>197</b>
<b>B</b>	<b>Source Code: main.js</b>	<b>205</b>
<b>C</b>	<b>Source Code: style.css</b>	<b>207</b>
<b>D</b>	<b>Source Code: index.jsp</b>	<b>214</b>
<b>E</b>	<b>Source Code: methods.class</b>	<b>217</b>
<b>F</b>	<b>Source Code: forte.class</b>	<b>224</b>
<b>G</b>	<b>Source Code: schultz.class</b>	<b>232</b>
<b>H</b>	<b>Source Code: servlets</b>	<b>266</b>
<b>I</b>	<b>Source Code: Microsoft Excel TG Segmentation Spreadsheet</b>	<b>271</b>
	<b>Bibliography</b>	<b>305</b>

# List of Figures

2.1	Charles Adams’s 15 Contour Types (Adams 1976, 199).	11
2.2	Arnold Schoenberg’s graphical representation of contour. Menuetto, String Quartet in D, K. 575, mvt. III, mm. 1-16 (Schoenberg 1967, 114).	12
2.3	Marvin and Laprade’s algorithmic process for determining contour prime (Marvin and Laprade 1987, 236).	16
2.4	A sample from Marvin and Laprade’s contour classification table (Marvin and Laprade 1987, 257).	16
2.5	Robert Morris’s 25 Basic Prime Classes (Morris 1993, 220).	18
2.6	Robert Morris’s contour reduction algorithm (Morris 1993, 221).	19
2.7	Mozart’s Piano Sonata No. 12, K. 332 with Beard’s labeling of x- and y- components (Beard 2003, 81).	22
2.8	Rob Schultz’s reinterpretation of the contour reduction algorithm (Schultz 2009, 130).	24
2.9	A portion of Rob Schultz’s Universal Tree Diagram showing contours up to cardinality 3 (Schultz 2009, 19).	26
2.10	Employment of Mustafa Bor’s 3-window algorithm on Schoenberg’s op. 19, no.4, second phrase (Bor 2009, 63).	27
2.11	Employment of Mustafa Bor’s 5-window algorithm on Schoenberg’s op. 19, no.4, second phrase (Bor 2009, 68).	28
2.12	Employment of Handelman and Sigler’s Z-chain algorithm to the opening melody of Petzold’s Minuet in G major (Handelman and Sigler 2011, 12).	30
2.13	Yi-Cheng (Daniel) Wu’s algorithm for determining CSIM-AT (Wu 2012, 132).	31
2.14	Charles Adams’s 15 contour types related using a contour smooth network (Wu 2012, 136).	32
2.15	An example of <i>nesting</i> . Webern, Five Pieces for Orchestra Op. 10 no. 4, example B (Forte 1973, 93).	38

2.16	An example of <i>imbrication</i> . Pitch-classes taken from Webern, Five Pieces for Orchestra Op. 10 no. 4, example B (Forte 1973, 93). . . . .	39
2.17	Process for Temporal Gestalt Segmentation (the numbering of the process in steps is my own, but is based on the material presented in Tenney and Polansky 1980, 206–27). . . . .	41
2.18	Lefkowitz and Taavola’s grouping of musical dimensions into four domains Lefkowitz and Taavola 2000, 176. . . . .	48
3.1	CAT web application. . . . .	64
3.2	Example of source code for determining mod 12. . . . .	65
3.3	Verbal algorithm for determining mod 12 of a pitch represented as an integer. . . . .	66
3.4	Basic operation of computer programming language. . . . .	69
3.5	Basic operation of computer programming language as applied in this dissertation. . . . .	69
3.6	Components of the web application CAT. . . . .	70
3.7	Flowchart of web application structure. . . . .	71
3.8	Verbal algorithm for determining normal form. . . . .	76
3.9	Verbal algorithm for determining prime form. . . . .	78
3.10	Corrections made to Jay Tomlin’s Set Theory Calculator HashMap. .	79
3.11	Portion of fully calculated TG segmentation spreadsheet for Kazuo Fukushima’s <i>Mei</i> for solo flute. . . . .	87
3.12	Example of a macro written using VBA which determines the pitch interval at the <i>clang</i> level of TG segmentation. . . . .	89
3.13	An example of music engraving using Lilypond: J.S. Bach BWV 610. .	92
3.14	An example of music engraving using Lilypond: Schenker graph. . . .	92
3.15	An example of music engraving using Lilypond: Ancient notation (Lilypond Notation Reference, section 2.9). . . . .	93
3.16	Example of Lilypond syntax. . . . .	94
3.17	Engraved output of Lilypond syntax shown in Figure 3.16. . . . .	94
3.18	An example of $\text{\LaTeX}$ syntax. This example comes from Unwalla 2006, 33. . . . .	95
3.19	Output for example of $\text{\LaTeX}$ syntax shown in Figure 3.18. . . . .	96
4.1	Kazuo Fukushima’s <i>Mei</i> for solo flute. Both Lee’s formal division and TG segmentation are indicated. © Sugarmusic S.p.A. - Suvini Zerbini, Milano (Italy) . . . . .	100

4.2	<i>Mei</i> discontinuities and Total Segmentation Values (TSVs) in measures 1-26 (Lefkowitz and Taavola 2000, 198). . . . .	109
4.3	Synchronic contour results of Kazuo Fukushima's <i>Mei</i> for solo flute. .	114
4.4	DTA contour primes from beginning to all 244 pitches of Kazuo Fukushima's <i>Mei</i> . . . . .	116
4.5	Percentage of contour primes occurring diachronically in Kazuo Fukushima's <i>Mei</i> . . . . .	118
4.6	Kazuo Fukushima's <i>Requiem</i> for solo flute labeled with twelve-tone row construction and TG segmentation. © Sugarmusic S.p.A. - Suvini Zerboni, Milano (Italy) . . . . .	120
4.7	Synchronic contour results for <i>Requiem</i> . The third column displays pitches as integers representing the range of the flute where C4=0 and C7=36. . . . .	127
4.8	Surface occurrences of contour prime <2 3 0 1> in Kazuo Fukushima's <i>Requiem</i> . © Sugarmusic S.p.A. - Suvini Zerboni, Milano (Italy) . . .	129
4.9	Diachronic contour primes from beginning to all 162 pitches of Kazuo Fukushima's <i>Requiem</i> . . . . .	131
4.10	Percentage of contour primes occurring diachronically in Kazuo Fukushima's <i>Requiem</i> . . . . .	132
4.11	Segmentation of Tōru Takemitsu's <i>Itinerant</i> for solo flute. Takemitsu ITINERANT in Memory of Isamu Noguchi, SJ 1055 Copyright © 1989 by Schott Music Co. Ltd., Japan All Rights Reserved Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan . . . . .	137
4.12	Synchronic contour results for <i>Itinerant</i> . . . . .	144
4.13	Diachronic contour primes from beginning to all 266 pitches of Tōru Takemitsu's <i>Itinerant</i> . . . . .	145
4.14	Percentage of diachronic contour primes occurring diachronically in Tōru Takemitsu's <i>Itinerant</i> . . . . .	147
4.15	Analysis of Tōru Takemitsu's <i>Itinerant</i> for solo flute (dynamics and text markup removed for legibility). Takemitsu ITINERANT in Memory of Isamu Noguchi, SJ 1055 Copyright © 1989 by Schott Music Co. Ltd., Japan All Rights Reserved Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan . . . . .	148

4.16	Form diagram of sections and subsections for Tōru Takemitsu’s <i>Air</i> for solo flute. Sections are labeled 1-6. Subsections are labeled A, B, or C. Measure numbers are given for each subsection. . . . .	154
4.17	Segmentation of Tōru Takemitsu’s <i>Air</i> for solo flute. Takemitsu AIR dedicated to Auréle Nicolet for his 70th birthday, SJ 1096 Copyright © 1995 by Schott Music Co. Ltd., Japan All Rights Reserved Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan . . . . .	156
4.18	Analysis of Tōru Takemitsu’s <i>Air</i> for solo flute (dynamics and text markup removed for legibility). Pitch-class set collection use is marked above the staff. Motivic gestures a-d are labeled below the staff. Takemitsu AIR dedicated to Auréle Nicolet for his 70th birthday, SJ 1096 Copyright © 1995 by Schott Music Co. Ltd., Japan All Rights Reserved Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan . . . . .	163
4.19	Diachronic contour primes from beginning to all 472 pitches of Tōru Takemitsu’s <i>Air</i> . . . . .	170
4.20	Percentage of contour prime occurring diachronically in Tōru Takemitsu’s <i>Air</i> . . . . .	174

# Chapter 1

## Introduction

Two concepts are synthesized in this dissertation: 1) the creation of accessible computer applications for melodic segmentation and contour reduction and 2) the application of segmentation and contour reduction to analyze twentieth-century post-tonal works for unaccompanied flute. Two analytical methodologies have been chosen: James Tenney and Larry Polansky's temporal gestalt segmentation theory and Rob Schultz's refinement of Robert Morris's contour reduction algorithm. The investigation also utilizes Rob Schultz's concept of diachronic-transformational analysis in conjunction with contour reduction. While both segmentation and contour reduction are useful analytical tools, they are meticulous and time-consuming processes. Computer implementation of these procedures produces quick and accurate results while reducing analyst fatigue and human error. Microsoft Excel is used to complete melodic segmentation. Java computer programming language is used to create a contour reduction web application. Each implementation greatly reduces the time needed to segment and analyze a melody. Computer programming is combined with pitch-class set analysis to produce informed analyses.

## 1.1 Principles and Definitions

Contour is the function that tracks a musical parameter over time.<sup>1</sup> Generally, the parameter is pitch because contour is primarily recognized as a melodic property. However, contour analysis has become significant for other parameters, such as dynamics and duration. It is also an important structural feature in the music of many post-tonal composers such as Iannis Xenakis, Olivier Messiaen, Edgard Varèse, and György Ligeti (Morris 1993, 205).<sup>2</sup> The fact that composers of post-tonal music specifically turned to contour as a formative ingredient has made the implementation of contour analysis a necessity.

David Cope, in his book *Hidden Structure: Music Analysis Using Computers*, provides three more reasons for focusing computer analysis on post-tonal music:

First...tonal music analysis has a long and distinguished history that details approaches to melody, harmony, counterpoint, form and structure. Although the advent of computer technology can certainly add tools and allow the discovery of new concepts, the range of potential for truly new approaches is, I feel, somewhat limited compared to the potential for analytical possibilities in post-tonal music. Second, many diverse and useful programs for tonal analysis already exist. In fact, most early computer computational experiments with music analysis involved tonal music. On the other hand, although many computer pitch-class set programs exist,

---

1. While other terms will be introduced in this document that have the same definition as *parameter* i.e. domain or dimension, for clarity and universality the author uses only the term *parameter* to represent any of a set of compositional or auditory properties whose values determine the characteristics or behavior of a piece of music unless speaking about another scholar's specific method.

2. The term “post-tonal” is used in this document to refer to any type of music composition in which there is no hierarchy of functional harmony, no distinction between consonance and dissonance, for which a traditional tonal analysis analysis does not yield satisfying analytical results or for which a pitch-class set analysis alone is not sufficient to illuminate the structure of the music.

little beyond this easily created software is currently available. Third, and possibly most important, post-tonal music represents the *lingua franca* of today's concert music...(Cope 2008, xxii)

I have chosen the work of Robert Morris and Rob Schultz for computer implementation.<sup>3</sup> Robert Morris applies set theory to contour, creating hierarchical contour relationships, applying contour to any musical parameter, and determining a measure for contour equivalence. He is largely responsible for a generalized theory of contour with formal methodology. He combines set theory with contour theory and demonstrates how “pitch classes and their sets, brought out by contour hierarchies, are related to each other as well as to adjacent pitch-class sets by abstract intersection and complement relations” (Morris 1993, 206).<sup>4</sup> The end result is “a complete taxonomy of all contour types” (*ibid*).

Morris's process allows for analysis and comparison of contours of any cardinality. The core of his theory is the *contour reduction algorithm*. An algorithm is a finite, step-by-step process for solving a problem which frequently involves repetition of an operation (Cope 2008, 3). Through implementation of the contour reduction algorithm, Morris delineates local high and low points of a contour segment, called *maxima* and *minima*, and recursively eliminates all non-maxima and non-minima (called *pruning*) until a contour segment's fundamental structure, or *contour prime*, is revealed. He introduces the concept of *contour depth* which measures how many times one traverses through the algorithm before reaching the contour prime. The contour prime generates a hierarchical level of contour pitch salience and the contour depth number gives a rough measure of the complexity of a contour.

Rob Schultz refines Morris's contour reduction algorithm and recognizes two flaws.

---

3. See Morris 1987, Morris 1993, Schultz 2008, and Schultz 2009.

4. For a complete discussion of set theory, see Forte 1973.



First, it fails to reduce “wedge-shaped” contours; that is, a contour in which every pitch is a maximum or minimum. Second, the manner in which it deals with repeated contour pitches provides no indication of which repeated contour pitch should be pruned. Arbitrary pruning of a repeating maximum or minimum may result in different contour primes at the deepest level.

Rob Schultz combines contour theory, phenomenology, and genealogy into a general theory of temporally ordered contour relationships. From a phenomenological perspective, Schultz uses the work of Edmund Husserl to explain that contours should be heard as a series of “now-points.” A listener does not conceive that a contour instantaneously exists. Rather, they hear it unfold through time. Contour analysis should account for this phenomenon. His dissertation focuses on a *diachronic* view of contour rather than a *synchronic* view. Diachronic relates to how something evolves through time. Synchronic refers to elements which exist at the same time; they are concurrent.

Schultz incorporates diachronic phenomenology with the contour reduction algorithm. The algorithm passes over the contour as each new contour pitch occurs, changing the cardinality. As a result, the first, last, highest, and lowest contour pitches and the contour depth number are all subject to continued revision as the contour unfolds from the initial contour pitch to its final length. The result is a set of embedded contours of differing maxima/minima and contour depth.

Segmentation is the process of dividing a composition into smaller, meaningful parts (Lefkowitz and Taavola 2000, 171). In order to apply contour analysis to longer musical spans, it becomes necessary to determine how to divide a work into components. This task is generally left to the psychological perception of the listener or performer. However, the process can be completed through the use of algorithms.

Theories of segmentation exhibit a wide and variable syntax.<sup>5</sup> Terminology overlaps between theories, but with changing definitions. The theories focus on a range of musical genres, from monophonic music of the troubadours and trouvères, to the classical idiom, to twentieth-century post-tonal music. While the syntax and focus of each method is wide-ranging, the authors all reach the same basic conclusion: that musical segmentation is a linear, context-driven process based on continuity/discontinuity in which aural perception must be paired with analysis of the musical surface.

James Tenney and Larry Polansky present an algorithmic computer process based on similarity and proximity of the parameters of pitch, time, timbre, and intensity.<sup>6</sup> Their 1980 article established the first widely-accepted algorithmic segmentation process. They describe a methodology and implementation of a computer model for generating a segmentation of a monophonic work.

Tenney and Polansky’s algorithm is based on gestalt psychology, the operational principle of which is that the brain has self-organizing tendencies and is concerned with whole or complete systems. This principle maintains that the human eye sees objects in their entirety before perceiving their individual parts (Hergenhahn 2013, 446). In terms of Tenney and Polansky’s model, segmentation is the process by which the gestalt principles of *similarity* and *proximity* group musical objects that are closer together or more similar to each other. The grouping of objects creates what Tenney and Polansky call *temporal gestalt units*.

Tenney and Polansky define five terms which correspond to the different hierarchical levels of perception: *element*, *clang*, *sequence*, *segment*, and *section*. An *element* is a temporal gestalt unit that is not further divisible. It is the smallest possible unit.

---

5. See Tenney and Polansky 1980, Nattiez and Barry 1982, Lerdahl and Jackendoff 1983, Friedmann 1985, Lefkowitz and Taavola 2000, Hanninen 2001, Schultz 2008, Schultz 2009, and Hanninen 2012.

6. See Tenney and Polansky 1980.

A *clang* is a temporal gestalt unit that consists of two or more successive elements. Any next higher level must consist of two temporal gestalt units at the previous level. The various levels represent a bottom-up approach to musical division and create hierarchical levels from smallest to largest. The result is a segmentation algorithm with a consistent approach.

The concepts of melodic segmentation and contour reduction are algorithmic and therefore lend themselves to computer implementation. Computers provide music theorists with opportunities to analyze music quickly and accurately. They supply the means to accomplish music analyses in a fraction of the time of doing them by hand, with far more accuracy and without human error. Computer programming, when combined with music analysis, presents unique opportunities for music theorists to better understand the music they study. Computers not only reduce fatigue and increase accuracy, but also allow for analyses previously considered unachievable (Cope 2008, xxi).

## 1.2 Methodology

For segmentation, an interactive macro-enabled Microsoft Excel spreadsheet was created in which an end user enters preliminary numerical data for the parameters of pitch, time, intensity, and timbre (or any other parameter). The end user may also input weighting values for each parameter. Embedded macros in the spreadsheet perform all operations to complete a temporal gestalt segmentation of the data.

The art of computer programming was used for implementation of contour analysis and reduction. Computer programming is the process that leads from an original formulation of a computing to an executable program (Nakov 2013, 73). The purpose

of computer programming is to find a sequence of instructions that will automatically perform a specific task or solve a given problem. Quality programming code should be reliable, robust, usable, portable, maintainable, and efficient. While there has been much research into computer programming for music analysis, many of the computer programs fail to demonstrate all of these characteristics. Often, web links to these computer programs are broken, the source code is out of date for the current edition, the computer programs are old and have not been transferred to an updated platform, or there is a large learning curve for utilizing a highly specialized computer program.

The chosen programming language is Java. Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is designed to have as few implementation dependencies as possible.<sup>7</sup> It can be run across multiple platforms.

Java programming language was used to create an application for computing Rob Schultz's contour reduction algorithm. This computer program, called CAT, is accessed through the web. Pitches are entered as integers with spaces in between. The computer program allows for decimals, so microtones are easily accounted for. The computer program output presents the overall contour, the contour depth as well as the normal form, prime form and, if applicable, the Forte set class label. The application can give a snapshot of the overall contour, as if it had instantaneously occurred, or provide repetitive analysis as each new pitch is added to a melody (or any parameter which the analyst chooses that can be represented as an integer). The application can also open a Musical Instrument Digital Interface (MIDI) file. When paired with software like Finale, Sibelius or Photoscore, it is quick and easy to analyze an entire piece of music.

---

7. See Farrell 2013.

## 1.3 Scope and Caveats

This dissertation only analyzes post-tonal music for solo flute, specifically the works of composers Kazuo Fukushima and Tōru Takemitsu. However, the methods herein could be applied to any monophonic piece of music. One caveat of these methods is that only monophonic music can be analyzed.<sup>8</sup> While the author does not deny that composer intent and listener cognition are important for music analysis, particularly of works written in the twentieth and twenty-first centuries, this study focuses primarily on objective principles and data analysis.

## 1.4 Chapter Summary

Chapter 2 provides a literature review of all previously published theories of segmentation, melodic contour, and melodic analysis using computers and demonstrates how computers and music analysis can be combined to develop a rich set of analytical tools. Chapter 3 discusses the methodology of the dissertation, including a detailed description of the coding process, explanation of the contour reduction application, and explanation of the segmentation spreadsheet. Chapter 4 uses the new computing applications to offer analyses of four solo flute works by Kazuo Fukushima and Tōru Takemitsu. Chapter 5 concludes the dissertation with generalizations of the analytical results and discussion of future applications of computer programming to music analysis.

---

8. For information on polyphonic segmentation see Meredith, Lemstrom, and Wiggins 2002.

# Chapter 2

## Literature Review

### 2.1 Contour Theory

This section reviews literature from the last forty years pertaining to contour theory. Contour is the function that tracks a parameter, such as pitch, over time. The study of contour was born out of ethnomusicological studies of folk melodies and focused primarily on melodic pitches. More recently it has been applied to post-tonal music and to parameters other than pitch, such as dynamics or duration. As contour theory has evolved, the vocabulary used to describe this feature of music has become more technical, pulling terminology and metaphors from other research areas such as linguistics, phenomenology, genealogy, mathematics, and biology. The aim of this review is to summarize each method, show the evolution from one method to another, and highlight the most important algorithmic functions used to determine the equivalence of two contours.

Charles Adams wrote the first comprehensive discussion of contour analyses (Adams 1976). He arranges previous discussions of melodic contour into four categories: 1) methods which are narrative in nature, 2) methods that display contour through a string of symbols, 3) methods that use metaphorical descriptions, and 4) methods that use graphical representations of a melody. Adams concludes that there is no

unanimous methodology for contour analysis and proposes the need for a specific definition of contour. He presents the first codification of an analytical method for contour analysis.

Adams defines contour as “the product of distinctive relationships among the minimal boundaries of a melodic segment”(Adams 1976, 195). A melodic segment is “any series of differentiated pitches”(ibid). The minimal boundaries are the first, last, highest, and lowest pitch of a melody. He labels these boundaries as Initial (I), Final (F), Highest (H), and Lowest (L) and describes the relationship between these elements as less than ( $<$ ), greater than ( $>$ ), and equal to ( $=$ ).

He defines three primary and three secondary relationships between the minimal boundaries. The primary relationships are slope (S), deviation (D), and reciprocal (R). (S) is the relationship between I and F. (D) is the change of direction of (S). (R) is the relationship between the first (or only) (D) and I.

Secondary relationships define a *melodic contour shape*. They are repetition (REP) of H and L, recurrence of H and L without intervening boundary pitches (RA), and recurrence of H and L with intervening boundary pitches (RB). The product of these relationships define fifteen melodic contour types (**Figure 2.1**).

Adams applies his method to two sets of Native American folksongs. Although he recognizes that his study does not cover or resolve all issues pertaining to contour theory, his method represents the first steps towards the clarification and specific implementation of contour theory.

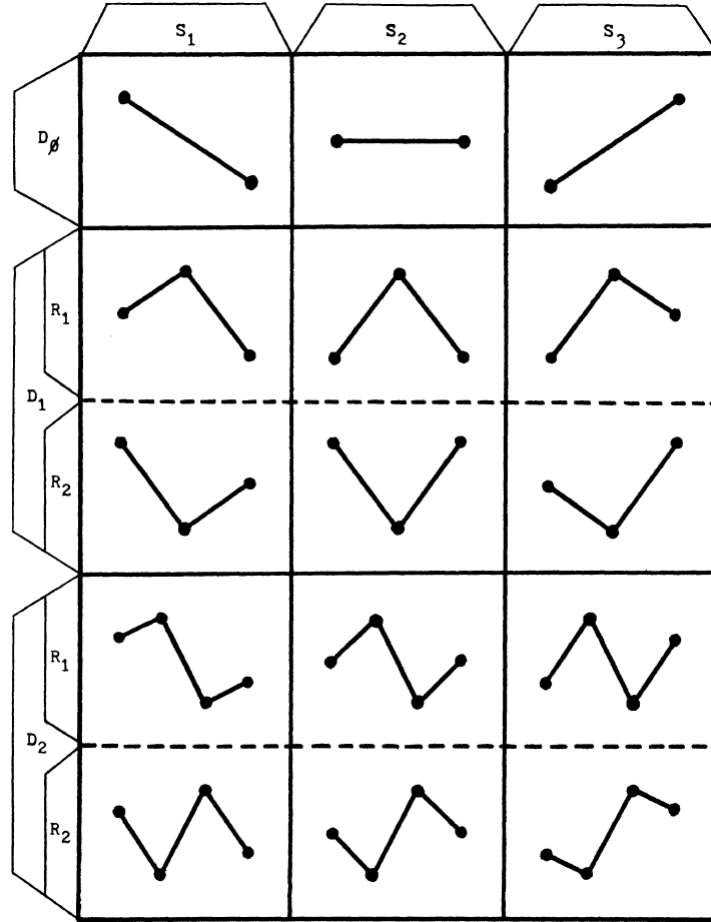


Figure 2.1: Charles Adams’s 15 Contour Types (Adams 1976, 199).

At this point it is worth mentioning that in Adams’s extensive description of contour analysis categories and analysts, he fails to mention Arnold Schoenberg’s 1967 posthumous publication *Fundamentals of Musical Composition* (Schoenberg 1967). Schoenberg’s description of melody would fall into Adams’s category of graphical representations. Schoenberg provides a verbal description of contour and discusses the “singableness” of a melody, regardless of whether or not it is written for voice. He presents graphic depictions of melodies which are then superimposed over the staff. An example of such an analysis can be seen in **Figure 2.2**.



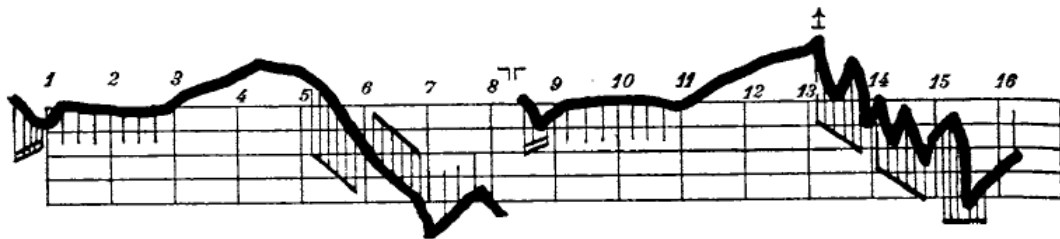


Figure 2.2: Arnold Schoenberg's graphical representation of contour. Menuetto, String Quartet in D, K. 575, mvt. III, mm. 1-16 (Schoenberg 1967, 114).

Michael Friedmann discusses the contour of post-tonal music, specifically that of Schoenberg (Friedmann 1985). He suggests that the “contour of a musical unit may be the melodic parameter that is most easily grasped” by a listener (Friedmann 1985, 224-5). Although he offers no formal definition of contour, he develops an analytical system which applies the techniques of set theory to contour.<sup>1</sup>

Two characteristics of his method must be pointed out: 1) he determines contour based on the relative position of pitches and 2) his method allows for the analysis of adjacent and non-adjacent pitches. Friedmann proposes two tools for describing contours: the *contour adjacency series* (CAS) and the *contour class* (CC).

The CAS is a succession of directional moves up or down in a melody. They are designated, similarly to Adams's relationships of (H) and (L), by the symbols + and -. For example, in the melodic segment (C4, G3, E4, C4) the CAS is < -, +, - >, symbolizing the motion of the melody between the pitches C4 and G3 < - >, G3 and E4 < + >, and E4 and C4 < - >.<sup>2</sup> He uses the CAS to identify order rotation, inversion, and retrograde of a contour and establishes contour equivalence akin to set class equivalence.

The CC explains the position of the pitches (high or low) in relation to each other.

1. For a complete discussion of set theory, see Forte 1973.

2. This document uses the octave identification standard as established by The Acoustical Society of America in Young 1939.

Friedmann orders the pitches of a segment from lowest to highest, renaming the lowest 0 and the highest  $n - 1$ , where  $n$  equals the number of distinct pitches in a musical unit. In the previous example the CC would be  $\langle 1 \ 0 \ 2 \ 1 \rangle$ . The lowest pitch, G3, is renamed 0 and the remaining pitches are renamed based on a hierarchy from low to high.

Using these tools, Friedmann finds shape similarities between segments with different pitch content. He introduces the idea of contour *subsets*, allowing for embedding relationships between contours. He also establishes that contour analysis need not only be used in the pitch domain, but may also be applied to other compositional domains, such as timbre and register.

Much like Michael Friedmann, Robert Morris applies the concept of set theory to contour, creates hierarchical contour relationships, applies contour to any musical parameter, and determines contour equivalence (Morris 1987). Morris is largely responsible for a generalized theory of contour with formal methodology.

Morris defines contour as an ordered set of *contour pitches* (cps). The cps of a *contour segment* (cseg) generate a *contour space* (c-space). Morris's c-space is equivalent to Friedmann's CC. C-space is the ranking of cps from low to high, ignoring the exact intervals between cps. He introduces two more concepts: the *interval succession* (INT<sup>+</sup>) and the *comparison matrix* (COM-matrix). The INT<sup>+</sup> is equivalent to Friedmann's CAS. The COM-matrix is a two-dimensional representation of the INT<sup>+</sup> results. It compares a cseg to itself. For the INT<sup>+</sup>  $\langle 0 \ 3 \ 2 \ 4 \ 1 \rangle$  the COM-matrix is:

	0	3	2	4	1
0	0	+	+	+	+
3	-	0	-	+	-
2	-	+	0	+	-
4	-	1	1	0	-
1	-	+	+	+	0

Each data value across the top of the array is compared to each data value along the side of the array. If the values are the same, the result is (0). If the horizontal point is greater than ( $>$ ) the vertical point, the result is (+). If the horizontal point is less ( $<$ ) than the vertical point, the result is (-).

The COM-matrix allows Morris to determine equivalence between contours of the same cardinality. Contour equivalence is determined using two properties: 1) similitude of COM-matrices and 2) relationship by the transformational properties of Identity (P), Transposition (T), Inversion (I), and Retrograde Inversion (RI). Contour equivalence reveals relationships between csegs containing disparate pitch content.

Elizabeth West Marvin and Paul A. Laprade use Morris’s concept of contour equivalence as their point of departure (Marvin and Laprade 1987). They do not offer a definition of contour. They define a cseg, much like Morris, as “an ordered set of cps in c-space.” They define a *contour subsegment* (csubseg) as “any ordered sub-grouping of a given cseg” (Marvin and Laprade 1987, 228). A csubseg may be made up of contiguous or non-contiguous cps.

Marvin and Laprade determine deeper contour equivalence relationships by defining three functions that determine contour similarity. Unlike Morris, these functions can compare csegs of the same or different cardinality. The functions are 1) *contour similarity* (CSIM), 2) *contour embedding* (CEMB), 3) *contour mutual embedding of n cardinality* (CMEMB<sub>n</sub>), and 4) *adjusted contour mutual embedding* (ACMEMB).

CSIM uses Morris’s COM-matrices to measure the CSIM of two csegs of equal cardinality. It tallies the number of equivalent entries in the upper right-hand triangles of the COM-matrices of the csegs being compared. This number is then divided by the total possible number of entries and returns a value between 0 and 1. 0 indicates dissimilarity. 1 indicates maximal similarity.

The use of the CEMB function is restricted to contours of unequal cardinality. This function tallies the number of times the smaller cseg occurs as a csubseg of the larger cseg, regardless of cp adjacency. This value is then divided by the total number of possible csubsegs, also returning a value between 0 and 1.

They adjust CEMB to measure CSIM between csegs of equal or unequal cardinality with the use of the function CMEMB<sub>*n*</sub>, where *n* equals a certain cardinality and CMEMB<sub>*n*</sub> measures how many times a csubseg of *n* cardinality occurs in the cseg.

ACMEMB tallies the total number of mutually embedded csubsegs within any two csegs and divides the result into the total number of possible mutually embedded csubsegs, again returning a value from 0 to 1. This function can determine CSIM between csegs of equal or unequal cardinality.

Through these four functions, Marvin and Laprade can compare the structures of two csegs. The resulting decimal numbers from each function offer a general idea about the level of CSIM between the csegs. Marvin and Laprade transform csegs into a “normal form,” which is equivalent to Morris’s c-space (and Friedmann’s CC). They define a “prime form” for each cseg in which the first cp of the cseg must be lower than the last. They offer an algorithmic process (shown in **Figure 2.3**) to determine contour prime form and display the results in a table of contour prime forms. A sample of this table may be seen in **Figure 2.4**. The csegs in the contour table range from two to six cps. Marvin and Laprade assign classification labels of the same type

as Allen Forte's set class labels.<sup>3</sup>

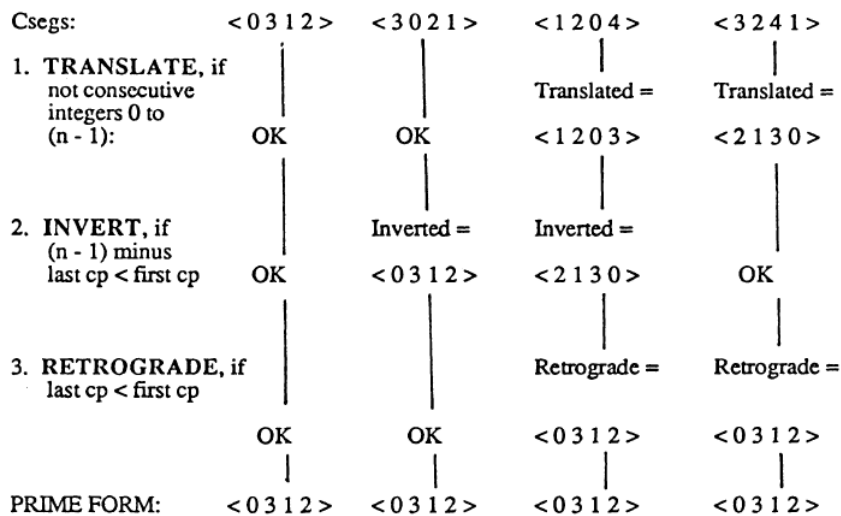


Figure 2.3: Marvin and Laprade's algorithmic process for determining contour prime (Marvin and Laprade 1987, 236).

C-space segment classes for cseg cardinality 2		
Csegclass/RIinv.	Prime form	INT(1)
c 2-1*	< 0 1 >	< + >
C-space segment classes for cseg cardinality 3		
Csegclass/RIinv.	Prime form	INT(1)
c 3-1*	< 0 1 2 >	< + + >
c 3-2	< 0 2 1 >	< + - >
C-space segment classes for cseg cardinality 4		
Csegclass/RIinv.	Prime form	INT(1)
c 4-1*	< 0 1 2 3 >	< + + + >
c 4-2	< 0 1 3 2 >	< + + - >
c 4-3*	< 0 2 1 3 >	< + - + >
c 4-4	< 0 2 3 1 >	< + + - >
c 4-5	< 0 3 1 2 >	< + - + >
c 4-6	< 0 3 2 1 >	< + - - >
c 4-7*	< 1 0 3 2 >	< - + - >
c 4-8*	< 1 3 0 2 >	< + - + >

Figure 2.4: A sample from Marvin and Laprade's contour classification table (Marvin and Laprade 1987, 257).

Larry Polansky and Richard Bassein offer the first critique of the previous approaches (Polansky and Bassein 1992). They are concerned with the melodic per-

3. For a discussion of set classes see Forte 1973.

ception of a listener and how contour affects melody memory and recognition. They point out that recent research in music perception has shown that “listeners are most sensitive to adjacent relationships” (Polansky and Bassein 1992, 260). Their critique of previous contour theories (Friedmann, Morris, and Marvin and Laprade) is that they allow for *combinatorial contour*, that is, analysis of both adjacent and non-adjacent cps. Analyzing non-adjacent cps goes against research concerning melodic perception.

They define contour as a series of relationships ( $>$ ,  $<$ ,  $=$ ) between cps. They re-label these relationships as 0 ( $<$ ), 1 ( $=$ ) and 2 ( $>$ ). They state that using a combinatorial approach to determine all possible contours of a certain cardinality creates both possible and impossible contours. An impossible contour is a contour that does not correspond to an actual melodic shape. In the process of determining all possible combinatorial contours, many impossible contours are created which violate the law of transitivity. So while we may be able to calculate, using a COM-matrix, all ‘conceivable’ contours, the results may not contain contours which are ‘possible.’ Polansky and Bassein determine that if all impossible contours can be removed, leaving only possible contours, then what is left could serve as “a formal lexicon for large-scale analysis” (Polansky and Bassein 1992, 272).

Robert Morris’s follow-up article to Marvin and Laprade reviews the state of contour theory and its utilization for analysis (Morris 1993). He combines set theory with contour theory and determines how “pitch-classes and their sets, brought out by contour hierarchies, are related to each other as well as to adjacent [pitch-class] sets by abstract intersection and complement relations” (Morris 1993, 206). The end result is “a complete taxonomy of all contour types” (*ibid*).

Morris’s process allows analysis of contours of any cardinality. At the core of his

theory is the *contour reduction algorithm* (CRA, from here on). The full algorithm is reproduced in **Figure 2.6**. Through the implementation of the CRA, Morris delineates the local high and low points of a cseg, which he calls *maxima* and *minima* respectively, and recursively eliminates all non-maxima and non-minima (a process which he calls “pruning”) until a cseg’s fundamental structure, or “prime,” is revealed. He introduces the concept of *contour depth* which measures how many times one traverses through the algorithm before reaching the prime. The prime generates a hierarchical level of cp salience and the depth number gives a rough measure of the complexity of a contour. Once the prime is determined, one can use a COM-matrix to determine contour relationships. Morris determines 25 basic prime classes, shown in **Figure 2.5**.


























	1 distinct pitch:	2 distinct pitches:	3 distinct pitches:	4 distinct pitches:
1 time- point	A: <0> 	C: <{01}> 		
2 time- points	B: <0 0> 	D: <0 1>  E: <{0 1} 0>  F: <{01} {01}> 	H: <{01} 2>  I: <{0 2} 1>  J: <{01} {02}>  K: <{01} {12}> 	Q: <{01} {23}>  R: <{02} {13}>  S: <{03} {12}> 
3 time- points		G: <0 1 0> 	L: <0 2 1>  M: <{01} 2 0>  N: <{01} 2 1>  O: <1 {02} 1> 	T: <{01} 3 2>  U: <{02} 3 1>  V: <0 3 {12}>  W: <1 {03} 2> 
4 time- points			P: <1 0 2 1> 	X: <1 0 3 2>  Y: <1 3 0 2> 

Figure 2.5: Robert Morris’s 25 Basic Prime Classes (Morris 1993, 220).

Algorithm: Given a contour C and a variable N

0. Set N to 0.
1. Flag all maxima in C; call the resulting set the *max-list*.
2. Flag all minima in C; call the resulting set the *min-list*.
3. If all pitches in C are flagged, go to step 9.
4. Delete all non-flagged pitches in C.
5. N is incremented by 1 (i.e., N becomes N+1).
6. Flag all maxima in the max-list. For any string of equal and adjacent maxima in the max-list, either: (1) flag only one of them; or (2) if one pitch in the string is the first or last pitch of C, flag only it; or (3) if both the first and last pitch of C are in the string, flag (only) both the first and last pitch of C.
7. Flag all minima in the min-list. For any string of equal and adjacent minima in the min-list, either: (1) flag only one of them; or (2) if one pitch in the string is the first or last pitch of C, flag only it; or (3) if both the first and last pitch of C are in the string, flag (only) both the first and last pitch of C.
8. Go to step 3.
9. End. N is the “depth” of the original contour C.

Figure 2.6: Robert Morris’s contour reduction algorithm (Morris 1993, 221).

Ian Quinn finds that Morris and Marvin and Laprade fail to provide meaningful results due to their dependence on exact contour equivalence (Quinn 1997). He argues that two contours only need to be *sufficiently* equivalent for a meaningful discussion of similarity to occur. The formal rigor of the methods of Morris and Marvin and Laprade “leaves little room for the slight variation inherent in any group characterized by family resemblance instead of equivalence” (Quinn 1997, 237).

Quinn incorporates *fuzzy logic*, logic based on approximation rather than a binary true/false relationship, to allow a certain amount of flexibility to Morris’s COM-



matrix. He uses Marvin and Laprade’s CSIM function to determine a minimum similarity threshold which he then uses to determine if a given cseg is part of a larger, familial, group of csegs.

He creates algorithm A<sup>6</sup> which consists of two major steps: 1) find the average of all contours to be analyzed and 2) judge each contour against the average contour using a similarity function. Step 1 requires finding the *ascent relation* ( $C^+$ ) between each member of the contour.  $C^+$  simply measures, for an ordered pair of cps (p,q), whether q is greater than p. From this data, Quinn creates a new type of matrix, the  $C^+$  *matrix*. A  $C^+$  matrix is the same as a COM-matrix, except it only represents whether there was an ascent (1) or no ascent (0). Once a  $C^+$  matrix for each cseg is determined, all  $C^+$  matrices are averaged together by summing the corresponding values of each matrix and dividing by the total number of contours in the group being analyzed. The values of this final matrix are weighted based on how often a contour occurs in the group. If a contour occurs four times within a contour group, then the items in its  $C^+$  matrix are counted four times towards the average. The result is a matrix containing values from 0 to 1.

Step 2 compares each member of this contour group to the average of the group. This is done through computing the similarity function between the contour member and the group average. Quinn’s version of this similarity function is called  $C^+SIM$ . Contours with a  $C^+SIM$  above a certain threshold, determined by the analyst, are then considered to have a strong relationship to the average contour.

R. Daniel Beard uses a more direct mathematical approach in his 2003 dissertation on contour (Beard 2003). He defines contour in terms of a vector space as “the general shape of an object, often, but not exclusively, associated with elevation or height, as a function of distance, length or time” (Beard 2003, 1). He lists the previous methods

of Adams, Morris, Friedmann, Marvin and Laprade, and Quinn and presents the critique that all of these methods replace exact interval content with relative distance and disregard rhythm and duration. He introduces a theory that incorporates these missing elements.

He incorporates pitch and rhythm into a theory of contour through the use of *multiple linear regression* in which data is presented as an ordered pair of numbers and converted into both a graphic plot and an equation that represents them. He uses his method to analyze the opening melodies of Mozart's piano sonatas.

The first step is to convert the pitches and durations of a melody into numerically equivalent ordered pairs of numbers (x, y), where x represents duration and y represents pitch. The system used to represent numerical equivalence is an arbitrary decision made by the analyst. Beard chooses to assign the value of zero to the lowest tonic in the melody. Each half-step is a movement by a whole number in either the positive or negative direction. For duration, he assigns the first downbeat the value of 1 and each subsequent beat represents a whole number. Half, eighth and sixteenth-beats are represented as decimals. **Figure 2.7** shows one of Beard's examples in which he has labeled the pitches, relative to the lowest tonic, and the rhythmic attack points (in this example he has labeled the downbeat as 0, but his point is that this labeling choice is arbitrary).<sup>4</sup>

---

4. Beard labels this example as Piano Sonata No. 1, K. 332. The sonata is generally listed at No. 12, so this identification is used in this dissertation.



Figure 2.7: Mozart’s Piano Sonata No. 12, K. 332 with Beard’s labeling of x- and y-components (Beard 2003, 81).

Beard reveals two basic contour shapes in Mozart’s piano melodies: undulating melodies with large vertical changes in the y-coordinates, which he calls MD (“Manic-Depressive”), and melodies that are relatively flat, but which often contain a “spike,” which he calls LB (“Laid-Back”). He arrives at these types by applying Marvin and Laprade’s similarity functions to his data, specifically to the melodic “extrema,” the points along the graph that ultimately define its shape. His method shows not just simply a function of change in pitch level along a unitless scale, but also shows the function of pitch attack points in time and, using computing and mathematics, determines overall melodic movement.

Rob Schultz refines Morris’s CRA in his 2008 article (Schultz 2008). He finds two flaws with the CRA. First, it fails to reduce “wedge-shaped” contours; contours in which every pitch is a maximum or minimum. Second, the manner in which it deals with repeated cps provides no indication of which repeated cp should be pruned. Arbitrary pruning of a repeating maximum or minimum may result in different csegs at the deepest level.

Schultz modifies the algorithm so that Step 3 (see **Figure 2.6**) no longer prematurely ends the algorithm before every cp has been subjected to the pruning procedure

at least once. This solves the issue pertaining to wedge-shaped contours, since in Morris’s CRA none of the cps would be flagged in step 1 and step 2, leading to no cps being pruned and no reduction of the contour. He modifies the algorithm further to include additional instruction for the flagging of adjacent maximum or minimum. If two adjacent maxima have an intervening minimum (or if two adjacent minima have an intervening maximum) then the adjacent maximum (or minimum) is flagged and retained. If there is no intervening cp, then only one of the adjacent cps is kept. Finally, he proposes the addition of two new contour prime classes to Morris’s basic contour prime classes:  $\langle 1\ 0\ 2\ 0\ 1 \rangle$  and  $\langle 1\ 0\ 3\ 0\ 2 \rangle$ , which allow for the irreducible adjacent minima which are a product of the new version of his algorithm. The final version of Schultz’s algorithm is shown in **Figure 2.8**.

Schultz continues his work with contour in his 2009 dissertation (Schultz 2009). He combines contour theory, phenomenology, and genealogy into a general theory of temporally ordered contour relationships. From a phenomenological perspective, Schultz uses the work of Edmund Husserl to explain that contours should be heard as a series of “now-points.” A listener does not conceive that a contour instantaneously exists. Rather, they hear it unfold through time. Contour analysis should account for this phenomenon. His dissertation focuses on a *diachronic* view of contour rather than a *synchronic* view of contour. Diachronic relates to how something evolves through time. Synchronic means that elements, such as cps, exist at the same time; they are concurrent.

Algorithm: Given a contour  $C$  and a variable  $N$ :

0. Set  $N$  to 0.
1. Flag all maxima in  $C$  upwards; call the resulting set the *max-list*.
2. Flag all minima in  $C$  downwards; call the resulting set the *min-list*.
3. If all c-pitches are flagged, go to step 6.
4. Delete all non-flagged c-pitches in  $C$ .
5.  $N$  is incremented by 1 (i.e.,  $N$  becomes  $N + 1$ ).
6. Flag all maxima in the max-list upward. For any string of equal and adjacent maxima in the max-list, flag all of them, unless: (1) one c-pitch in the string is the first or last c-pitch of  $C$ , then flag only it; or (2) both the first and last c-pitches of  $C$  are in the string, then flag (only) both the first and last c-pitches of  $C$ .
7. Flag all minima in the min-list downward. For any string of equal and adjacent minima in the min-list, flag all of them, unless: (1) one c-pitch in the string is the first or last c-pitch of  $C$ , then flag only it; or (2) both the first and last c-pitches of  $C$  are in the string, then flag (only) both the first and last c-pitches of  $C$ .
8. For any string of equal and adjacent maxima in the max-list in which no minima intervene, remove the flag from all but (any) one c-pitch in the string.
9. For any string of equal and adjacent minima in the min-list in which no maxima intervene, remove the flag from all but (any) one c-pitch in the string.
10. If all c-pitches are flagged, and no more than one c-pitch repetition in the max-list and min-list (combined) exists, not including the first and last c-pitches of  $C$ , proceed directly to step 17.
11. If more than one c-pitch repetition in the max-list and/or min-list (combined) exists, not including the first and last c-pitches of  $C$ , remove the flags on all repeated c-pitches except those closest to the first and last c-pitch of  $C$ .
12. If both flagged c-pitches remaining from step 11 are members of the max-list, flag any one (and only one) former member of the min-list whose flag was removed in step 11; if both c-pitches are members of the min-list, flag any one (and only one) former member of the max-list whose flag was removed in step 11.
13. Delete all non-flagged c-pitches in  $C$ .
14. If  $N \neq 0$ ,  $N$  is incremented by 1 (i.e.,  $N$  becomes  $N + 1$ ).
15. If  $N = 0$ ,  $N$  is incremented by 2 (i.e.,  $N$  becomes  $N + 2$ ).
16. Go to step 6.
17. End.  $N$  is the “depth” of the original contour  $C$ .

Figure 2.8: Rob Schultz’s reinterpretation of the contour reduction algorithm (Schultz 2009, 130).

Any contour begins with an initial cp. From this point it could proceed to a cp equal to, greater than, or less than itself, resulting in three possible contours:  $\langle 0\ 0 \rangle$ ,  $\langle 0\ 1 \rangle$  or  $\langle 1\ 0 \rangle$ . With the addition of a third cp, each of these options must also be evaluated to determine how the contour will proceed, generating thirteen possible three-note contours. The number of possibilities increases greatly as the cardinality of the contour increases. Schultz displays these transformational chains in the form of a Universal Tree Diagram (a small portion of which can be seen in **Figure 2.9**) and relates the contours to each other in Mendelian genealogical terms (parent, sibling, cousin, etc.). The Universal Tree Diagram shows all possible contours as a melody proceeds from left to right through time. The left-most column shows the first cp of the contour, the parental generation (P). Subsequent columns, or generations, are called filial generations ( $F_1$ ,  $F_2$ , etc.). Every contour is a unique transformational path along the Universal Tree Diagram and contour similarity is based on how similar two contours' paths are along the tree.

Although Schultz also discusses overlapping functions and creates a transformational network based on Lewinian practices, this literature review will bypass these topics and move on to his discussion of Morris's CRA.<sup>5</sup> Schultz combines Husserlian phenomenology with the CRA. The original Universal Tree Diagram is labeled depth = 1. The CRA passes over the contour as the cardinality changes. As a result the first, last, highest, and lowest cps and the depth number are subject to continued revision as the contour unfolds from the parental cp to its final length. The result is a Universal Tree Diagram whose various csegs contain contours of differing maxima/minima and depth due to the reiteration of the CRA.

---

5. For detailed descriptions of transformational theory see Lewin 1982, Lewin 1987, and Lewin 1993.

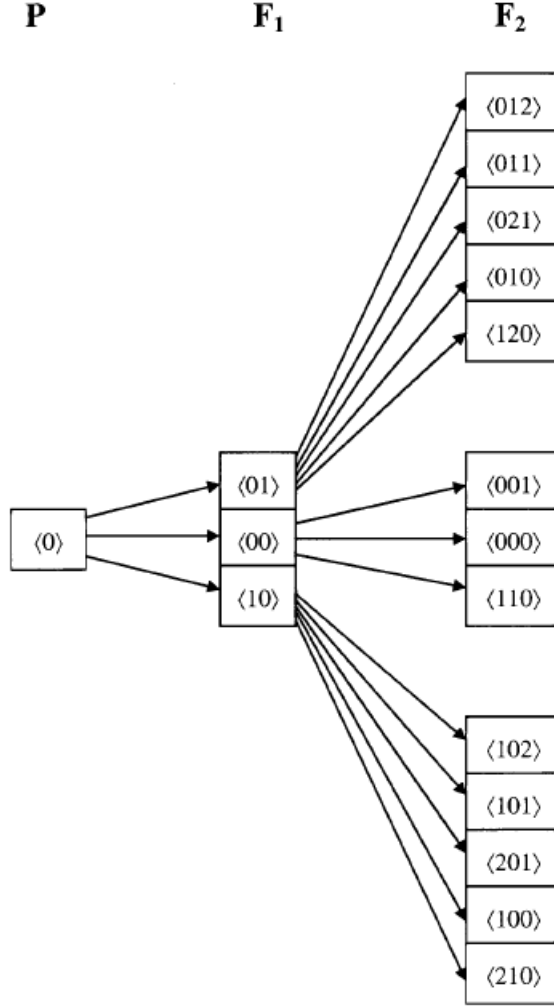


Figure 2.9: A portion of Rob Schultz’s Universal Tree Diagram showing contours up to cardinality 3 (Schultz 2009, 19).

Mustafa Bor proposes a new set of algorithms based on the CRA in his 2009 dissertation (Bor 2009). He describes two stages of the Morris algorithm. The first stage evaluates cps in a single pass. The second stage only evaluates the maximum or minimum. His criticism, similar to the that of Schultz, is that using two distinct ways of evaluating cps for pruning weakens the utility and consistency of the algorithm.

Bor offers a solution that differs from Schultz. He proposes two different algorithmic processes which provide the methodological benefits of reiteration, but use a single type of pruning at all stages. His solution is to use what he calls a 3-window

and a 5-window algorithm.

In the 3-window algorithm, the center cp is evaluated in comparison to the preceding and succeeding cps (just as in the first stage of Morris's CRA). There is an imaginary frame that is three cps wide and slides from left to right one cp at a time. An example of this motion can be seen in **Figure 2.10**. Note in this example that frames 1 and 12 only contain two pitches. Bor uses a null value on either side of the cseg. Therefore, in frame 1, C $\sharp$ 5 is compared to only E5 and is maintained because it is a minimum and in frame 12 A $\sharp$ 4 is compared to G $\sharp$ 4 and is maintained because it is a maximum. This holds true for the 5-window algorithm. Unfortunately, this pruning via the 3-window may only be performed once and no further reduction is possible.



Figure 2.10: Employment of Mustafa Bor's 3-window algorithm on Schoenberg's op. 19, no.4, second phrase (Bor 2009, 63).

To allow for further iterations, Bor creates a 5-window algorithm. In a 5-window, a central cp is evaluated in comparison to the two preceding and two succeeding cps. If it is a maximum or a minimum, it is retained. An example of this motion can be seen in **Figure 2.11**. The 5-window algorithm allows for further pruning through reiteration of the process. For each iteration, the depth is increased by 1. Bor states that an analyst can switch between the two algorithms to get flexible answers of a



specific depth or cardinality.



Figure 2.11: Employment of Mustafa Bor’s 5-window algorithm on Schoenberg’s op. 19, no.4, second phrase (Bor 2009, 68).

Eliot Handelman and Andie Sigler propose a method which works with exact pitches on the music surface, rather than converting them to a prime form without interval relationships (Handelman and Sigler 2011). Instead of segmenting a melody, they find all of the interlocking and overlapping smaller structures with the purpose of showing the flow from one pattern to the next. They investigate the particular details of a melody and take into account listeners’ expectations; specifically how thwarting a listener’s expectation of simple, recursive patterns influence their perception of the music.

Like previous methodologies, they describe three intervallic orientations, although they refer to them as “up,” “down,” and “be” (same). They develop the term *Z-chain*, which is the core of their methodology. A *chain* is “a series of like orientations” i.e. “up-chain,” “down-chain” and “be-chain” (Handelman and Sigler 2011, 7). Chains are the simplest melodic shape. A *Z-chain* is a repetitive inspection of the melody for a chain with a specific orientation. The term Z-chain comes from the “zig-zaggy” shape of the patterns created (Handelman and Sigler 2011, 9). Handelman and Sigler “decompose” a melody into Z-chains. *First-order* Z-chains are the simplest. They show all the possible movements of “up,” “down,” and “be.” In **Figure 2.12** the

top staff is the original melody. First-order Z-chains are labeled “up,” “down,” and “be.” *Second-order* Z-chains are determined by comparing the orientation of an element, appointed by the analyst, between first-order Z-chains. In **Figure 2.12**, tHandelman and Sigler chose to compare the orientation between the last pitch of each first-level up-chain resulting in two new staves called “up-up” and “up-down.” The labels represent the intervallic motion of the first-order and second-order Z-chains. “Up-up” means that the chain was an up-chain at the first order and is also an up-chain (meaning that the last pitch of each first-order up-chain also moved up) at the second-order.

This technique is akin to a Schenkerian reduction.<sup>6</sup> It reveals patterns, parallels, and other melodic relationships not only at the musical surface, but also at increasingly deeper levels as the Z-chain process is reiterated.

Yi-Cheng (Daniel) Wu synthesizes systems of contour theory by Adams and Marvin and Laprade with the transformational networks of David Lewin, the voice-leading methods of Joseph Straus, and the set theory of Allen Forte, creating a method to analyze both the vertical and the horizontal aspects of twentieth-century music (Wu 2012).<sup>7</sup> His theory is tailored to analyze compositions with imitative voices. It considers voice-leading and contour at the same time and regards them as creating a “unitary musical element” (Wu 2012, xviii).

Wu’s *average Transformed Voice Pair interval class set* (average-TVPicset) determines the compactness of a vertical sonority. He finds the interval classes (ics) between all pairs of adjacent pitches and averages their sum into a *transformed interval class* (tic). The smaller the number, the more compact the harmony is.

---

6. For more on the theories of Heinrich Schenker, see Schenker 1935.

7. For a discussion of twentieth-century voice-leading, see Straus 2003.



Figure 2.12: Employment of Handelman and Sigler's Z-chain algorithm to the opening melody of Petzold's Minuet in G major (Handelman and Sigler 2011, 12).

The motion from one verticality to another creates simultaneous melodic voices. Wu analyzes these voices using the techniques of Adams, and Marvin and Laprade. First, he adopts Adams's fifteen contour types as the basis for his contour study (see **Figure 2.1**). He creates a contour similarity algorithm which measures the similarity between any pair of Adams's contour types. He calls this *contour similarity of Adams's type* (CSIM-AT). To establish a CSIM-AT he first determines the slope

of the two contours being compared (the relationship of the initial pitch to the final pitch). Then he finds the absolute difference between the slope of the first contour and the slope of the second contour (i.e. how many changes need to occur to the first contour to transform it into the second contour). This results in a similarity range from 0-2, with 0 meaning that the contours contain no similar cps and 2 meaning that the contours are identical. Then he determines the deviation between the two contours; that is, the difference in cardinality. This also results in a range from 0-2, with 0 meaning that they are of the same cardinality and 2 meaning there is a difference of 2 between the cardinalities of the contours.

He combines these two range scores to determine the CSIM-AT. The similarity of two contours is represented as a number from 0 to 4, where 0 represents the greatest similarity and 4 represents a total dissimilarity of cps between two contours. The complete process of his algorithm can be seen in **Figure 2.13**.

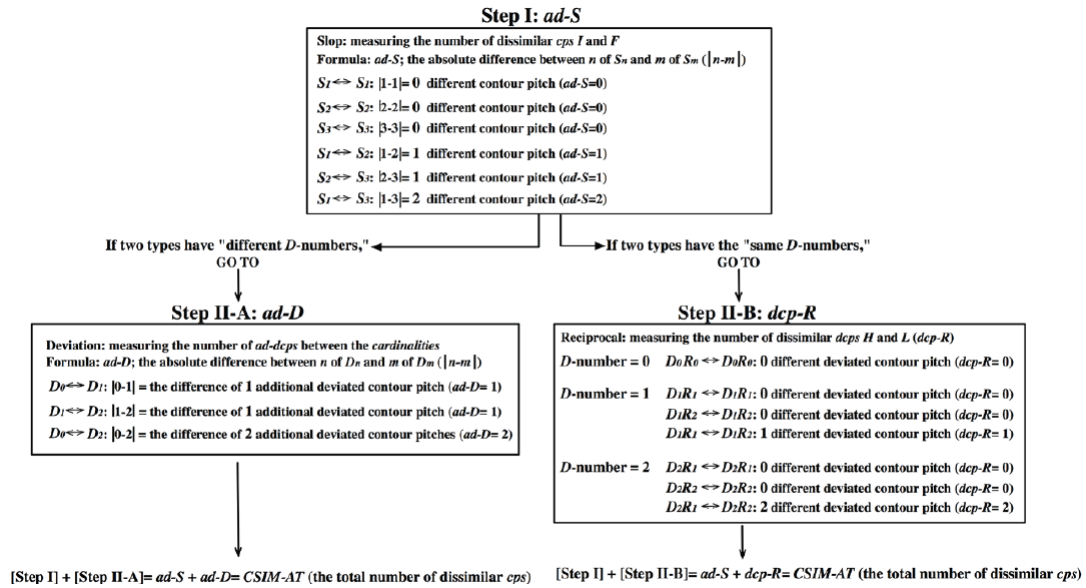


Figure 2.13: Yi-Cheng (Daniel) Wu's algorithm for determining CSIM-AT (Wu 2012, 132).

Wu creates a *contour smooth network*. This network is a uniformly balanced con-

tour web, similar to the transformational networks of David Lewin, which connect all contours that differ by only one cp. An example of this network can be seen in **Figure 2.14**. He uses this web to map the shortest distance between two contours, therefore finding the smoothest way to move from one contour to another. By combining the average-TVPicset with the CSIM-AT and contour smooth network, Wu is able to derive measurements of the harmonic degree of compactness and contour similarity in a given section of music.



Figure 2.14: Charles Adams's 15 contour types related using a contour smooth network (Wu 2012, 136).

Finally, in a recent article by Mitchell S. Ohriner, contour theory becomes a tool used to determine relationships between duration rather than pitch (Ohriner 2012).

Ohriner is interested in expressive performance timing in tonal music, specifically that of Chopin. He introduces the term *group final lengthening* (GFL), where a deceleration occurs at the ends of groups (a group is equivalent to a phrase, or a cseg). This term is taken from linguistics where it is known as *phrase-final lengthening*. He analyzes the duration contours of a group. If a group has a CAS (Friedmann 1985) of  $< + >$  (duration lengthened) or  $< - + >$  (duration accelerated and then lengthened) then that group's duration contour is *GFL-reflective*, meaning that there is a deceleration or a lengthening duration. He uses Morris's CRA to reduce durational contours. They must reduce to  $< + >$  or  $< - + >$  to be considered GFL-reflective.

Ohriner uses the software Sonic Visualizer to analyze the spectrogram of 24 different artists performing various works by Chopin. He divides the music into various levels of time-span organization (2 bars, 4 bars, 8 bars) and looks for relationships between the occurrence or nonoccurrence of GFL between the various artists. The existence of GFL at the same points in each performance would represent agreement between performers as to the grouping structure of each work.

The conclusion is that studies in timing in performance can demonstrate relationships between GFL and grouping structure when performers are *already in broad agreement* on the structure. However, when the performers disagree about the grouping, this relationship can fail to exist.

To summarize, Charles Adams was the first scholar to publish a review of previous contour analysis methods. He defined contour as a product of the minimal boundaries of a melody (initial, final, highest, and lowest pitches) and created fifteen contour types which can be used for a comparative analysis of two contours. Michael L. Friedmann uses contour to analyze post-tonal music. His method finds similar-

ity between contours of different pitch content by implementing the CAS and CC functions. He also introduces contour subsets. Robert Morris creates a language for discussing contour, introducing the terms of c-space, cp, cseg, etc. He combines the functions of the COM-matrix and INT<sup>+</sup> with transformational properties to determine equivalence between contours of the same cardinality. Elizabeth West Marvin and Paul A. Laprade extend Morris's concept of contour equivalence to contours of the same or different cardinality through use of the functions CSIM, CEMB, CMEMB<sub>n</sub>, and ACMEMB. These functions produce decimal numbers which describe the degree of similarity between two csegs. They develop an algorithm to determine a contour normal form from its prime and then classify the normal forms in the same manner as Allen Forte. They compile a table of all normal forms with cardinality 2 through 6.

Larry Polansky and Richard Bassein point out that some c-space segment classes create melodically impossible contours. Robert Morris analyzes contours of any length with the CRA. Through employment of the CRA he can determine different hierarchical contour levels and contour complexity. Ian Quinn uses fuzzy logic to determine the minimal similarity threshold to determine cseg inclusion in a familial cseg class. R. Daniel Beard uses multiple linear regression to determine contour characteristics without losing the features of exact pitch and duration. His regression plots determine overall melodic motions of pitch and pitch attack through time. Rob Schultz reconfigures the CRA to repair methodological errors. He then combines contour theory with phenomenology and genealogy to produce a diachronic analysis of contour that reveals the transformational nature of a contour through time.

Mustafa Bor uses the CRA methodology of reiteration to create 3-window and 5-window algorithmic processes. His windows provide the methodological benefits

of Morris's reiteration process, while only using one type of pruning at all stages of reduction. Handelsmann and Sigler's Z-chains allow for a Schenkerian-type contour analysis with varying levels of relationships shown directly on the music surface, without removing intervallic relationships. Their method demonstrates how a contour can interlock and overlap. Yi-Cheng (Daniel) Wu integrates all popular, current twentieth-century analytical tools into a complete method for analyzing twentieth-century contrapuntal music. His contour smooth network shows the most closely-related contour types in Charles Adams's classification system. Using the contour smooth network can illuminate a smooth voice-leading process as one contour moves to another.

Researching these methods clarifies an evolutionary process of contour reduction from Charles Adams through Robert Schultz. Adams's designations for Initial (I), Final (F), Highest (H), and Lowest (L) pitches in folk melodies developed into Morris's *maxima* and *minima*. Morris added recursion to the process and established terminology for discussing hierarchical contour relationships between post-tonal melodies. Rob Schultz critically examined and removed weaknesses from Morris' original algorithm. The remainder of the methods represent outgrowths from contour reduction and generally attempt to incorporate it with trending post-tonal analytical methodologies. The combination of the contour reduction algorithm with diachronic/synchronic contours provides a well-defined construct for determining hierarchical contour relationships.



## 2.2 Segmentation Theory

This section reviews literature from the last fifty years pertaining to melodic segmentation theory. Each theorist discussed herein offers a different definition for the term *segmentation*. They exhibit a wide and variable syntax for describing segmentation. Terminology overlaps between theories, but with changing definitions. The theories focus on a range of musical genres, from monophonic music of the troubadours/trouvères, to the Classical idiom to 20th-century post-tonal music. While the syntax and focus of each method are wide-ranging, the authors all reach the same basic conclusion: that musical segmentation is a linear, context-driven process based on continuity/discontinuity in which aural perception must be paired with analysis of the musical surface. Once all methods are presented, I shall offer a definition for *segmentation* specific to the purposes of this particular study. What will arise is the evident need to create one methodology and syntax for determining and describing musical segmentation.

Nicolas Ruwet’s *Méthodes d’Analyses en Musicologie*, originally published in 1966, is an early attempt to clearly present an essential method for melodic segmentation (Ruwet and Everist 1987). He emphasizes the need for defining the analytical criteria used to determine segmentation and asserts that previous analyses presupposed such criteria. His methodology is influenced by studies of grammatical language, a trend which will continue throughout future theories.

Ruwet describes segmentation as “the process of division” (Ruwet and Everist 1987, 15). He identifies relationships between musical fragments based on recurrence and/or repetition of musical variables. He chooses one, *non-parametric* musical variable at a time and determines how it functions to divide the musical surface.

A *non-parametric variable* is a musical aspect which is not constant throughout a piece, such as pitch, duration, intensity (dynamics) or scale. A non-parametric variable also does not function in a binary on/off manner (eg. the way the texture of a concerto-grosso alternates between *ripieno* (on) and *concertino* (off)). Based on this single-variable approach, Ruwet creates an algorithmic process for melodic segmentation in which repetitions and transformations delineate segments. The process repeats until musical fragments are irreducible.

Ruwet describes his process as a “machine for identifying elementary identities” (Ruwet and Everist 1987, 17). After determining which variable to use, the analyst passes this ‘machine’ over the musical surface, identifying the largest possible fragments and resultant ‘remainders.’ During this process, an analyst may find that the chosen variable is inadequate, requires improvement, or needs to be rejected.

While Ruwet’s work represents an initial attempt to define the criteria by which analysts determine a segmentation, he implements it only on monophonic music of the twelfth through the fourteenth centuries. While this repertoire serves well for demonstrative purposes, the algorithm is only applicable in the simplest of cases.

Allen Forte’s discussion of segmentation has been mostly ignored in segmentation studies (Forte 1973). However, his discourse lays a good foundation for the segmentation of atonal music. Forte defines segmentation as “the procedure of determining which musical units of a composition are to be regarded as analytical objects” (Forte 1973, 83). He remarks that segmentation in tonal music is more easily perceptible than in atonal music because tonal music contains familiar structural patterns. Forte centers his discussion on the segmentation of atonal music, which is more problematic due to the lack of “familiar morphological formations” (Forte 1973, 83).

Forte defines two basic terms: *primary segment* and *composite segment*. A *pri-*

*primary segment* is a musical unit demarcated by conventional means, such as by rests. *Composite segments* are formed by segments or subsegments that are contiguous or that are otherwise linked in some way. These subdivisions are determined through *imbrication* and *nesting*. In the process of *nesting*, the analyst systematically joins elements of the segment. For example, one might join pitches from left to right, as shown in **Figure 2.15**, and look for an emerging pattern of pitch-class sets. *Imbrication* is the process of segmenting a composite segment into smaller, overlapping subsegments (see **Figure 2.16**). The result is an analysis depicting both stratification and intersection of pitch-class sets. An analyst can use contextual criteria, such as recurrence and set complexes, to determine relationships. Importantly, knowledge of a particular composer’s style provides context for segment determination. This process results in “a temporal stratification of primary and composite segments” (Forte 1973, 91). The amount of stratification can illuminate the complexity of a piece or passage.

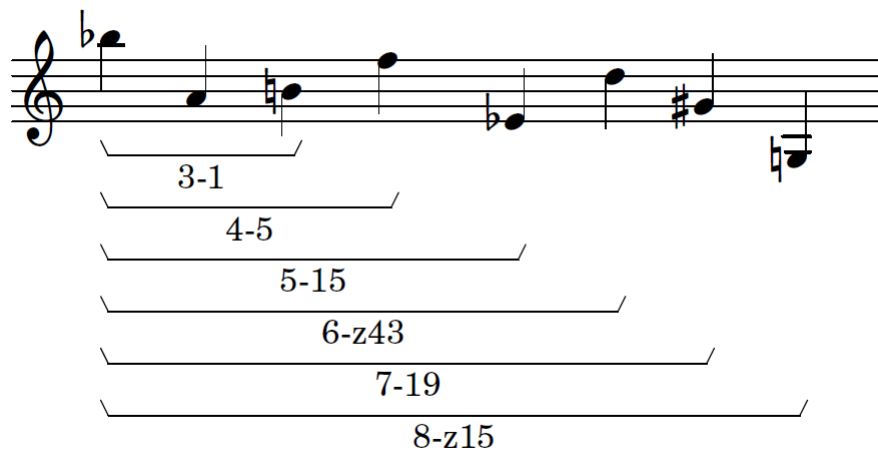


Figure 2.15: An example of *nesting*. Webern, Five Pieces for Orchestra Op. 10 no. 4, example B (Forte 1973, 93).

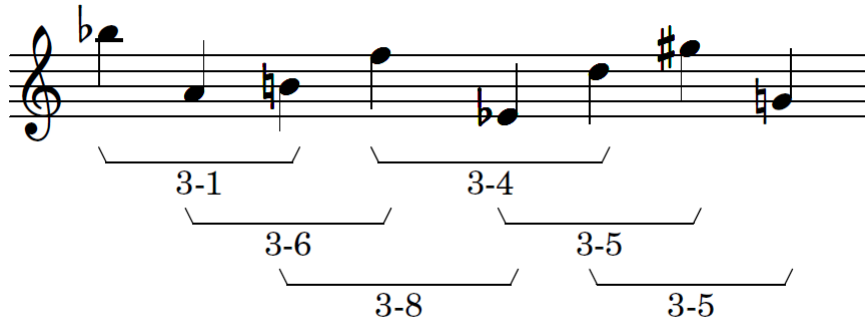


Figure 2.16: An example of *imbrication*. Pitch-classes taken from Webern, Five Pieces for Orchestra Op. 10 no. 4, example B (Forte 1973, 93).

Forte highlights two issues that arise from this segmentation process. First, it may produce insignificant structural units and editing may be required. Second, while Forte advocates the use of musical context he admits that “it is virtually impossible to systematize all of the contextual criteria used to determine segmentation” (Forte 1973, 91). While his discussion is very general, he highlights important concepts utilized by future theorists, such as context sensitivity and knowledge of a given composer.

James Tenney and Larry Polansky establish the first widely-accepted algorithmic segmentation process (Tenney and Polansky 1980). They describe an algorithmic methodology for segmentation and the implementation of a computer model for generating melodic segmentation. In this section, I will focus on their methodology and will discuss the computing process in Chapter 3.

The algorithm of Tenney and Polansky (hereafter referred to as T & P) is based on gestalt psychology. The operational principle of gestalt psychology is that the brain has self-organizing tendencies and is concerned with whole or complete systems. This principle maintains that the human eye sees objects in their entirety before perceiving their individual parts (Hergenhahn 2013, 446). In terms of T & P’s model,

segmentation is the process by which the gestalt principles of *similarity* and *proximity* group musical objects that are closer together or more similar to each other. The grouping of objects creates what T & P call “temporal gestalt units” (TGs).

T & P define five terms which correspond to the different hierarchical levels of perception: *element*, *clang*, *sequence*, *segment*, and *section*. An *element* is a TG that is not further divisible. It is the smallest possible unit. A *clang* is a TG that consists of two or more successive *elements*. The next two levels are referred to as *segments* and *sections* and the highest level is the entire piece or movement. The various levels represent a bottom-up approach to musical segmentation and create hierarchical levels from smallest to largest.

The algorithmic process for determining *clangs* from *elements* (or any other hierarchical level from the preceding level) is shown in **Figure 2.17**. There is no restriction on the number of factors which can be considered. T & P use examples which illustrate pitch and time. In this dissertation, a third factor, intensity, is added during the computer modeling process. All values are expressed as integers. Steps 1 - 3 determine *clang* initiation.<sup>8</sup> Steps 4 - 6 determine *sequence* initiation as well as initiation for all higher levels.

A common argument against this algorithm is with the arbitrary weighting system applied to the variables until an optimal segmentation is achieved.<sup>9</sup> A weighting system solves conflicts between pitch interval and duration, but the rate to which a variable is weighted is controlled by the analyst. Also, T & P focus on the basic parameters of pitch, time, and intensity. However, other musical parameters such as articulation, meter, register or repetition can offer criteria for segmentation.

---

8. The numbering of the process in steps is my own, but is based on the material presented in Tenney and Polansky 1980, 206-27.

9. See Lefkowitz and Taavola 2000, 173, Nattiez and Barry 1982, 329, Uno and Huebscher 1994, 10, and Hanninen 2001, 370-2.

1. Determine the distance interval for each parameter other than time being examined between each *element*. This is the sum of the absolute value of the distance between each *element*. The labeling method is contextually determined.
2. Determine the distance interval for the parameter of time between each *element*. Again, the labeling method is contextually determined.
3. Sum together the parameter distance interval(s) and time distance interval between each *element*. *Clang* initiation occurs when the summed value between two *elements* is greater than the summed value of the *element* on either side.
4. Determine the mean of the distance values within each *clang* or parameters other than time. Then find the distance interval between the mean values. This is called the mean-interval value.
5. Determine the time distance from the first pitch of each *clang* (or whatever the next lowest level is) to the starting pitch of the next *clang*. This is called the mean-interval value for time.
6. Sum the parameter and time mean intervals.
7. Determine the boundary intervals for each parameter, including time. This is found by calculating the difference between the final *element* of a *clang* (or whatever the next lowest level is) and the initial *element* of the next *clang*.
8. Sum all parameter boundary intervals. This calculation is hierarchical and is divided by an increasing factor of 2 for each new level. For example, for the *sequence* level the sum is divided by 2; for *segments* by 4; for *glsplsection* by 8, etc.
9. Sum the mean interval sum and boundary sum (divided by a factor of 2). This is the *disjunction* value. *Sequence* (or any future level) termination occurs at the end of *clangs* (*sequences*, *segments*, etc.) where the disjunction value is greater than the disjunction value on either side.

Figure 2.17: Process for Temporal Gestalt Segmentation (the numbering of the process in steps is my own, but is based on the material presented in Tenney and Polansky 1980, 206–27).

This method only applies to monophonic music and does not account for harmonic, motivic, or thematic relationships between pitches. However flawed, their method is a consistent approach using a recursive algorithm for hierarchical segmentation. This system is akin to the approach Ruwet called for. Also, its algorithmic approach led to a computerized version capable of quickly processing a melodic segmentation.

Christopher Hasty determines continuity or discontinuity between numerous musical domains in post-tonal music and assesses the strength of a boundary by tallying the number of domains in which continuity occurs (Hasty 1981). He presents the established definition of segmentation as “the division of a musical work into structural components.” He relates this definition to post-tonal music where segmentation entails determining “structurally relevant pitch components or pitch-class sets” (Hasty 1981, 54).

Hasty discusses a two-step method of segmentation. First, listen to the music and note your structural perception. Second, devise rules to form a theory which can account for your perceptions. His paper focuses on the second step of this method.

Some terminological differences exist between Hasty and previous authors. First, he uses the term *domain* instead of *parameter* and defines it as a musical property such as timbre, dynamics, register, or contour. Defining each domain is a stylistic matter inferred from the current musical context. Second, his use of the word *element* is different from that of T & P. Hasty’s *elements* are distinct objects isolated by discontinuity in one or more domain(s). They are not necessarily, as in T & P, the smallest possible unit.

He examines continuity and discontinuity in various domains. A change in the value of a domain creates discontinuity and highlights structurally important elements. Based on this observation, he offers a refined definition of segmentation: “the

formation of boundaries of continuity and discontinuity which result from the structures of various domains” (Hasty 1981, 59). Contextual elements, such as repetition or pitch-class set ordering, can provide sufficient evidence for segmentation. There may be infinite possibilities for the segmentation of a piece, but they can be limited by determining the strength of segmentation possibilities. The more domains exhibiting discontinuity, the more likely it is that there is a strong case for segmentation at that given moment.

Hasty acknowledges that there is a multiplicity of possible answers and that analytical methodology must be combined with aural perception of a passage or piece. This analytical model does not always produce an incontrovertible answer and the level of detail needed to produce a structural analysis makes it difficult to apply to a large section or a whole work.

Jean-Jacques Nattiez introduces ‘paradigmatic analysis’ (Nattiez and Barry 1982). A ‘paradigmatic analysis’ is a method for organizing musical elements into categories (paradigms). Nattiez’s ‘segmentation’ is “the partitioning of the work into units according to abstract paradigmatic axes, that is, axes which group together identical or equivalent units from an explicitly stated point of view” (Nattiez and Barry 1982, 245). His method cannot be replicated for every piece of music. Rather, each piece is a new environment with new rules. Therefore, analysis is a mixture of trial, error, and intuition (Nattiez and Barry 1982, 301).

He focuses on the *neutral level* of the music; that is, the exhaustive description and inventory of all possibly conceivable configurations in a score. He defines two other levels: the *poietic level*, which contains the compositional procedures and intentions of the composer, and the *esthesis level*, which is the analyst’s or listener’s perception of the piece.



Segmentation is an important part of his process. He works from the smallest to the largest units, partitioning the piece according to paradigms which are at the discretion of the analyst and can shift in importance depending on the context. He adopts Ruwet's technique in which relationships are established based on recurrence and repetition. The result is a segmentation at four levels: units, segments, sections, and parts (from smallest to largest). There is no formula for finding structurally significant relationships. Analysis is the result of 'churning' the data exhaustively and determining every possible association. He states: "It is hard to see how a computer could automatically establish an equivalence which depends on a judgment of similarity transcending concrete resemblances and differences" (Nattiez and Barry 1982, 257). As it turns out, this statement becomes an explicit challenge for future theorists.

Fred Lerdahl and Ray Jackendoff's *A Generative Theory of Tonal Music* (GTTM) attempts to describe how a listener intuitively creates an understanding of a complete musical structure (Lerdahl and Jackendoff 1983). Their theory aims to replicate the mental procedures under which a listener constructs an unconscious understanding of music, and uses these tools to illuminate the structure of individual compositions. Lerdahl states: "A listener familiar in a musical idiom organizes its sounds into coherent structures. GTTM attempts to characterize those musical structures that are hierarchical and to establish principles by which the listener arrives at them for a work in the classical idiom" (Lerdahl 2004, 3).

This approach furthers the linguistic aspects of Ruwet and the psychological aspects of T & P. While the authors generate grouping principles based on the gestalt maxim of auditory perception, they also utilize the generative transformational grammar of Noam Chomsky. Generative linguistic theory focuses on unconscious knowl-

edge; the innate knowledge we already have that allows us to learn language (Donnelly 1994, 40).

This literature review will deal explicitly with one of the four “hierarchical structures” outlined by Lerdahl and Jackendoff: *grouping structure*. *Grouping structure* is the listener’s segmentation of a piece. “The listener naturally organizes the sound signals into units such as motives, themes, phrases, periods, theme-groups, sections, and the piece itself. Performers try to breathe (or phrase) between rather than within units” (Lerdahl and Jackendoff 1983, 12). For each hierarchical structure, L & J provide *well-formedness rules* (WFRs) and *preference rules* (PRs). WFRs specify possible structural descriptions. PRs designate, from the structural descriptions, those groupings which correspond to an experienced listeners’ hearings of any particular piece. The list of WFRs and PRs for grouping structure are as follows:

### **Grouping Well-Formedness Rules (GWFRs):**

1. Any contiguous sequence of pitch-events, drum beats, or the like can constitute a group, and only contiguous sequences can constitute a group.
2. A piece constitutes a group.
3. A group may contain smaller groups.
4. If a group  $G_1$  contains part of a group  $G_2$ , it must contain all of  $G_2$ .
5. If a group  $G_1$  contains smaller group  $G_2$ , then  $G_1$  must be exhaustively partitioned into smaller groups.<sup>10</sup>

### **Grouping preference rules (GPRs):**

1. Avoid analyses with very small groups- the smaller, the less preferable.

---

10. Reproduced from Lerdahl and Jackendoff 1983, 37-8.

2. (Proximity) Consider a sequence of four notes  $n_1, n_2, n_3, n_4$ . The transition  $n_2$  -  $n_3$  may be heard as a group boundary if:
  - (a) (slur/rest) The interval of time from the end of  $n_2$  is greater than that from the end of  $n_1$  to the beginning of  $n_2$  and that from the end of  $n_3$  to the beginning of  $n_4$ .
  - (b) (attack/point) The interval of time between the attack points of  $n_2$  and  $n_3$  is greater than that between the attack points of  $n_1$  and  $n_2$  and that between the attack points of  $n_3$  and  $n_4$ .
3. (Change) Consider a sequence of four notes,  $n_1, n_2, n_3, n_4$ . The transition  $n_2$  -  $n_3$  may be heard as a group boundary if marked by:
  - (a) Register
  - (b) Dynamics
  - (c) Articulation
  - (d) Length
4. (Intensification) Where the effects of GPRs 2 and 3 are relatively more pronounced, a larger-level group may be placed.
5. (Symmetry) Prefer grouping analyses that most closely approach the ideal subdivision of groups into two parts of equal length.
6. (Parallelism) Where two or more segments of music can be construed as parallel, they preferably form parallel parts of groups.
7. (Time-span and prolongational stability) Prefer a grouping structure that results in more stable time-span and/or prolongational reductions.<sup>11</sup>

---

11. Reproduced from Lerdahl and Jackendoff 1983, 43-54.

Using these rules to define groupings results in a musical surface which is divided into a sequence of distinct events that combine into hierarchically organized groupings. GTTM reflects an excellent synthesis of previous methodologies as well as an exceptional representation of segmentation methodology presented as clear-cut steps. It projects the mental procedures of a listener as they construct an unconscious understanding of music. The authors admit a few assumptions.<sup>12</sup> First, they assume that the score is the musical surface. Second, they assume that the listener is experienced and, finally, they assume that their theory determines the final state of a listener’s understanding.<sup>13</sup> GTTM is only used with the Classical tonal idiom (Lerdahl 2004, 3). Also, other dimensions of musical structure - notably timbre, dynamics, and motivic-thematic processes - are not hierarchical in nature, and are not treated directly by the theory.

David Lefkowitz and Kristin Taavola also aim to replicate grouping formulated during the listening process. They use concepts of continuity and discontinuity to develop a “musically-responsive weighting” system based on the role of each *dimension’s* effect on perceptual grouping (Lefkowitz and Taavola 2000). They define segmentation as “the process of parsing a composition into meaningful parts” (Lefkowitz and Taavola 2000, 171). Their criteria for segmentation are placed into four *domains* (pitch, timbre, rhythm, and articulation) and subcategories called *dimensions* (or musical parameters). A chart of their domains and dimensions is shown in **Figure 2.18**.

They develop a computational model for how a listener would psychologically parse music into segments. Drawing from gestalt psychology, segments are produced

---

12. These assumptions are paraphrased from Lerdahl 2004, 5.

13. For a discussion of the controversy surrounding the concept of presupposed musical knowledge and structural cognition, see Clarke 1986.

		Domains			
Dimensions	[	<u>Pitch</u>	<u>Timbre</u>	<u>Rhythm</u>	<u>Articulation</u>
		Pitch	Instrumentation	Attack Point	Attack Type
		Pitch Class	Timbral Articulation	Duration	Dynamics
		Interval	Texture	Meter	Expressive Markings
		Interval Class	(Register)	Tempo	
		Register	(Expressive Markings)		
		Contour			

Figure 2.18: Lefkowitz and Taavola’s grouping of musical dimensions into four domains Lefkowitz and Taavola 2000, 176.

when a “change in the rate-of-change” creates discontinuity along any dimension. For example, if a melodic leap occurs in the midst of an overall stepwise melody, that “change in the rate-of-change” may implicate the placement of a boundary. If a boundary conflict occurs between dimensions, they use patterns as a force in resolving or removing the conflict.

For their weighting system, each of the four domains is weighted equally at the start of the analysis. A preliminary evaluation determines the existence of discontinuities within each domain. If a particular domain is entirely absent from the segmental process, it is eliminated from the analysis. The ratio of discontinuities within a particular domain to the total number of events represents an approximation of the likelihood that a particular domain will yield a productive segmentation. This is called the *discontinuity to event ratio* or DTER. The DTERs of each domain are fed into a scaling function to produce quantitative values representing the domains’ relative segmental weight. Then all of the domains’ relative weights are normalized so that they sum to 1.0. This is called the *normalized weighting value* or NWV. The NWV creates a unique representation of the importance of each domain for a particular passage or piece.

The NWV, and the subsequent segmentation, depend on the size of the section

the analyst chooses when computing the weights. It can therefore produce different segmentations for a single passage depending on if the NWV is determined from the composition as a whole or just a single passage. Lefkowitz and Taavola only use the weighting of the four domains to determine segmentation, perhaps losing some of the nuance that could be gained from the weighting of each dimension.

Dora Hanninen, in her 2001 article “Orientations, Criteria, Segments: A General Theory of Segmentation for Music Analysis” and in her subsequent 2012 book *A Theory of Music Analysis: On Segmentation and Associative Criteria*, develops a vocabulary for segmentation that is applicable to all types and textures of music (Hanninen 2001, 2012). She recognizes the role of association in music segmentation and highlights the connection between, and the recurrence of, segments through *associative organization*, determined through the taxonomy described below.<sup>14</sup>

Hanninen introduces many terms and I shall only highlight those necessary to develop a general knowledge of her theory. *Segmentation* is the parsing of the musical surface with the goal of generating musical objects. A *segment* is a grouping of musical events that one deems to be significant in an analytical discussion. There are two types of segments: 1) *genosegments*, which involve only one sonic or contextual criterion (to be defined below) and 2) *phenosegments*, which are readily perceptible musical units that can involve any number or combination of sonic or contextual criteria, with or without coincident structural criteria (see below). Note that Hanninen is drawing her vocabulary from the area of genetics.

A *domain* is the system of all musical knowledge pertaining to the area under consideration. It is delineated by a set of musical phenomena or ideas. The three domains under consideration are: the *sonic domain*, the *contextual domain*, and the

---

14. The following discussion is paraphrased from Hanninen 2012 3-235.

*structural domain*. The *sonic domain* is “the psychoacoustic aspect of music”; That is, the domain that deals with the perception of sound and the sensations produced by sound (Bosi and Goldberg 2003, 149). The *contextual domain* recognizes the role of repetition, association, and categorization in music. The *structural domain* is shaped based on reference to a theory of musical structure or syntax which may be chosen or developed by the analyst. For example, the analyst could use twelve-tone, Schenkerian or Neo-Riemannian techniques to determine a recommended segmentation.

The analysis of each domain is approached by concentrating on each of three different *orientations*. An *orientation* is a method of conceptualizing music. They are differing points of reference from which analysts can adjust their analytical process. The three orientations are 1) *disjunctive orientation*, 2) *associative orientation*, and 3) *theory orientation*.

In a *disjunctive orientation*, boundaries are found by looking for differences between musical events. The greater the difference is between events, the stronger the disjunction is between two musical units. The stronger the disjunction, the stronger the implied boundary. A disjunction may be determined by a sudden change in register, duration, dynamics, or timbre.

*Association* is the act of looking for relationships between groupings of musical events. Association focuses on relationships supported by repetition, equivalence, or similarity. In an associative orientation, one locates points of correspondence between segments.

*Theory orientation* is the interpretation of musical grouping based on an orienting theory (again, such as Schenkerian theory, etc.). A theory is a coherent system of rules or principles, commonly regarded as correct, that can be used as principles for explanation and prediction for a class of phenomena. In music, a theory is an orderly

system of concepts and rules that guides or governs the recognition, interpretation, and organization of significant musical units. The concepts and rules of such theories provide means to select segments which the orienting theory deems well-formed and to have potential to be structurally significant.

For each orientation, the analyst must determine a rationale for musical segmentation, or *criteria*. There are three basic types of criteria, which mirror the three domain types: 1) *sonic criteria*, 2) *contextual criteria*, and 3) *structural criteria*. *Sonic criteria* are sounds and silence. Examples of sonic criteria are pitch, attack-point, duration, dynamics, timbre, and articulation. Hanninen highlights three “linear sonic dimensions:” pitch, duration, and dynamics. Segmentation using these linear criteria is a matter of calculation and not interpretation, which has made them attractive candidates for computer modeling. *Contextual criteria* define segments and imply boundaries. They determine the rationale for segmentation based on repetition, equivalence, or similarity between two or more groupings. Examples of contextual criteria are pitch contour, pitch content, pitch-class set, set class, scale degree ordering, and rhythm. *Structural criteria* indicate a segmentation supported by a specific orienting theory.

The result is a nesting of analytical techniques and a bottom-up approach to music analysis which determines important musical units from smallest to largest. These units are built by supporting the criteria through *instantiation*, *coincidence*, and *realization*. A single criterion that delineates a segment is an instantiation. Two or more criteria that identify the same segment are called coincident. Realization is a type of coincidence in which one criterion is structural and the other contextual, generating a stronger argument for segmentation.

Hanninen’s process is highly context-sensitive. Phenosegment formation is subject



to interpretation. But there is a positive correlation between the number of coincident genosegments and phenosegment strength. A segmentation is *complete* if it includes every note in the passage under discussion. A *clear* segmentation is one comprised solely of disjunct phenosegments. Segmentations become progressively ambiguous as the number of phenosegments with boundaries that overlap increases. Segmentations which are complete and clear are stronger: that is, when phenosegments are disjunct, temporally adjacent, registrally contiguous, and account for nearly every note.

To summarize, Nicholas Ruwet develops a method for melodic segmentation utilizing a linear algorithmic process which identifies, one non-parametric variable at a time, recurrence, repetition, and transformation of the chosen variable, highlighting possible segments. Allen Forte uses the musical context and knowledge of a given composer's style to determine primary segments and composite segments in a linear manner. James Tenney and Larry Polansky present an algorithmic computer process based on gestalt psychology in which the similarity and proximity of musical parameters create a bottom-up segmentation for monophonic music. Christopher Hasty uses continuity and discontinuity within musical domains, determined from the musical context, to develop a domain weighting system for the segmentation of post-tonal music. His theory is a combination of analytical theory and aural perception.

Jean-Jacques Nattiez introduces paradigmatic analysis in which the neutral level of the musical surface is divided from smallest to largest segments, adopting Ruwet's methodology of recurrence and repetition to determine segmentation. His procedure is contextual and every piece is a new environment. The process is meticulous, determining every possible relationship between paradigms. Fred Lerdahl and Ray Jackendoff combine the psychological ideals of gestalt theory with Noam Chomsky's

generative transformational grammar. Their grouping structure is meant to replicate how a person mentally divides music. They create a rule-system for grouping units based on a listener's natural tendency to mentally organize.

David Lefkowitz and Kristen Taavola also replicate the grouping process as formulated by the listener. Like Hasty, they use continuity and discontinuity to develop a calculus for segmentation and a musically-responsive weighting system based on gestalt psychology. They determine the change in the "rate-of-change" within domains and dimensions to determine boundaries. Finally, Dora Hanninen develops a new vocabulary for segmentation. Her theory is applicable to all types and textures of music. Through repetition, association, and psychoacoustics, each viewed through the lenses of three different orientations, she determines disjunction which, in turn, determines segment boundaries. Her theory is a bottom-up, context-sensitive approach.

The threads binding these theories together are many. There is an attempt to create a methodology for segmentation, which is generally algorithmic in nature. Segmentation is a bottom-up, linear process which creates the smallest units first, and then seeks out relationships between those units to create higher levels of segmentation. These theories are often based on linguistic and psychological processes, which is often the case in determining the language of musical analysis. Finally, while all strive for a methodology, they all admit that segmentation is a highly context-sensitive process of trial and error, in which analysis of the various parameters of a musical surface must be paired with the listener's aural interpretation.

Based on the research listed herein by multiple theorists, I offer a definition of segmentation. Segmentation is the procedure for determining how a musical work is divided into structurally significant musical units. To determine a segmentation, the

analyst must first listen to the work or passage and determine any aurally perceivable boundaries. Then the analyst may use parameters, based on the context of the musical surface and knowledge of the composer's style, to determine continuities such as recurrence, repetition, transformation, etc. The presence of discontinuity (changes in register, timbre, dynamics, pc-set, etc.) highlights locations for possible boundaries. The more parameters displaying discontinuity, the greater the argument is for boundary placement.

While this definition serves as a universal description of segmentation, it offers no additional information as to how to develop a method of segmentation for post-tonal music. A piece of post-tonal music may have no aurally perceivable boundaries. Additionally, a typical surface analysis, akin to that for tonal music, does not reveal the necessary continuities/discontinuities for determining formal units. While many of the methods in this section are feasible solutions to the question "How do we begin a post-tonal analysis?" the author has chosen Tenney and Polansky's temporal gestalt segmentation algorithm because it is already well-established as a methodology and because it is based on the psychological perception of the listener. Their algorithm will be paired with contour analysis and pitch-class set analysis in order to verify boundary locations.

## 2.3 Computer-Assisted Analysis

This section will present a summary of currently available computer-assisted analysis tools. Since the 1960s, computing has been used for any of the following musical processes:

1. Composition
2. Analysis
3. Emulating tuning systems
4. Searching, browsing, and retrieving musical information
5. Segmentation
6. Spectral analysis
7. Improvisation
8. Creation and grading of classroom materials

A detailed historical account of computer assisted analysis through the year 2000 has been completed by Nico Schöler.<sup>15</sup> David Cope continued the cataloging of historical computation analysis through 2008.<sup>16</sup> Therefore, this section will focus on music analysis computer programs currently available at the time of publication of this dissertation.

### **Melisma (Modular Event-List-Input System for Music Analysis) Music Analyzer**

In 2001, David Temperley and Daniel Sleater created the Melisma Music Analyzer. It extracts meter, voicings, phrases, key, and chord function from tonal music (Cope

---

15. See Schler 2000.

16. See Cope 2008.

2008, 36).<sup>17</sup> The authors describe it as a computational system for music analysis that identifies metrical, harmonic, and stream information (Temperley and Sleater 2013). A stream is akin to a contrapuntal line. Melisma also estimates the probability of note patterns. The original version consisted of separate modules that could be detached or added as needed during analysis (Cope 2008, 36). The modules were:

1. The Meter Program: produces a framework of rows of beats from a list of notes.
2. The Grouper Program: groups notes of a melody into phrases.
3. The Stream Program (a.k.a. the counterpoint program): groups notes into contrapuntal lines or “streams.”
4. The Harmony Program: produces a harmonic analysis of the piece labeled with roots.
5. The Key Program: takes a list of notes and beats and produces a key analysis, dividing the piece into sections labeled with key names. It can also take harmonic information and produce a “Roman numeral analysis,” showing each chord’s position within a key.<sup>18</sup>

The more recent version (2009) is a single integrated computer program. User input is in the form of a MIDI or piano-roll representation.<sup>19</sup> The model then derives three kinds of musical structure: metrical structure, harmonic structure, and stream structure (Temperley 2009, 4).

---

17. Version 1.0 [www.link.cs.cmu.edu/music-analysis/](http://www.link.cs.cmu.edu/music-analysis/). Version 2.0 <http://theory.esm.rochester.edu/temperley/melisma2/>

18. Names and descriptions of each module are paraphrased from Temperley and Sleator’s Version 1.0 website listed above.

19. A ‘piano-roll’ representation is a list of notes indicating the on-time and off-time (in milliseconds) and pitch of each note (Temperley 2009, 4).

## RUBATO

RUBATO is a music software environment developed at the Department of Informatics, University of Zürich, under the direction of Guerino Mazzola.<sup>20</sup> Just like version 1.0 of Melisma, it is modular. RUBATO analyzes, composes, and performs music. The modular architecture allows third-party developers to create their own modules (Cope 2008, 36). It provides convenient mathematical tools which aid in music composition.

G  rard Milmeister developed a module called Rubato Composer, which is an open source Java 1.5 application for music composition.<sup>21</sup> It follows the RUBATO software architecture and consists of a main application and a number of plug-in modules, named rubettes. In a graphical user interface, the rubettes can be connected and run by pressing a button. It is only a tool for composition and requires a theoretical mathematical background to operate.

## Tonalities

The Tonalities computer program, created by Anthony Pople at the University of Nottingham, was designed for analysis of Western tonal music of the late nineteenth and early twentieth centuries (Pople 2002, 1).<sup>22</sup> Tonalities allows users to analyze passages of music in terms of differing keys that may be detailed from a range of supplied options. It is an add-on to Microsoft Excel spreadsheet software (Cope 2008, 36).

User input is necessary. The user enters pitch and segmentation data. Analysis occurs in the form of a window output which provides prolongation, function, figured

---

20. <http://www.rubato.org/>

21. For a complete description, see Mazzola et al. 2008, and Milmeister 2009.

22. <http://www.hud.ac.uk/tonalities/>

bass and pitch-class content for each segment. The user may then scroll through the segments to review the analysis (Pople 2002, 4).

### **Harmonia and MTW (Music Theory Workbench)**

MTW was developed by Heinrich Tauber at the University of Illinois. It is designed to analyze short tonal works and the output is an annotated graphic score. Analysis includes chord and inversion classification, nonharmonic tone determinations, primary and secondary tonal center identification, and a functional harmonic analysis. Music information is parsed into a time line of vertical sonorities, like in a harmonic reduction, that can be identified as triads or seventh chords. These sonorities are then classified by type (major, minor, diminished, augmented, seventh chord, etc.) and inversion. Sonorities that are not classifiable as chords are subjected to nonharmonic tonal analysis, with the resulting nonharmonic tone(s) identified by their type (passing, neighbor, suspension, etc.) (Cope 2008, 36-7).

MTW code is used in the implementation of the music theory application *Harmonia*.<sup>23</sup> *Harmonia* combines music notation, automatic music analysis and grading, word processing, and multimedia playback. The goal of *Harmonia* is to replace paper-based music theory (textbooks, workbooks, handouts, homework, and tests) with enriched portable document format (PDF) documents that allow music content to be created, edited, searched, annotated, automatically analyzed, and automatically graded (Taube and Burnson 2013).

---

23. <http://camil.music.illinois.edu/software/harmonia/>

## Music21

Music21 is a Python-based, object-oriented toolkit for analyzing, searching, and transforming music into symbolic forms (such as a musical score) (Cuthbert and Ariza 2010, 637).<sup>24</sup> According to the authors, “applications of this toolkit include computational musicology, music [information], musical example extraction and generation, music notation editing and scripting, and a wide variety of approaches to composition, both algorithmic and directly specified” (Cuthbert and Ariza 2010). Music21 supports the input of most musical data formats, including MIDI, MusicXML, Humdrum, and Lilypond. Because Python is a well-established and easy computer programming language, Music21 is an approachable and more user-friendly application for those who are not computationally inclined. Just a few of its useful analytical tools are:

1. Neo-Riemannian analysis
2. Harmonic reduction
3. Contour finder
4. Creation and grading of counterpoint examples
5. Chord realization and voice-leading analysis

## VisiMus

VisiMus is currently developed by the Genos research group.<sup>25</sup> Genos is a multi-disciplinary research group in music theory, composition, and computer music focused on development and use of free software (Genos 2013). Marcos da Silva Sampaio and Pedro Kröger developed a web-based computer application called MusiContour which

---

24. <http://web.mit.edu/music21/>

25. <http://genosmus.com/>



is now part of the VisiMus application.<sup>26</sup> It is a software application for calculating and plotting musical contour relation operations.<sup>27</sup> VisiMus can process contours related to different parameters such as pitch, duration, dynamic level, and chord density. The application calculates various contour properties as defined in the publications of Michael Friedmann and Elizabeth Marvin West and Paul Laprade.

These computer programs offer an assortment of analytical tools, most of which are user-friendly. However, the majority focus on the analysis of tonal music. Music21 and VisiMus offer contour analysis options. Music21 can only locate contours, and does not offer the capability of contour reduction and further analysis or comparison. VisiMus offers tools for more in-depth contour analysis, but does not offer an option to use the updated contour reduction algorithm of Rob Schultz. Input must be performed by hand, as the computer program does not allow for file upload. Visimus only allows for mod12 input of integers 0–11. Finally, none of these computer programs offer the ability to compute the diachronic transformational analysis described by Rob Schultz.

This dissertation introduces a new application for musical contour analysis. This application performs Rob Schultz’s version of the contour reduction algorithm. It allows users to upload a MIDI-file, saving the time needed to input pitch data by hand. It also allows users to input data in any means they wish: mod12, MIDI number, pitch frequency or any numerical system chosen by the analyst. Finally, it offers the ability to compute a diachronic transformational analysis of an entire piece of monophonic music.

---

26. See Sampaio and Krger 2009.

27. <http://visimus.com/>

# Chapter 3

## Computer Programming Process

This chapter explains *coding* procedures. Coding is the process of writing logical statements in a computer programming language using human-readable computer instructions called *source code* (code) (Freedman, accessed July 15 2014, “source code”). This chapter is also meant to be an introductory text for any music analyst wishing to write computer programs for research and analysis, specifically using Java programming language. Refer to the glossary as necessary for definitions of terminology used herein.

Computer programming (programming) is the process of writing a computer program (program) (Hoare 1969, 2). The purpose of programming is to find a sequence of instructions that will automatically perform a task or solve a problem. Quality code should be reliable, robust (that is, capable of performing without failure under a wide range of conditions), usable, portable, maintainable, and efficient (Spinellis 2006, 1-7). The process of coding includes the following steps:

1. Understanding the requirements for solving a problem resulting in an algorithm.
2. Verification of the requirements of the algorithm including correctness and its resource consumption.<sup>1</sup>

---

1. Examples of resource consumption are memory use and CPU time (Nakov 2013, 363). CPU

3. Implementing (or coding) of the algorithm in a target programming language.
4. Testing, debugging and maintaining the source code.
5. Implementation of the build system.<sup>2</sup>
6. Management of derived artifacts such as source code and providing user support.<sup>3</sup>

The initial problem statement for this research was: how can Robert Morris’s CRA be automated? While the results of contour reduction are useful for determining a number of analytical qualities about a melody, when executed by hand the process is tedious and error-prone. The repetitive, rule-driven algorithm lends itself to computer implementation.

While Robert Morris’s CRA was the impetus, this dissertation focuses on automating Rob Schultz’s amended version of the CRA. Although the bulk of the discussion focuses on Schultz’s work, the code for Morris’s algorithm is available in **Appendix A**.

A *computer programming language* (language) is a formal vocabulary designed to communicate a sequence of instructions to a computer (Nakov, 71). A number of languages could have been used for this project: Python, Ruby, Java, C++, etc. *Java* language was chosen because it is widely used and the resultant program is accessible on a wide variety of electronic devices, operating systems, and Internet browsers. Java is a simple language. A novice Java programmer does not need extensive programmer

---

stands for central processing unit and is the hardware within a computer that carries out the instructions of the program by performing the basic arithmetical, logical, and input/output operations of the system.

2. A build system is a set of program tools designed to automate the process of program compilation. Its primary goal is to efficiently create an executable.

3. Paraphrased from Nakov 2013, 31, 76-7.

training, which is ideal for a musicologist/theorist wishing to bridge the gap between music and computer science (Gosling and McGilton 1995).

Java is an *object-oriented* language. In object-oriented programming the priority is data and using data (Horstmann and Cornell 2013, section 1.2.2). Objects are defined in data types called *classes* (Horstmann and Cornell 2013, section 4.1.0). A class is the document in which programmers store all of their *methods* (their objects) (Horstmann and Cornell 2013, section 4.1.1). A method is the process that the object performs (Horstmann and Cornell 2013, section 3.1.0). It is best to think of classes as nouns and methods as verbs (Horstmann and Cornell 2013, section 4.1.3).

The final result of the coding process specific to the research in this dissertation is a *World Wide Web-based application* (web application) for analyzing musical contour. A web application is any application that uses a Web browser as a *client* (Freedman, accessed July 15 2014, “web application”). A client is the program that an end user (user) interacts with (Freedman, accessed July 15 2014, “client”). An (user) is the target individual for a program (Freedman, accessed July 15 2014, “user”). The application is named: Contour Analysis Tools (CAT). CAT is available for use at [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). A screen-shot of the World Wide Web-based application’s main webpage is shown in **Figure 3.1**.

The screenshot shows the CAT web application interface. At the top, a dark blue header contains the title "Contour Analysis Tools" and the subtitle "and other tools for music analysis". To the right of the header are links for "Home", "About", and "Contact", and social media links for "Facebook", "YouTube", "UConn Music", and "USAO". Below the header is a navigation bar with four tabs: "Contour", "More Tools", "More Tools", and "More Tools". The "Contour" tab is selected. The main content area is titled "Contour" and contains the instruction "Input pitches as integers or decimals (i.e. 7 2 7 2 7 2 7 11 14)". Below this is a form with a label "Enter Pitches:" followed by a text input field and a "Submit" button. Underneath the input field is a large empty rectangular box. Below this box is a file upload section with a "Choose File" button, the text "No file chosen", and an "Upload File" button. Below the file upload section is another large empty rectangular box. Below this box is the instruction "Copy and paste upload results into box labeled 'Enter pitches.'". At the bottom of the main content area is a paragraph: "The application will return the contour reduction according to Rob Schultz Contour Reduction Algorithm, depth, prime and normal forms, set class info and the contour prime". The footer of the page is dark blue and contains the copyright notice "© Copyright 2014 - CooleySekula | Website Template By Matthias Le Brun".

Figure 3.1: CAT web application.

### 3.1 A Short Coding Example

A brief example of code will be used to explain some common syntax of programming language. It should be explained that pitches are converted to a numerical value to make them readable by the program. This is common practice in post-tonal musical analysis in which modular arithmetic is used to convert all pitches to integers from 0 to 11, with 0 equivalent to the pitch-class C and 11 equivalent to the pitch-class B.

To accurately measure contour, representation for octave determination is necessary. For this dissertation, the labeling system  $C4 = 0$  is adopted because this is typically the lowest note of the transverse flute. Therefore, the integers -1 through 39 are used to represent the full range of the flute; -1, for flutes with a B-foot (B3), through  $D\sharp 7$ . CAT goes beyond integer notation to allow decimal values. This means that quarter-tones (or any other microtone) are easily accounted for. A user could use any numeric value for pitches, such as MIDI number (middle C = 60) or frequency in hertz (middle C = 262.262 Hz).

```
ArrayList<Double> pitches = new ArrayList();

pitches.add(0);
pitches.add(12);
pitches.add(24);

int i = 0;

for ( i = 0 ; i < pitches.size() ; i++) {
    if (pitches.get(i) >= 0 && pitches.get(i) < 12)
    {}
    else
    {pitches.set(i, pitches.get(i) % 12);}
}
```

Figure 3.2: Example of source code for determining mod 12.

**Figure 3.2** displays a small portion of code that calculates a mod 12 pitch assignment for any numerical pitch value. The first line creates an *ArrayList* named ‘pitches.’ An *array* is a data structure that stores data elements of the same type (integer, double, string, etc.) with a set size in the order in which they were entered (Horstmann and Cornell 2013, section 3.10.0). An *ArrayList* is an array with a capacity that automatically adjusts as data elements are added (Horstmann and Cornell, section 5.3.0).

Numerical values representing C4, C5 and C6 are added to ‘pitches.’ The com-

mand ‘`pitches.add(0)`’ adds the pitch C4, represented by the integer 0, to the ArrayList. The next two lines of code add C5 (12) and C6 (24), respectively. If printed by the program, the ArrayList ‘`pitches`,’ with these additions, would display as [0, 12, 24].

Creating successful code requires the formulation of a logical process for the task that is to be automated. A specific procedure for determining mod 12 must be developed. The result is the verbal algorithm shown in **Figure 3.3**.

1. Convert pitches to integers.
2. Examine the first integer in the list.
3. If this integer is a value between 0 and 11, do nothing. It is already mod 12.
4. If this integer is not a value between 0 and 11, then divide it by 12. The remainder is its mod 12 value.
5. If all integers have been acted upon, go to step 7. If not, move to the next integer.
6. Go to step 3.
7. End.

Figure 3.3: Verbal algorithm for determining mod 12 of a pitch represented as an integer.

Determining logical steps for task completion makes assigning the code to automate the task more apparent. Step 1 of **Figure 3.3** is complete. C4, C5, and C6 were converted to the integers 0, 12 and 24 prior to being added to the ArrayList ‘`pitches`.’

Step 2 is completed using a process called a *for loop*. A *loop* is the main programming construct. It allows repeated execution of a fragment of code (Nakov 2013, 211). Verbally, a for loop means: “For the given data. . .” The statement, specific to the algorithm in **Figure 3.3**, is: “For the ArrayList called ‘`pitches`’ . . . ” A for loop has

three possible parenthetical statements which communicate the following information to the program: 1) the code to execute before the loop begins, 2) the condition for running/continuing the loop and, 3) the code executed after the loop is complete.

The code in **Figure 3.2** reflects all the characteristics of a for loop as described above. The *variable* (a structure that holds data) *i* is created by the statement `int i = 0`, where ‘int’ is short for ‘integer’ and 0 is the initial value of *i* (Nakov 2013, 111). This variable acts as a place-holder and communicates to the program which element in ‘pitches’ to currently act upon. The first line of code states:

```
for (i = 0 ; i < pitches.size() ; i++)
```

The information inside the parentheses initiates the following logical commands, which correspond with the three parenthetical statements defined above:

1. Start at the first position in the ArrayList ‘pitches.’ The integer variable *i* = 0. This integer is a placeholder and 0 represents the first data element in the ArrayList.
2. Once a given operation has been performed on the current data element, move to the next element. This is represented by the code `i++`. `++` is a symbol that means “move to the next element.”
3. Continue this process until the program progresses through the length of the ArrayList. This is represented by the code `i < pitches.size()`. The program determines how many elements, *n*, are in the ArrayList and continues to iterate through the elements until *n*−1 has been reached, since the first place in an ArrayList is zero.

What follows between the curly braces { } (see **Figure 3.2**) will be the operation to be performed on each element.



Inside of the for loop there is a process called an *if...else* statement. Verbally, an *if...else* statement means: “If such-and-such an operation has a certain outcome, then the program should do something *or else* do something different.” The next portion of code in **Figure 3.2** states:

```
if (pitches.get(i) >= 0 && pitches.get(i) < 12)
```

Verbally, this code means: “If the current element we are examining is an integer that is greater than or equal to (represented by the symbol  $\geq$ ) 0 *and* if the current element we are examining is an integer that is less than (represented by the symbol  $<$ ) 12...” The  $\&\&$  is a boolean statement representing that both statements on either side of  $\&\&$  must be true. The next portion of code communicates to the program what to do if both characteristics are met in the *if*-portion of the statement. In this example, the code shows only empty curly braces `{ }`. This means “do nothing.” If the examined integer is already a number between 0 and 11 it is already mod 12 and no action must be taken.

The next lines, the *else*-portion of the statement, communicates to the program what task to perform if both characteristics are not met in the *if*-portion of the statement:

```
{pitches.set(i, pitches.get(i) % 12);}
```

If the integer currently being examined is not a number between 0 and 11, then change the element (`pitches.set`) at that position in the `ArrayList` to the value of the current pitch mod 12 (`pitches.get(i) % 12`). The operation `% 12` is a pre-coded method that determines mod 12. `% 12` is part of the Java’s mathematical library. Libraries will be further discussed in **section 3.2.3**. The code inside the *if...else* statement completes step 3 of **Figure 3.2**. The for loop proceeds to the next integer (`i++`) and replicates the process until the end of the `ArrayList` is reached, completing steps 5 and 6.

In summary, a small section of code was written. An ArrayList named ‘pitches’ was created and integers representing pitches were added to that ArrayList. Using a for loop, every integer was checked to determine if it was already mod 12. If not, an operation was performed on that integer to make it mod 12 and to replace the old integer value with the new mod 12 value. This example has demonstrated the basic operations of computer programming language as listed in **Figure 3.4**. For the current dissertation, this operation equates to the list shown in **Figure 3.5**.

1. Determine the question to answer.
2. Determine the format of the end result.
3. Write a logical text-version of an algorithm to answer the question.
4. Write code that reflects the logic of the verbal algorithm.

Figure 3.4: Basic operation of computer programming language.

1. How can the process of contour reduction be automated?
2. The result should be a contour segment of pruned contour pitches represented by integers.
3. The logical text-version of the algorithm has been completed by both Robert Morris and Rob Schultz.
4. Code was written to complete this task and is listed in **Appendices A through H**.

Figure 3.5: Basic operation of computer programming language as applied in this dissertation.

While this coding example has been short, it represents the basic tenants of programming. Once a novice coder can understand and utilize these concepts, it is not difficult to branch out to more complex coding.

## 3.2 Computer Program Structure

CAT is launched as a *web application*. A web application uses a *Web browser* as a means of implementation and dispersement (Freedman, accessed July 15 2014, “web application”). A Web browser is the application program that serves as the primary method for accessing the World Wide Web (Freedman, accessed July 15 2014, “web browser”). The web application CAT has several components. These are listed in **Figure 3.6**.

1. Java class files which contain code and implement every process associated with the web application.
2. A jsp that determines how the web application looks and acts.
3. Servlet files which connect the user’s input with the Java classes on the server.
4. Libraries of pre-written code.
5. Configuration files.

Figure 3.6: Components of the web application CAT.

The flow-chart in **Figure 3.7** illustrates the structure of the files within the application. The top node is the IDE. This is the program used to write, *debug*, *compile*, and launch the application (Nakov 2013, 93-4). Debugging is the process of methodically finding and repairing defects in a program (Nakov 2013, 101). Compiling is the act of transforming code written in a language into another language, most commonly to create an *executable* program (Horstmann and Cornell 2013, section 1.2.6). An executable is a file that causes a computer to perform indicated tasks according to written code (Freedman, accessed July 15 2014, “executable”).

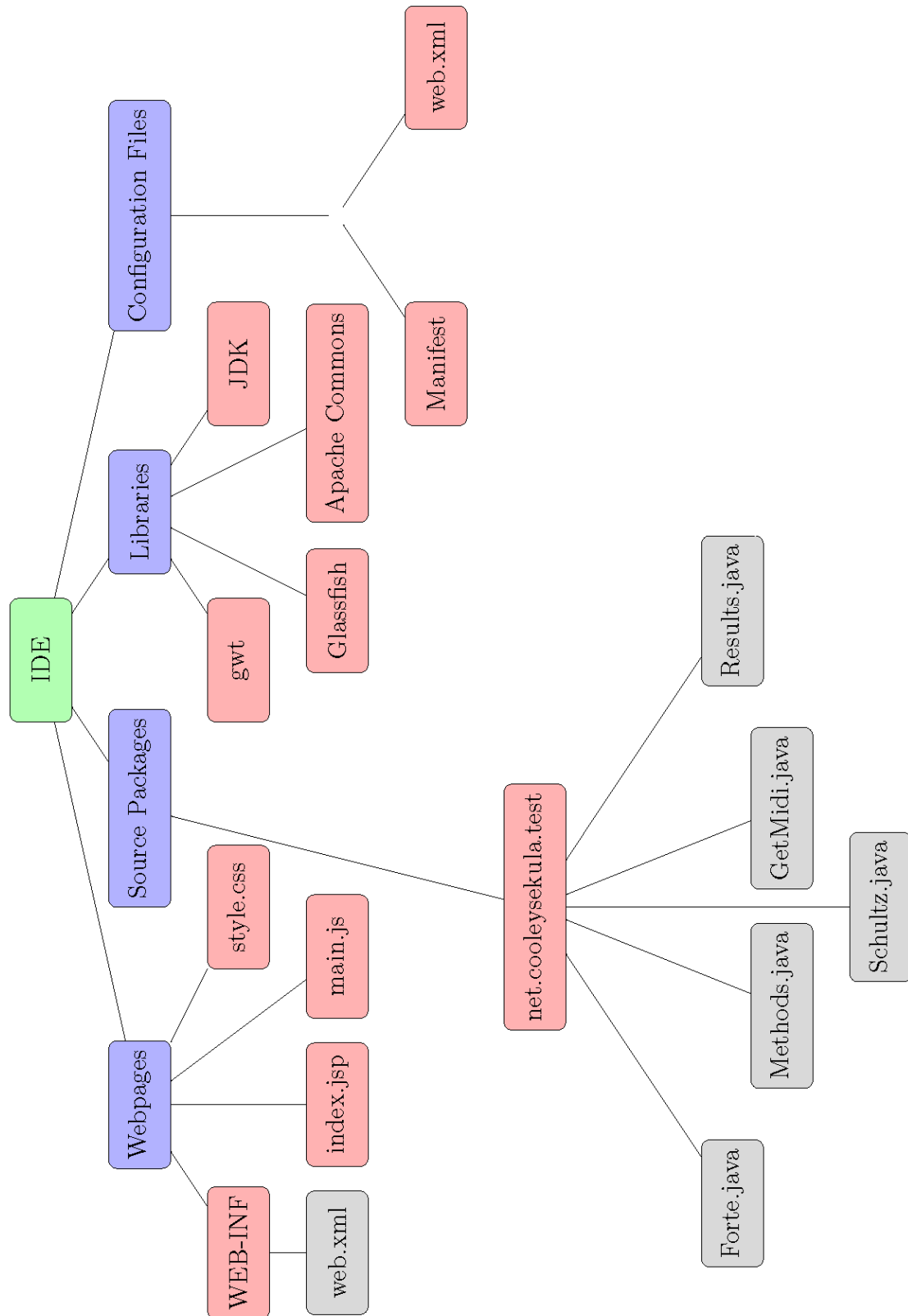


Figure 3.7: Flowchart of web application structure.

For this project the IDE Netbeans was used. Netbeans is an open source professional development environment (Horstmann and Cornell 2013, section 2.2). A program that is open source is distributed with its source code so that anyone can modify it for their own purposes (Freedman, accessed July 15 2014, “open source”). Netbeans provides support for many languages, including Java, and is itself written using Java language. It allows for a programming experience similar to working with a word processing program, offering tools, templates, and drag-and-drop capabilities for design.

### 3.2.1 Web Pages

The first node below the IDE in **Figure 3.7** shows the various files used to create the web pages that a user interacts with. The webpages folder contains the resources for creating the graphical user interface (GUI) which displays on the client side of the web application. A GUI is a program that allows a user to easily interact with a computer by creating clickable pictures (Nakov 2013, 167).

The file style.css was written using cascading style sheet (CSS) language. A CSS is used to determine the look and formatting of a document written in HyperText Markup Language (HTML), which is the standard document format for web pages (Freedman, accessed July 15 2014, “cascading style sheets”). There are numerous websites with pre-designed open source CSS templates.

The file main.js contains code for a javascript function (js is short for javascript) which is called by the file style.css and determines the look and formatting of the clickable tabs used to change between analysis programs in the GUI. To *call* something means to request services from another program (Freedman, accessed July 15 2014,

“call”). The tabs created in main.js can be seen in **Figure 3.1**. In the future, these tabs will house other analysis tools.

The file named index.jsp is the central file containing all information about how the web application should look and perform. A JavaServer Page (jsp) file combines javascript and HTML into one document (Jendrock et al. 2014, 25). The jsp communicates to the program how it should act and the HTML communicates to the program how it should look. It is the jsp file that first loads when a user navigates to the web page using a Web browser. The code for all three of these files is listed in **Appendices B, C and D**.

The folder WEB-INF is also located under the webpages node. WEB-INF is short for “web information.” It contains a file called web.xml. Extensible markup language (xml) is a language used to annotate documents (such as a web page) and defines rules for how to *encode* said document (Jendrock et al. 2014, 16). Encoding is the act of assigning code to represent data (Freedman, accessed July 15 2014, “encode”). The web.xml file for this application lists all of the servlets (to be discussed below) used by the web page. It also specifies which document is the first to be displayed when the web address is used, in this case the file index.jsp.

### 3.2.2 Java Source Packages

*Java packages* (packages) are the mechanism for organizing Java classes (Gosling and McGilton 1995, 46). Packages contain the code needed to run an application. The name of the package reflects the location on the *server* where the files for running the web application are stored. A server is a computer system in a network that is shared by multiple users (Freedman, accessed July 15 2014, “server”). A network is

a system that transmits data between users. It includes the network operating system in the client and server machines, the cables connecting them and all supporting hardware in between. In wireless systems, antennas and towers are also part of the network (Freedman, accessed July 15 2014, “network”). The files for CAT are kept on a server called cooleysekula, therefore the name of the package reflects this server name: net.cooleysekula.CAT. Classes and servlets (to be discussed below) needed to run the web application are stored within this Java package.

### **Methods.class**

CAT contains a class called Methods.class. Contained in this class is code for every method needed to run CAT. What follows is a an explanation of several of these methods. The code for Methods.class is available in **Appendix E**. Unless otherwise noted, all future discussion about the Methods.class will refer to this appendix.

### **Method: getMod12()**

This method contains code that determines the mod 12 of any pitch. It was examined in **Figure 3.2** but some additional coding characteristics warrant discussion. The name of the method is getMod12(). The capitalization is standard procedure for Java programming. The “get” is lowercase and the actual object “of the getting” is uppercase. Note the commenting process. Code should be highly commented. The author of the code includes a description of the intent of each line, so that when they return to it later, or if someone else uses it, they know exactly what is meant for each line. In Java, comments are marked with a // sign. This sign communicates to the program that text after the // is not code and should not be acted upon.

Examine the first line of code as shown in **Appendix E**:

```
public static ArrayList getMod12(ArrayList list)
```

The word *public* is an *access modifier*. Access modifiers control the amount of access other parts of the package have to the method. The modifier *public* declares that all other parts of the program can access this method (as apposed to *private*, which indicates that the method is only available to the current class) (Gosling and McGilton 1995, 46). The method `getMod12()` is *public* so that the program can call it using servlets (to be explained below). *Static* is a modifier that states that there is only one such object of this name in a class (Gosling and McGilton 1995, 45-6). In the class `Methods`, there is only one object called `getMod12()`.

The text that follows the modifiers communicates to the program what type of object is expected as the final product; in this case, an `ArrayList`. Then the name of the method is declared: `getMod12`. The parenthetical statement `(ArrayList list)` communicates to the program that the initial user input will be in the form of an `ArrayList` and it will be named ‘list.’

Next is the element called a *double*. It first occurs in the line:

```
public static ArrayList getMod12(ArrayList list).ArrayList<Double> pitches = new  
    ArrayList()
```

A double is a *floating type* (float). A float is used to define numbers with fractional parts, i.e. decimal numbers. A regular float has a precision of up to 7 digits. A double is more precise and can accurately handle up to 15 digits (Horstmann and Cornell 2013, section 3.3.2). Doubles, rather than integers, are used in this program so that operations may be performed using microtones. Since the program would rarely need to analyze a long (as in thousands) list of data elements, the choice of double minimally effects the speed of the program. In this line of code, an `ArrayList` named ‘pitches’ is created and the program expects that Doubles will be added to



that list.

Finally, there is a statement after the for loop ends that states: `return data`. This is called a *return* statement. It communicates the output of the method to the program. In this example, when the method `getMod12()` has been completed, the `ArrayList` named ‘data’ will be returned.

### Method: `getNormal()`

This method determines the *normal form* of a pcset. Normal form is an ordering of pitches within the SC that spans the smallest possible interval (Babbitt 1961, 77). It is the most compact and representative of all pcsets in the same SC (Morris 1991, 39). The verbal algorithm for determining normal form is shown in **Figure 3.8**.

1. Place pcs in numerical order from lowest to highest.
2. List all rotations of the pcset.
3. Determine which rotation has the minimum distance between the first and last element in the set.
4. If there is a tie, choose the rotation which has a minimum distance between the first and second numbers.
5. If there is still a tie, then check the first and third numbers, and so on until the tie is resolved.

Figure 3.8: Verbal algorithm for determining normal form.

The code for determining normal form uses looping techniques, but on a larger scale than the method `getMod12()`. There are several declarations that warrant explanation. The first is: *HashMultiMap*. A *map* is an object that assigns a relationship between two items. In Java these items are called *keys* and *values*. Keys with associated values are added to the map. Keys in the map can then be recalled using values or vice versa (Horstmann and Cornell 2013, section 13.2.8). It is helpful to visualize

a map as a two-column table where keys are placed in the first column and values in the second column. A *HashMap* is a map that is easily reviewed for information. It does not allow duplicate key/value pairs and does not keep elements in order. The process of hashing searches through key/value pairs without having to write code to iterate through all values as in an *ArrayList* (Horstmann and Cornell 2013, section 13.2.8). A *HashMultiMap* is a *HashMap* that can have multiple values for any key.

This method uses an operation called a *while* statement. Verbally, a while statement means: “While a certain property holds true, continue doing some predetermined operation.” This kind of statement is first used in relation to a ‘counter,’ set up in the preamble to the code. The counter, named ‘count,’ counts the number of times the code within the while statement is performed. The program repeats the code between the curly braces { } until ‘count’ is equal to the number of elements in the *ArrayList* ‘data.’ Once those two values are equivalent, every rotation of ‘data’ has been determined.

Finally, this method uses an operation called a *do...while* loop. Verbally, a do...while loop means: “Repeat an operation while a certain property holds true.” In this case, the property is a boolean variable called ‘done.’ As long as the value ‘done’ = ‘false’ then the operation within the do...while statement will continue. There are a series of for loops and if...else statements that result in the variable ‘done’ having its value changed to ‘true.’ As soon as this happens, the program breaks out of the do...while loop and continues to the next portion of code.

Note that the method `getNormal()` makes a call to another method within the `Method.class` file: `rotateleft()`. The method `getNormal()` is able to access this method because the access modifier `public` was included to make it accessible from all areas of the package.

### Method: `getPrime()`

The method `getPrime()` begins with a call to `getNormal()` because *prime form* is normal form transposed to begin on pc 0 (Forte 1973, 4-5). The verbal algorithm for determining prime form is shown in **Figure 3.9**.

1. Determine the normal form for the pcset.
2. Transpose the pcset so that the first number is zero.
3. Invert the pcset and determine the normal form of the inversion.
4. Determine, between the normal form of the original pcset or the normal form of its inversion, which has the smallest numbers. This is the prime form.

Figure 3.9: Verbal algorithm for determining prime form.

This method also makes a call to the method `toZero()`. This method completes Step 2 of **Figure 3.9** and transposes the normal form to begin on zero. The remainder of the code completes steps 3-4. One new implementation is a call to the Java Math library, which has a method for determining absolute value. Libraries are further discussed in section 3.2.3.

### **Forte.class**

`Forte.class` is a separate class containing the method to retrieve the SC name for any prime form, as designated by Allen Forte.<sup>4</sup> This class required the creation of a large `HashMap` containing all SC names and their correlating prime forms. The entries for this `HashMap` were taken from code written by Jay Tomlin for the Set Theory Calculator, which is available at <http://www.jaytomlin.com/music/settheory/> and as an iOS (iphone operating system) application. This `HashMap` contains key/-value pairings in which the keys are prime forms and the values are SC names. The

---

4. See Forte 1973, 179-81.

1. SC 4-3 was originally omitted. This SC was added.
2. SC 5-5 was entered twice, once with the correct pcs and once with the pcs for SC 5-3. The second entry was deleted.
3. SC 7-5 was entered twice, once with the correct pcs and once with the pcs for SC 7-3. The second entry was deleted.
4. SC 7-z18 had the prime form listed incorrectly as [0,1,2,3,6,7,10]. This was corrected to [0,1,4,5,6,7,9].
5. SC 7-22 had the prime form listed incorrectly as [0,1,3,4,7,8,9]. This was corrected to [0,1,2,5,6,8,9].
6. SC 7-33 had the prime form listed incorrectly as [0,1,3,5,7,9,11]. This was corrected to [0,1,2,4,6,8,10].
7. SC 7-34 had the prime form listed incorrectly as [0,1,3,5,7,9,10]. This was corrected to [0,1,3,4,6,8,10].
8. SC designations were added for two, ten and eleven-pitch SCs.

Figure 3.10: Corrections made to Jay Tomlin’s Set Theory Calculator HashMap.

HashMap was reviewed and edited for correctness. The changes made to the entries can be seen in **Figure 3.10**. The code for `Forte.class` is listed in **Appendix F**. This code searches the HashMap using the prime form as the key and returns the value, which is the SC name.

### **Schultz.class**

This class contains two methods: one which returns the contour reduction according to Rob Schultz’s algorithm listed in **Figure 2.8** and another which returns the contour depth. The code for these methods is listed in **Appendix G**. These methods use a different type of HashMap called a *LinkedHashMap*. This is a HashMap that stores entries in the order in which they are added, preserving the order of the pitches as the operations of flagging and pruning are completed (Horstmann and Cor-

nell 2013, section 13.2.9.2). The first method, `getSchultzCont()` (short for ‘get Schultz contour’), returns the contour reduction. The second method, `getSchultDepth()`, is exactly the same as `getSchultzCont()` except that it returns the depth for the contour reduction.

## Servlets

A *servlet* is a program that connects data entered by a user with the methods within the program. It is called into action by a jsp or HTML document, runs on the background computer server, and returns data to the user (Jendrock et al. 2014, 10). The two servlets used in CAT are `GetMidi.java` and `Results.java`. The code for both servlets is listed in **Appendix H**.

### GetMidi.java

This servlet allows a user to input a melody in the form of a *Musical Instrument Digital Interface* (MIDI) file. MIDI files encode information about music, such as pitch, duration, and volume (Cope 2008, 334). The code for this servlet reads the information in a MIDI file and searches for the number 144, which is the numerical signal for the start of a pitch. Whatever numerical value follows 144 is the MIDI pitch value. The program searches the MIDI file, finds all of the pitch events and places them into an `ArrayList` which can then be used by other methods to determine the contour reduction. In the future, other information could be parsed from a MIDI file such as duration and volume.

### Results.java

The `Results.java` servlet gathers user input, acts upon it using the classes and

methods for contour reduction and analysis, and returns the results in a text field embedded in the file `index.jsp`. The MIDI entry form and return text field can be seen in **Figure 3.1**.

### 3.2.3 Libraries

The next node in **Figure 3.7** is named libraries. Java libraries are collections of ready-made tools that are already written using Java language (Gosling and McGilton 1995, 13,76). These pre-coded tools are imported into Java documents. The command ‘import’ is used whenever a coder wants to use a library. For example, see the beginning of the method `getMidi()` in **Appendix H**. In the preamble of the document there is a list of imports for every library used in that document. CAT uses the Java utility library, GlassFish, Google Web Toolkit, the Apache Commons I/O library, the Apache Commons fileupload library, and the Java servlet library.

### Java Development Kit

The Java development kit (jdk) is a Java program development environment for programmers who want to write Java computer programs. It includes the Java virtual machine (JVM). The JVM is a hypothetical machine that converts a program in Java bytecode (intermediate language) into machine language and executes it (Horstmann and Cornell 2013, section 1.4.0). The jdk also contains a compiler, debugger, and the Java utility library for developing Java applications. The Java utility library contains the tools for previously discussed objects, such as `ArrayList`, integers, and `HashMaps`.

## **GlassFish**

GlassFish is the Java library that contains all the support for web applications, such as servlets and JavaServer Pages (Jendrock et al. 2014, 33). It allows a programmer to develop and run web applications. It also allows for the application to be tested without the use of a server.

## **Google Web Toolkit**

The Google Web Toolkit (gwt) is an open source program development kit from Google for creating browser-based applications (Freedman, accessed July 15 2014, “google web toolkit”). The gwt automatically handles some cross-browser issues, making web applications work smoothly across most Web browsers.

## **Apache Commons**

Apache Commons is a collection of open source reusable Java components. There are two tools within Apache Commons that are utilized by CAT. The Apache Commons fileuploader includes tools for reading and parsing user files (Freedman, accessed July 15 2014, “fileupload”). This tool was used in the getMidi servlet and is imported using the command: `import org.apache.commons.fileupload`. The Apache Commons I/O (input/output) tool is used in conjunction with servlets to handle I/O exceptions. An I/O exception is any unexpected problem in a Java program (Horstmann and Cornell 2013, section 11.1.1). It tells the program how to handle errors it encounters, such as missing files or reaching the end of a file while parsing. Also in the Apache Commons I/O library is an `InputStream` tool. It converts the characters of a file into an encoded stream of bytes. The bytecode can then be analyzed and parsed.

This library is also used in the getMidi servlet.<sup>5</sup>

### 3.2.4 Configuration Files

The final node in **Figure 3.7** is named Configuration Files. These documents contain the information that the program needs to execute. Execution is the act of running a program, which causes the computer to carry out its instructions, as indicated by the code (Freedman, accessed July 15 2014, “execute”). The manifest contains a list of all files that are used by the program when it is distributed. In Java, files are packaged as a Java archive (JAR) files for distribution. JAR files are a single file into which multiple Java files are bundled for distribution (Horstmann and Cornell 2013, section 10.1.0). The previously discussed web.xml file is also listed under the Configuration Files node. It is listed under two nodes because it offers information specific to the implementation of the web application as well as the overall configuration of the program.

### 3.2.5 Summary

The IDE Netbeans was used to write a web application named CAT, which analyzes musical contour. Rob Schultz’s version of the CRA, listed in **Figure 2.8**, is used as the primary tool for contour analysis. The application also provides information such as Forte SC designation, normal form, prime form, contour depth, and contour prime.

Java language was used. The user encounters a GUI that uses the file index.jsp

---

5. For descriptions of all classes in Apache Commons I/O, see <http://commons.apache.org/proper/commons-io/javadocs/api-release/index.html?org/apache/commons/io/input/package-summary.html>.



to determines the application’s look and feel. The user has the option to manually enter pitches as integers or to upload a MIDI file of a melody for the program to read. If the user opts to upload a MIDI file, the program calls the servlet `GetMidi.java`. This returns the pitch content of the file as a list of integers. If a user enters integers manually, clicking the “submit” button calls the servlet `Results.java`. The servlet then calls the various methods needed to determine the contour reduction, contour prime, normal form, prime form, and SC designation.

### 3.3 Microsoft Excel and TG Segmentation

Before proceeding with the description of the Microsoft Excel spreadsheet developed to complete temporal gestalt (TG) segmentation, some changes to the original process of Tenney and Polansky (T & P) should be pointed out. The author experimented with many different options for entering parameters, which parameters to use, and what additional methodologies to consider (such as Lefkowitz and Taavola 2000, Polansky 1979, and Hanninen 2012). The results of this experimentation, on the body of music represented here, show little or no effect on TG segmentation when implementing the following methods:

1. Changing the weighting values of parameters, whether arbitrarily or based on Leftowitz and Taavola’s Total Segmentation Value (TSV).
2. Including the initial and final intensity values for each pitch as described in Polansky 1979.
3. Using a boundary interval calculation at the clang level as described in Polansky 1979.
4. Adjusting the start-time based on metronome indications as described in Tenney and Polansky 1980.
5. Division of the boundary interval sum at further hierarchical levels by a factor of 2.

Based on these findings, the author has chosen the following method for TG segmentation. While the original method of T & P has been altered, the Microsoft Excel spreadsheet described shortly is flexible enough that a user can choose a preferred methodology.:

1. Convert pitch letter names to integer values.
2. Determine each pitch's duration based on a common rhythmic value, i.e. the quarter note. This is at the discretion of the analyst. If a piece contains mixed or asymmetrical meters, the eighth or sixteenth note may be the best value. If a pitch is followed by a rest, include that value in the duration. If a pitch or rest is marked with a *fermata*, add 1 to the total duration value. When dealing with tuplet figurations, include fractional information to at least the ten-thousandths decimal position.
3. Convert both pitch and time into normalized values between 0 - 1. To do so, divide each pitch value by the total number of pitches contained in the piece. Then divide each duration by the absolute value of the difference between the shortest and longest durations.
4. Use the normalized duration values to determine pitch start-times by adding each duration to the previous duration, starting at 0.
5. Determine the pitch distance interval between every pitch, which is the absolute value of the distance between each pitch.
6. Determine the start time distance interval between every pitch. This is the absolute value of the difference between the start-times of each pitch.
7. Determine the disjunction value by summing the pitch distance intervals and the start time distance intervals. *Clang* termination occurs when the disjunction value between two pitches is greater than the disjunction value on either side.
8. Determine the pitch mean interval for each *clang* by summing the pitch values in each *clang* and dividing by the total number of values.
9. Determine the time mean interval from the first pitch of each *clang* (or whatever the next lowest level is) to the starting pitch of the next *clang*.
10. Sum the pitch and time mean intervals.
11. Determine the pitch boundary interval by calculating the difference between the final pitch of the *clang* (or whatever the next lowest level is) and the initial pitch of the next *clang*.

12. Determine the time boundary interval by calculating the difference between the final pitch's start time of the *clang* (or whatever the next lowest level is) and the initial pitch's start-time of the next *clang*.
13. Sum the pitch and time boundary intervals.
14. Determine the disjunction value by summing the mean and boundary intervals. *Sequence* (or any future level) termination occurs when the disjunction value between two pitches is greater than the disjunction value on either side.

A different computing approach was used for the completion of TG segmentation. Macros were created using Visual Basic for Applications (VBA) within a Microsoft Excel spreadsheet. Macro language is used within spreadsheet or word processor applications to automate tasks (Freedman, accessed July 15 2014, "macro language"). VBA is the name of the programming language for Microsoft Excel (Syrstad and Jelen 2004, Introduction: Getting Results with VBA). Over forty macros were written to complete each step in the determination of a TG segmentation from the lowest level (*clang*) to the highest level (piece). The result is a spreadsheet available for download at: [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). Once the initial data for pitch and time are entered, it is possible to click a series of buttons that calculate each step of a TG segmentation and the macros embedded within the spreadsheet calculate the numerical data for the various TG levels.

An example of the spreadsheet, with TG segmentation already calculated, is shown in **Figure 3.11**. This example shows the opening pitch and time information, represented as integers, for Kazuo Fukushima's *Mei* for solo flute. Each higher level looks the same on the spreadsheet except that column headings have been changed to represent the level being analyzed. The X's represent where *clang* termination occurs.

	Enter pitch weight value ↓		Enter time weight value ↓	Enter intensity weight value ↓		Enter timbre weight value ↓
	1.000		1.00	0.00		0.00
Press Buttons For Results →	WEIGHTED PITCH VALUE	METRONOME ADJUSTED TIME	WEIGHTED TIME VALUE	WEIGHTED INITIAL INTENSITY VALUE	WEIGHTED FINAL INTENSITY VALUE	WEIGHTED TIMBRE VALUE
	0.083	0.00	0.00	0.00	0.00	0.00
	0.389	0.50	0.50	0.00	0.00	0.00
	0.444	0.62	0.62	0.00	0.00	0.00
	0.389	0.99	0.99	0.00	0.00	0.00
	0.444	1.05	1.05	0.00	0.00	0.00
	0.417	1.37	1.37	0.00	0.00	0.00
	0.444	1.58	1.58	0.00	0.00	0.00
	0.472	1.63	1.63	0.00	0.00	0.00
	0.500	1.70	1.70	0.00	0.00	0.00
	0.486	1.81	1.81	0.00	0.00	0.00
	0.500	1.81	1.81	0.00	0.00	0.00
	0.444	1.87	1.87	0.00	0.00	0.00
	0.472	1.89	1.89	0.00	0.00	0.00
	0.472	1.94	1.94	0.00	0.00	0.00
	0.472	2.06	2.06	0.00	0.00	0.00
	0.500	2.10	2.10	0.00	0.00	0.00
	0.458	2.14	2.14	0.00	0.00	0.00
	0.500	2.15	2.15	0.00	0.00	0.00
	0.472	2.23	2.23	0.00	0.00	0.00

Figure 3.11: Portion of fully calculated TG segmentation spreadsheet for Kazuo Fukushima's *Mei* for solo flute.

An example of a macro written using VBA is shown in **Figure 3.12**. Much of the same programming syntax used in VBA is the same as Java, and this holds true of all programming languages. However, there are a few differences. First, instead of methods, VBA uses subroutines. Subroutine is abbreviated as “sub” and every macro in this particular spreadsheet contains its own subroutine. In **Figure 3.12**, the subroutine is named “pitch\_interval()”. All variables are declared by the command “Dim,” which stands for “dimension.” Dimensions can be many types of variables such as integers, Doubles, and Ranges. Ranges are the main dimension in a spreadsheet because they represent a combination of rows and columns. For instance, using the Range command, the program is able to select a specific cell, such as ‘E2’. Then, using the command for ActiveCell, the program can perform a task using the value in that cell and then move to the next cell (ActiveCell.Offset(1,0).Select) and perform the operation again. Examples of the spreadsheet data for each piece in the

analysis chapter can be found online at [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). The code for the VBA macros can be seen in **Appendix I**.

This method, while decreasing significantly the amount of time needed to complete a TG segmentation, has many limitations. First, a user must still convert musical information on the score into numerical data at the *element* level. This is time-consuming and also opens a window for user error. Second, data for pitch, time, and intensity may be entered into this spreadsheet. A user may want to calculate TG segmentation based on other domains. Unfortunately, if additional columns are added to this spreadsheet, all of the macros must be edited to reflect the change in the number of columns. Finally, the output is strictly numerical. Therefore, a user must convert the numerical information back into pitch information to mark the TG segmentation on the musical score.

To overcome these limitations, in the future the spreadsheet could be programmed so that users enter pitch information in the form of pitch letter names and octave identification. An additional macro could be written to convert the pitch information into numerical data. The spreadsheet itself is available for free without password protection. Therefore, anyone can access and edit the macros as needed. Finally, while the spreadsheet is limited to the scope of the needs of this dissertation, the time saved in calculating TG segmentation has been invaluable to the author.

```

Sub pitch_interval ()
Dim b As Range
Dim c As Range
Dim i As Integer
Dim MyCount As Integer
MyCount = Application.CountA(Range "B:")
Set b = Range("B2:" & Cells(MyCount, "B").Address)
Dim answer As Double
Dim counter As Integer
i = 1
counter = 1

Range("E2").Activate

Do While counter < MyCount

For Each c In b
    answer = Abs(ActiveCell.Offset(0, -3).Value - ActiveCell.Offset(1, -3).Value)
    ActiveCell.Formula = answer
    ActiveCell.Offset (1, 0).Select
    counter = counter + 1
Next c
Loop
End Sub

```

Figure 3.12: Example of a macro written using VBA which determines the pitch interval at the *clang* level of TG segmentation.

### 3.3.1 How to Use the TG Segmentation Spreadsheet

1. The first column has no bearings on spreadsheet calculations. It may be used by an analyst as a reminder of measure or line numbers for a score.
2. Pitch data is entered into the second column.
3. Start-time data is entered into the third column.
4. Additional data may be entered for intensity, timbre, or metronome indications. If an analyst chooses to not use these parameters, then zeros should be entered for the length of the column, except for in the case of the metronome value, where 60 should be entered for the length of the column. While there is a column labeled “timbre,” any parameter value could be used in this space.
5. Adjust weighting values across the top of the second section as needed.
6. Press each button across the top of the spreadsheet to calculate the given step of TG segmentation.
7. All other calculations are completed by the spreadsheet (by the pressing of buttons) once a user has entered the initial pitch and start-time data. Further levels will appear in the tabs across the bottom of the spreadsheet labeled “Sequence\_Determination,” “Segment\_Determination,” and “Section\_Determination.”

## 3.4 Typesetting

### 3.4.1 Lilypond

The idea of programming has been taken one step further in this dissertation. The author has attempted to implement coding in every aspect of this document. A brief word should be said about the typesetting of the musical examples and the actual dissertation document itself. All musical examples were constructed using an open source music engraving program called Lilypond.<sup>6</sup> The authors of Lilypond set out to answer the question: “Why does most computer output fail to achieve the beauty and balance of a hand-engraved score?” Their program is designed to mimic the finest

---

6. [www.lilypond.org](http://www.lilypond.org).

hand-engraved scores (Lilypond Development Team, Accessed 14 July 2014, 1). First, they developed a Lilypond font which is more “voluptuous” and weightier than current music program fonts (Lilypond Development Team, Accessed 14 July 2014, 5). The font uses spacing algorithms which distribute the space between elements based on the duration of notes, forgoing mathematical precision for traditional music engraving which is more legible. The authors of Lilypond devised a concept called optical sizing, which means that the engraving font will print clearly at any size (Lilypond Development Team, Accessed 14 July 2014, 6-7). The language for Lilypond is a clear and unambiguous syntax for music engraving. It is reliable and robust. Lilypond can engrave musical examples in a wide range of musical genres, from J.S. Bach to Schenker graphs to neumatic and mensural notation. Just a few examples of the output capabilities of Lilypond are shown in **Figures 3.13** and **3.14**.<sup>7,8</sup>

---

7. Figure 3.13 is taken from the Lilypond examples website, accessed 16 July 2014 <http://lilypond.org/doc/v2.19/Documentation/web/examples>.

8. Figure 3.14 is taken from the Lilypond examples website, accessed 16 July 2014 <http://lilypond.org/doc/v2.19/Documentation/web/examples>.



# Jesu, meine Freude

BWV 610 Johann Sebastian Bach

**Largo**

Figure 3.13: An example of music engraving using Lilypond: J.S. Bach BWV 610.

# Wenn wir in höchsten Nöten sein

Analysis from Gene Biringer's Schenker Text, Ex. 5-27 J.S. Bach

Figure 3.14: An example of music engraving using Lilypond: Schenker graph.

Sal- ve, Re- gí- na, ma- ter mi- se- ri- cór- di- ae: Ad

te cla- má- mus, éx- su- les, fi- li- i He- vae. Ad te su- spi-

rá- mus, ge- mén- tes et flen- tes in hac la- cri-

má- rum val- le. E- ia er- go, Ad- vo- cá- ta no- stra, il-

los tu- os mi- se- ri- cór- des ó- cu- los ad nos con- vér- te.

Et Je- sum, be- ne- dí- tum fruc- tum ven- tris tu- i, no-

bis post hoc ex- sí- li- um os- té- n- de. O cle- mens: O

pi- a: O dul- cis Vir- go Ma- rí- a.

Figure 3.15: An example of music engraving using Lilypond: Ancient notation (Lilypond Notation Reference, section 2.9).

```
\relative c'{
  \numericTimeSignature
  \time 4/4
  \override DynamicLineSpanner.staff-padding = #4
  ees1^\markup { Lento e rubato } \pp \< ~ |
  ees2\! \fermata \> r8.\! d'16( \pp \< ~ d4 ~ |
  \times 2/3 { d4\! e8 ~ } e2.\> |\break

  e4) r4\! \times 2/3 { r8 d4(\mp ~ } d16 e8. ~ \pp \< |
  e2 ~ \times 2/3 {e4\!\sim e2\>\glissando \glissandoSkipOn} |
  e16 \glissandoSkipOff ees8. ~ ees4) r4\! r8
  \override Slur #'positions = #'(1 . 1) e?(\mp\<\sim | \break
```

Figure 3.16: Example of Lilypond syntax



Figure 3.17: Engraved output of Lilypond syntax shown in Figure 3.16.

After learning the syntax, a Lilypond user can create a musical score by first entering the information about the the score into a text document. Then Lilypond can be used to compile the document and turn it into an engraved score. An example of a portion of the text document for **Figure 4.1**, used in the following chapter, is

shown in **Figure 3.16**. **Figure 3.16**, when compiled through Lilypond, creates the first two lines of the score of Kazuo Fukushima’s *Mei* for solo flute, which is shown in **Figure 3.17**.

### 3.4.2 $\text{\LaTeX}$

$\text{\LaTeX}$  is a document preparation system for high quality typesetting. It is most often used for scientific and mathematical documents, however its high level of functionality and beautiful output would accommodate any discipline. Originally developed in the 1970s to typeset complex mathematical text, it is now widely used in the publishing industry for academic journals, particularly by mathematicians, physicists, and others who have complex notational requirements (Unwalla 2006, 33).

An author writes a document in a text editor using  $\text{\LaTeX}$  syntax. This syntax includes commands for how to process the text. The text document is then compiled and can be output as a portable document format (PDF) or PostScript file. GUI editors are available for both Lilypond and  $\text{\LaTeX}$  to make viewing and editing documents more streamlined. These GUIs contain debuggers and compilers as well as document preview options. A short example of  $\text{\LaTeX}$  syntax and output is shown in **Figures 3.18** and **3.19**.

```
\documentclass{article}
\begin{document}
A \textbf{bold} \emph{Hello \LaTeX} to start!
\end{document}
```

Figure 3.18: An example of  $\text{\LaTeX}$  syntax. This example comes from Unwalla 2006, 33.

A **bold** *Hello*  $\LaTeX$  to start!

Figure 3.19: Output for example of  $\LaTeX$  syntax shown in Figure 3.18.

The beauty of  $\LaTeX$  is that once the initial document is set up, the formatting of the document is completely automated. The bibliography, glossary, table of contents and list of figures are auto-generated. Figures renumber themselves according to their location in the document, even if moved, as long as they are given a label name. Once all definitions are added to a glossary, each term can be hyperlinked within the document. Therefore, as one reads this dissertation in a PDF, they may click on any term and be brought to the glossary page with its definition.

$\LaTeX$  interacts with another program called BibTeX which is used to process lists and references. A BibTeX file stores bibliographical information. Once a reference's information is entered into a BibTeX document and given a label, it can be easily referenced inside a  $\LaTeX$  document. The bibliographic style can be changed with one line of code which communicates to  $\LaTeX$  that the bibliography should be automatically formatted in APA, MLA, Chicago Style, etc. A user does not necessarily need to type all the bibliographical information into the BibTeX file. When using a search engine such as Google Scholar to search for resources, a user can click on the 'cite' link below the source's information and then click 'import into BibTeX' to procure the resource's bibliographical information already formatted for the BibTeX file.

### 3.4.3 Summary

The author has attempted to use the concepts of computer programming in every aspect of this document. The computer programs used or created by the author are all open source and are available to anyone wishing to utilize computer programming as a means of music analysis, document typesetting, or the creation of musical examples. While the resources used herein are current at the time of publication, new computer programs and resources become available every day and the author encourages readers

to continually seek out new ways to incorporate open source computer programs into their research.

# Chapter 4

## Analysis

### 4.1 Kazuo Fukushima: *Mei* (1962)

Kazuo Fukushima (b. 1930) composed *Mei* for solo flute as a requiem for Wolfgang Steinecke. Steinecke was the creator and director of the Darmstadt School International Summer Courses in New Music and was tragically struck and killed by a car in 1961 (Toop 2004, 360). *Mei* is Kazuo Fukushima's most well-known composition. It is a standard in the flute repertoire and has been a required piece in both national and international flute competitions (Watanabe 2008, 16).

*Mei* is written in Western notation for the Western flute, but mimics the timbre, scale content, and character of the *noh-kan*, a Japanese bamboo flute used in *Noh* theater. Fukushima evokes the sound of the *noh-kan* by using microtones, *glissandi*, and *portamenti*. Key clicks mimic the sound of *Noh* percussion. Scholars Mihoko Watanabe and Chung-Lin Lee have revealed that the work also uses melodic motives and scale systems that are characteristic of *noh-kan* performance as well as indicative intervals, such as the seventh, which on the *noh-kan* is the approximate pitch obtained by over-blowing.<sup>1</sup>

An aspect of Japanese art prevalent in the work of Fukushima is *ma*. *Ma* is inter-

---

1. See Watanabe 2008 and Lee 2010.

preted as emptiness or space. In Fukushima’s works it refers to the expressive space between musical phrases (Watanabe 2008, 18). “[It is] a keen, intuitive awareness containing some tension—a perceptual silence (Watanabe 2008, 18).” In *Mei*, *ma* delineates the piece’s overall form as well as the small-scale segmentation.

There are two errata in the printed score. First, Fukushima intended for the first pitch, Eb4, to have a *fermata* (Artaud and Dale 1994, 156). Second, there is a misprint six measures from the end. The A4 on beat one should be a Bb4, just as in measure 10 (Watanabe 2008, 19). These errata have been corrected in the examples used within this document.

## Form

A complete score is shown in **Figure 4.1**. *Mei* is composed in ABA’ form. Each section is demarcated by *ma* (notated as *fermatas*) and a tempo change. Section A occurs between measures 1 and 15. It is labeled *Lento e rubato* and ends with a *fermata* over a quarter-note rest.<sup>2</sup> Section B occurs between measures 16 and 51. It is labeled *Piú mosso* and ends with a *fermata* over a half-note rest. In his 2010 dissertation, *Analysis and Interpretation of Kazuo Fukushima’s Solo Flute Music*, Chung-Lin Lee divides section B into four subsections, which are labeled B1 through B4 (Lee 2010, 68). Subsection B1 occurs between measures 16-26; B2 between measures 26-35; B3 between measures 36-43; B4 between measure 44-51.

---

2. The use of the term “section” in this analysis should not be confused with T & P’s use of the term *section*. “Section” (without italics) is used in terms of the regular nomenclature for discussing musical form. Where T & P’s term *section* is meant, it will be placed in italics to avoid confusion.



Section: 1  
Segment: 1  
Sequence: 1  
Clang: 1

SC 3-1 {234}

Clang: 2

Clang: 3

SC 3-1 {234}

Clang: 4

Clang: 5

Sequence: 2  
Clang: 6

SC 3-1 {456}

Clang: 7

Clang: 8

Sequence: 3  
Clang: 9

Clang: 10

Sequence: 4  
Clang: 11

Clang: 12

SC 3-1 {567}

Clang: 13

Sequence: 5

Clang: 14

SC 3-2 {9t0}

SC 3-1 {9te}

SC 4-12 {0346}

Figure 4.1 shows a musical score for Kazuo Fukushima's *Mei* for solo flute. The score is divided into six systems, each containing a staff of music with various annotations. The first system includes a circled 'A' and a key signature of one flat. The score is marked with dynamic levels (pp, mp, mf, f, sf, fff, p) and articulation (acc., rall., a tempo). It also features specific sequence and clang markings (e.g., SC 3-1 {234}, Clang: 1, Clang: 2, etc.). The notation includes various note values, rests, and slurs, with some measures containing triplets or other rhythmic groupings.

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute. Both Lee's formal division and TG segmentation are indicated.

© Sugarmusic S.p.A. - Suvini Zerboni, Milano (Italy)

Segment: 2  
Sequence: 6  
Clang: 15

Più mosso

(B1)

Clang: 16

Sequence: 7  
Clang: 17

Clang: 18

SC 3-1 {e01}

SC 3-1 {e01}

Clang: 19

Clang: 20

SC 3-1 {123}

Clang: 21

Sequence: 8  
Clang: 22

Clang: 23

Clang: 24

SC 3-1 {234}

Clang: 25

Clang: 26

Clang: 27

SC 3-1 {e01}

SC 3-1 {012}

Clang: 28

Segment: 3  
Sequence: 9

(B2)

PPP

PPP

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute (cont).

28 Clang: 29 SC 3-8 {026} Sequence: 10 Clang: 30

31 Clang: 31 SC 3-8 {931} SC 4-12 {0236} Sequence: 11 Clang: 32 SC 3-1 {te0}

33 Clang: 33 SC 3-1 {123} Clang: 34 Clang: 35 Sequence: 12 Clang: 36 Clang: 37 SC 3-1 {9te}

35 Segment: 4 Sequence: 13 Clang: 38 Flatt. B3 Clang: 39 Clang: 40 SC 3-1 {678}

37 Clang: 41 Clang: 42 Sequence: 14 Clang: 43 SC 3-1 {345}

Dynamics: *mp*, *sf*, *p*, *ppp*, *fff*, *pp*, *mf*, *ff*, *fff*, *p*, *pp*, *fff*, *mp*, *ff*, *fff*, *mf*, *f*, *fff*, *ff*, *sf*, *mp*, *mp*, *sf*, *fff*, *ff*, *sf*.

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute (cont).

4

Clang: 44

Clang: 45

Clang: 46

Sequence: 15

Clang: 47

SC 3-1 {678}

SC 3-8 {60t}

SC 4-12 {28te}

39

Clang: 48

Clang: 49

Sequence: 16

Clang: 50

Clang: 51

Clang: 52

SC 3-2 {023}

SC 3-2 {124}

41

Clang: 53

Flatt.

Sequence 17

Clang: 54

Clang: 55

Clang: 56

SC 4-12 {17te}

SC 3-4 {078}

SC 3-5 {671}

SC 3-2 {578}

43

Sequence: 18

Clang: 57

(B4)

Clang: 58

Clang: 59

Clang: 60

Clang: 61

quasi SC 3-1 {012}

45

Clang: 62

Clang: 63

Clang: 64

Clang: 65

Clang: 66

SC 5-1 {6789t}

SC 3-2 {124}

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute (cont).

Section: 2  
Segment: 5  
Sequence: 19  
Clang: 67

*accel.*

Clang: 68

SC 4-12 {2568}

SC 3-2 {9t0}

*fff* *ff* *ff*

*molto*

Sequence: 20 Clang: 70  
Clang: 69 Clang: 71 Sequence: 21 Clang: 72 Clang: 73

SC 3-1 {345} SC 3-1 {789} SC 3-1 {te0}

SC 3-8 {248} SC 3-2 {679}

*mf* *fff* *pp* *f* *molto*

47

48

49

*molto* *fff* *subito ppp*

*lunga*

Meno Mosso

Segment: 6  
Sequence: 22  
Clang: 74

Clang: 75 Clang: 76

SC 3-1 {234}

*pp* *mp* *pp* *pp*

52

(A')

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute (cont).

6

Sequence: 23  
Clang: 77

Clang: 78

56

SC 3-1 {456}

*p* *mf* *pp*

Sequence: 24  
Clang: 80

Clang: 79

Clang: 81

58

SC 4-1 {5678}

*mp* *mf* *mp* *mf* *f*

Sequence: 21  
Clang: 82

60

SC 3-2 {9t0}

*ff* *pp* *mp* *pp* *ppp*

SC 3-1 {9te}

Sequence: 25  
Clang: 83

Clang: 84

63

SC 4-12 {0346}

*ppp* *ppp* *pppp*

Figure 4.1: Kazuo Fukushima's *Mei* for solo flute (cont).

Section B is more complex than sections A and A'. Dynamic contrast increases dramatically and the melody is more angular, changing register often. The end of subsection B1 coincides with longer rhythmic values, softer dynamics and the use of rests to create space between it and the next subsection. Subsection B2 begins with a large change in register but maintains the *ppp* dynamic of subsection B1. The end of subsection B2 coincides with a *decrescendo* and a breath mark. Subsection B3 begins with a change of timbre, created by the use of key clicks. It ends with a *decrescendo* to a C#4 prolonged with a *fermata*. The delineation between subsections B3 and B4 is blurred because there is no notated space or registral or dynamic contrast between the two subsections. Subsection B4 ends with a prolonged B6 followed by a half rest with a *fermata* marked *lunga*.

The A' section occurs from measure 52 to the end. It is labeled *meno mosso*. While the pitch-class content and pitch order of section A' are essentially the same as section A, rhythmic changes do not allow the labeling of this section as a direct repetition of section A.

## Segmentation

For temporal gestalt (hereafter referred to as TG) segmentation, pitches were entered as semitones and durations were related to quarter-note beats. Grace notes or *acciaccatura* were entered as a fraction of the beat, .01, an analytical approximation of the value of a grace note determined by the author. This value of .01 was subtracted from the value of the previous pitch. A time value of one beat was added for each *fermata*. These values were then normalized to a range between 0-1. Pitch was normalized by dividing every pitch value by the number 37 (the range of the piece being 0-36) and duration values were divided by the number 15.5 (the absolute value of the difference between the longest and shortest durations).

The following discussion is a comparison of Chung-Lin Lee's division of the piece

with the results of TG segmentation. TG segmentation divides the work into 84 *clangs*, 25 *sequences*, six *segments* and two *sections*. Microsoft Excel data for this piece is available at [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). The placement of *clangs*, *sequences* and *segments* and *sections* is shown in **Figure 4.1**.

*Segments* 1 and 8 coincide with sections A and A'. *Segment* 1 is divided into 14 *clangs* and five *sequences*. The onset of each *clang* generally coincides with the start of a phrasing slur, except in measures 6 and 9. In measure 6, *clang* 3 begins on Eb5 in the middle of a phrasing slur and in measure 9 *clang* 9 begins on the last pitch, F5, of a phrasing slur. More will be said on this issue shortly.

TG segmentation results in a subdivision of section B that differs from Lee's analysis. Section B is divided into four *segments* (*segments* 2-5), which occur at different locations than Lee's subsections. *Segment* 2 and subsection B1 are in agreement. *Segment* 3 and subsection B2 begin at the same location in measure 26 but discrepancy follows. *Segment* 3 concludes at the end of measure 34. *Segment* termination at this location is due to a large disjunction value between the pitches Bb6 and A4. Lee's subsection B3 ends in measure 35, not in measure 34. This division coincides with Fukushima's notation of a breath mark and a change in timbre.

*Segment* 4 begins on beat 1 of measure 35 on the pitch A4 and continues until measure 47. *Segment* 4 termination occurs in measure 47 between pitches Ab6 and D4. While this TG segmentation is the result of a larger disjunction value, there is little musical evidence to support this point of division. The *crescendo* to *sff* adds further complication since the performer must provide a smooth motion from Ab6 to D4. Segmentations such as this one reveal the problematic nature of TG segmentation. While it offers a rule-based framework for analysis of post-tonal music, an analyst must still use all musical evidence to determine segmentation.

A brief discussion of the work of David S. Lefkowitz and Kristin Taavola will further illustrate this problem. Lefkowitz and Taavola include an analysis and dis-



cussion of *Mei* in the article “Segmentation in Music: Generalizing a Piece-Sensitive Approach (Lefkowitz and Taavola 2000).” Their analysis examines measures 1-26, so it cannot aid in understanding the discrepancy between TG segmentation and Lee’s formal divisions. However, their method contains the same issues as TG segmentation; that is, placing points of segmentation within phrasing slurs (see **Figure 4.2**). Lefkowitz and Taavola’s method places a segmentation in measure 11, prior to the pitch B5. This segmentation occurs in the middle of a phrasing slur. It has a Total Segmentation Value (TSV) of 1.0, which is the highest possible TSV and represents the strongest point of segmentation. Lefkowitz and Taavola point out that the results of their “system are not, therefore, ‘answers’ in and of themselves to traditional questions about the structure and content of a piece of music- that is, they do not necessarily reveal underlying unities or hierarchies (Lefkowitz and Taavola 2000, 220).” The discrepancy between the methods of Lee, Lefkowitz/Taavola and T & P confirm the definition of segmentation as presented by the author in Chapter 2: that segmentation is a linear, context-driven process based on continuity/discontinuity in which aural perception must be paired with analysis of the musical surface.

Based on the previous discussion, the author has chosen to utilize Chung-lin Lee’s division of *Mei*. This division coincides with *ma*, phrasing, dynamic tapering and timbral changes. More importantly, it coincides with Fukushima’s use of aggregates.



Figure 4.2: *Mei* discontinuities and Total Segmentation Values (TSVs) in measures 1-26 (Lefkowitz and Taavola 2000, 198).

## Pitch Class Content

Pitch-class content is aggregate-based, producing a series of “almost” aggregates which omit a single pitch-class. The omitted pitch-class is used in the subsequent aggregate. Aggregate content is as follows:

Section A: “Almost” aggregate, pc 1 omitted.

Subsection B1: SC 6-1 (e01234).<sup>3,4</sup>

Subsection B2: “Almost” aggregate, pc 7 omitted.

Subsection B3: “Almost” aggregate, pc 9 omitted.

Subsection B4: Complete aggregate.

Section A’: “Almost” aggregate, pc 1 omitted.

The melodic writing is highly chromatic, producing a pervasive use of set classes (hereafter referred to as SC) 3-1 (012) and 3-2 (013).<sup>5</sup> The use of trichords is brought out by the phrasing. For example, in section A, complete trichords are delineated by rests and occur under phrase markings, especially in measures 4 (beat 3.33) through 6 (beat 2), 6 (beat 4.5) through 8 (beat 2.5), 8 (beat 4) through 11 (beat 3), and 12 (beat 1) through 13 (beat 2). The tetrachord SC 4-12 is introduced at the end of section A. This SC is structurally important. SC 4-12 and its subsets, especially those highlighting the tritone (SC 3-5 (016) SC 3-8 (026), SC 3-10 (036), and SC 2-6 (06)), predominantly occur in subsections B2 through B4. **Figure 4.1** shows SC use throughout the piece.

Section A begins on pc 3 (Eb4) and proceeds upwards in a generally chromatic fashion. A brief silence (.75 beats) and a leap of a seventh (the interval obtained by overblowing on the *noh-kan*) follows pc 3. Pc 2 embellishes pc 3 as a lower-neighbor translated up one octave to D5. Motion to pc 4 (E5) completes the first statement of

---

3. The convention of using t to represent pc 10 and e to represent pc 11 is used throughout this document.

4. For a detailed description of aggregate structure in *Mei*, see Lee 2010, 69-70.

5. For this document the convention is adopted to label prime form in parentheses (), unordered collections in curly braces {}, and ordered collections in angle brackets <>.

SC 3-1 {234}. After another silence (1.33 beats), SC 3-1 {234} is repeated. This time, all pitch-classes are in the same octave. After another silence (1.5 beats), another instance of SC 3-1 {456} occurs, now transposed up a step and without any rest between the two SC statements.

The next phrase begins, after a brief silence (.5 beats), in measure 8. The melodic ascent continues. Various forms (pitches or microtones) of a lower neighbor pc 5 (F5) decorate pc 6 (F#5). Pc 5 receives prominence via heavy articulation on beat 4 of measure 9 and ascends to pc 7 (G5), generating another instance of SC 3-1 {567} one half-step up from the previous iteration. There is no silence. Instead, the melody continues to ascend while *crescendoing* and accelerating to pc e (B5). However, the impending climax-tone is thwarted. The *accelerando* becomes a *rallentando*. The dynamic of *fff* becomes *mp* with a *decrescendo* and the performer must slur into pc e and elongate the *decrescendo* via a *tenuto* marking. Eight pitch-classes of the first aggregate have been introduced thus far.

The next phrase proceeds *a tempo* and is in the lowest register since the beginning of section A. While the melodic motion of measures 1-11 was predominantly step-wise, measures 12-15 introduce new pitch-classes and intervals: an augmented octave descent to pc t (Bb4), a minor third skip between pc 9 (A4) and pc 0 (C5), a perfect fourth between pc 9 (A4) and pc 4 (E4), an augmented second between pc 3 (Eb4) and pc 6 (F#4) and a tritone between pc 6 (F#4) and pc 0 (C4). SC 3-2 (a subset of SC 4-12) replaces SC 3-1 in measures 12-13 (through beat 2) {9t0}. While the last pitch-class of the aggregate, pc 0, is introduced in measure 12 as an *acciaccatura*, the aggregate is definitively completed in measure 14-15 by the prolongation of pc 0 (C4). Measures 13 (beat 3) through 15 represent the first statement of set class 4-12 {0346}. SC 4-12 is replicated in sections B2, B3, and B4, the most active of the piece's sections both in pitch-class density and rhythmic complexity, specifically occurring in measures 31-32, 41-42, 47, and 63-66.

Section B replicates the motion of SC 3-1 proceeding to SC 3-2 and eventually to instances of SC 4-12. This larger pitch-class motion is replicated as a smaller scale in each subsection. In subsection B1, Fukushima immediately introduces the omitted pc 1 of section A as a C $\sharp$ 6 in the opening motive: B4-C5-C $\sharp$ 6. The melody of subsection B1 is a chromatic ascent resulting in instances of SC 3-1. The flute is in a higher register with a louder dynamic level. Pc e and pc 0 function as book ends, utilized in the ascending melodic motion in measures 16 and 17 and again in measures 24-26. The pc content of subsection B1 results in the fully chromatic hexachord SC 6-1 {e01234}.

Subsection B2 changes register and begins with a dynamic of *ppp* on a pc 5 (F4) microtone. Instead of using SC 3-1, Fukushima favors the tritone in measures 26-29, an important element of SC 4-12. The introduction of the tritone F $\sharp$ 5-C4 in measures 26-28 leads to a statement of SC 3-8 {026} at the completion of measure 29. This is followed by another instance of SC 3-8 {913} in measures 30-31 and then SC 4-12 {0236} in measures 31-32. The remaining measures of subsection B2 have an angular, melodic motion changing between high and low registers and returning to predominant use of the chromatic SC 3-1. The pc content of subsection B2 creates an “almost” aggregate omitting pc 7.

Subsection B3, separated from subsection B2 by a breath mark, begins with a change of timbre in the form of key clicks. Fukushima emphasizes pc 7, omitted from the previous aggregate, by using it as a pedal tone while the *acciaccatura* in measures 37-39 create SC 3-1 {345}. Pc 7 is then embellished with a tritone pc 1 (C $\sharp$ 6) at the end of measure 37. The predominantly chromatic motion of measures 36-39 gives way to the subsets of SC 4-12 in measures 38-43. Pc 7 is transposed 7 semitones and replicated on pc 2 (D6) at the end of measure 29, at which point SC 3-2 becomes predominant. Bridging the motion between pc 7 and pc 2 is SC 4-12 {8te2} in measures 38-39. It is again preceded by an instance of SC 3-8 {6t0}. After

several iterations of SC 3-2 in measures 39-41, SC 4-12 {7t13} returns in measure 41. Again, the first portion of this subsection, measures 44-46, is generally chromatic while the following measures, 47-51, contain instances of SC 4-12 and its embedded subsets. The pitch-class content of subsection B3 creates an aggregate omitting pc 9.

The first complete aggregate occurs in subsection B4. Both SCs 3-1 and 3-2 overlap throughout subsection B4. Another instance of SC 4-12 {2568} occurs in measures 47-48. Section A' returns to the pitch-class material of section A. While rhythmically variant, the aggregate construction is the same as section A.

Overall, the pitch-class content of *Mei* shows reliance on trichords as a compositional feature, especially SCs 3-1 and 3-2. SC 3-1 represents the highly chromatic passages which tend to occur at the beginning of sections A and A' as well as the beginning of each subsection in B. This chromatic motion then proceeds to an emphasis on the tritone, leading to eventual statements of SC 4-12. An essential feature of SC 4-12 is the tritone. This SC provides the genesis of other predominant SCs, especially 2-6, 3-2, 3-5, 3-8, and 3-10.

## Contour

Contour analysis of section A reveals several striking features. When analyzed from what Rob Shultz calls, “a synchronous analytical perspective,” (that is, one which looks at pitch data as if all pitches were presented simultaneously) it highlights important highs and lows of the melody within each *clang*, *sequence* and *section* (Schultz 2009, abstract). Synchronic contour data for *Mei* is shown in **Figure 4.3**. A Diachronic Transformational Contour Analysis (DTA) highlights the chromatic ascent in section A, but also accentuates that, as each pitch is added, the contour continually ascends. The DTA only produces the ascending contour primes <0 1> or <0 2 1>. DTA makes apparent that the contour continually ascends until the final pitch of section A, C4 (0). At this point the overall contour inverts from predominantly <0

1> or <0 2 1> to <1 2 0>. This unveils a facet of the pitch contour previously overlooked without the use of DTA: the diachronic contour is ever-ascending beyond B5 and does not relax to descent until the last pitch of the section.

<u>Section</u>	<u>Contour</u>	<u>Pitches</u>	<u>Normal Form</u>	<u>Set Class</u>	
A	1 2 0	3 23 0	e 0 3	3-3	
B	1 0 3 2	11 1 36 35	e 0 1	3-1	
	<u>Subsection</u>	<u>Contour</u>	<u>Pitches</u>	<u>Normal Form</u>	<u>Set Class</u>
	B1	0 2 1	11 35 24	e 0	2-1
	B2	1 0 3 2	5.5 1 34 9	1 5.5 9 t	N/A
	B3	1 2 0	8 36 1	0 1 8	3-4
	B4	0 1	.5 35	e .5	N/A
A'	1 2 0	3 23 0	e o 3	3-3	

Figure 4.3: Synchronic contour results of Kazuo Fukushima's *Mei* for solo flute.

The piece's overall synchronic contour is <1 2 0>, displayed as the pitches Eb4 (measure 1), C7 (measure 41), and C4 (measure 66). The synchronic contour of sections A and A' is <1 2 0>, displayed as the pitches Eb4 (measure 1 and 52), B5 (measure 11 and 60), and C4 (measures 14-15 and 64-66). This is a nesting of the piece's overall contour.

Section B has a synchronic contour of <1 0 3 2>, displayed as the pitches Bb4 (measure 16), C4 (measure 31), C7 (measure 41), and B6 (measure 48-51). These pitches generate an instance of SC 3-1 (e01), revealing use of the chromatic trichord in hierarchical levels other than the melodic surface. This contour also contains two pcs, e and 0, which, as already stated, have melodic importance in section B.

The synchronic contour of subsection B1 is <0 2 1> (B4, B6, C5), reiterating the emphasis on pcs e and 0. The synchronic contour of subsection B2 is <1 0 3 2>, a nesting of the most predominant diachronic contour (to be discussed below). The synchronic contour of subsection B3 is <1 2 0> (Ab4 , C7, C#4), a nesting of the piece's overall synchronic contour. Subsection B4 has a synchronic contour of <0 1>,

ascending from the C4 microtone to B6. The synchronic contour of section B is <1 0 3 2>, another nesting of the most predominant diachronic contour and an instance of SC 3-1 {e01}.

The data for the DTA is shown in **Figure 4.4**. It is color-coded according to the recurrence of each contour prime. The color coding is arbitrary, but is consistent throughout this document. The change of contour highlighted by the color-coding will be shown to be structurally important in the discussion of Fukushima's *Requiem*, where a change in predominant contour prime is referred to as a change in *contour shade*.

As the contour unfolds diachronically, <1 0 3 2> is the most commonly-occurring contour, representing 60% of the contours generated by the DTA (see **Figure 4.5**). No other contour comes close in usage. While the entire piece has a descending synchronic contour of <1 2 0>, the predominant diachronic contour is ascending, illustrating that, while synchronic contour can be a useful tool in finding musical relationships, it does not provide the entire description of the contour.



Measure	Pitch Location from 1-41	Contour Prime
1	1	0
2	2	0 1
3	3	0 1
4	4	0 2 1
	5	0 1
6	6	0 2 1
	7	0 1
7	8	0 1
	9	0 1
	10	0 2 1
	11	0 1
8	12	0 2 1
	13	0 1
	14	0 1
	15	0 1
	16	0 1
9	17	0 2 1
	18	0 1
	19	0 2 1
	20	0 1
	21	0 2 1
	22	0 1
10	23	0 2 1
	24	0 1
	25	0 2 1
	26	0 1
	27	0 1
11	28	0 1
	29	0 1
12	30	0 2 1
	31	0 2 1
	32	0 2 1
	33	0 2 1
13	34	0 2 1
14	35	0 1 0
	36	0 2 1
	37	1 2 0
16	38	1 3 0 2
	39	1 3 0 2
	40	1 0 2
	41	1 0 3 2

Measure	Pitch Location from 42-82	Contour Prime
	42	1 0 2
17	43	1 0 3 2
	44	1 3 0 2
	45	1 0 2
	46	1 0 3 2
	47	1 0 2
18	48	1 0 2
	49	1 0 2
	50	1 0 3 2
19	51	1 0 3 2
	52	1 0 3 2
	53	1 0 3 2
20	54	1 0 3 2
	55	1 0 3 2
	56	1 0 2
	57	1 0 3 2
	58	1 0 3 2
	59	1 0 3 2
21	60	1 0 3 2
	61	1 0 2
	62	1 0 3 2
	63	1 0 2
	64	1 0 2
22	65	1 0 2
	66	1 0 2
	67	1 0 3 2
	68	1 0 3 2
	69	1 0 3 2
	70	1 0 3 2
23	71	1 0 3 2
	72	1 0 3 2
	73	1 0 3 2
24	74	1 0 3 2
	75	1 0 2
25	76	1 0 3 2
26	77	1 0 3 2
	78	1 0 3 2
28	79	1 0 3 2
	80	1 2 0
	81	1 3 0 2
	82	1 2 0

Measure	Pitch Location from 83-123	Contour Prime
29	83	2 3 0 1
30	84	1 3 0 2
31	85	1 3 0 2
	86	1 3 0 2
	87	1 0 3 2
	88	1 2 0
32	89	1 2 0 1
	90	1 2 0
	91	1 3 0 2
	92	1 3 0 2
	93	1 0 3 2
33	94	1 0 3 2
	95	2 0 3 1
	96	2 0 3 1
	97	2 0 3 1
	98	1 0 3 2
	99	1 0 3 2
	100	1 0 3 2
	101	1 0 3 2
34	102	1 0 3 2
	103	1 0 3 2
	104	1 0 3 2
	105	1 0 3 2
	106	1 0 3 2
	107	1 0 3 2
	108	1 0 3 2
	109	1 0 3 2
	110	1 0 3 2
	111	1 0 3 2
	112	1 0 3 2
35	113	1 0 3 2
36	114	1 0 3 2
	115	1 0 3 2
	116	1 0 3 2
	117	1 0 3 2
	118	1 0 3 2
	119	1 0 3 2
	120	1 0 3 2
37	121	1 0 3 2
	122	1 0 3 2
	123	1 0 2 1

Figure 4.4: DTA contour primes from beginning to all 244 pitches of Kazuo Fukushima's *Mei*.

Measure	Pitch Location from 124- 164	Contour Prime	Measure	Pitch Location from 165-205	Contour Prime	Measure	Pitch Location from 206-end	Contour Prime
	124	1 0 3 2		165	1 0 3 2		206	1 0 3 2
	125	1 0 3 2		166	1 0 3 2		207	1 0 3 2
	126	1 0 3 2		167	1 0 3 2		208	1 0 3 2
	127	1 0 3 2	43	168	2 0 3 1		209	1 0 3 2
	128	1 0 3 2		169	2 0 3 1		210	1 0 3 2
	129	1 0 3 2	44	170	2 0 3 1		211	1 0 3 2
38	130	1 0 3 2		171	2 0 3 1	52	212	1 0 2 1
	131	1 0 3 2		172	2 0 3 1	54	213	1 0 3 2
	132	1 0 3 2		173	2 0 3 1		214	1 0 3 2
	133	1 0 3 2		174	2 0 3 1	55	215	1 0 3 2
	134	1 0 3 2		175	2 0 3 1		216	1 0 3 2
	135	1 0 3 2		176	2 0 3 1		217	1 0 3 2
	136	1 0 3 2		177	2 0 3 1		218	1 0 3 2
	137	1 0 3 2		178	2 0 3 1	56	219	1 0 3 2
	138	1 0 3 2		179	2 0 3 1		220	1 0 3 2
	139	1 0 3 2	45	180	2 0 3 1	57	221	1 0 3 2
39	140	2 0 3 1		181	2 0 3 1		222	1 0 3 2
	141	1 0 3 2		182	2 0 3 1		223	1 0 3 2
	142	1 0 3 2		183	2 0 3 1	58	224	1 0 3 2
	143	1 0 3 2		184	2 0 3 1		225	1 0 3 2
	144	1 0 3 2		185	2 0 3 1		226	1 0 3 2
	145	1 0 3 2	46	186	2 0 3 1		227	1 0 3 2
40	146	1 0 3 2		187	1 0 3 2	59	228	1 0 3 2
	147	1 0 3 2		188	1 0 3 2		229	1 0 3 2
	148	1 0 3 2		189	1 0 3 2		230	1 0 3 2
	149	1 0 3 2		190	1 0 3 2		231	1 0 3 2
	150	1 0 3 2		191	1 0 3 2		232	1 0 3 2
	151	1 0 3 2		192	1 0 3 2		233	1 0 3 2
	152	1 0 3 2	47	193	1 0 3 2		234	1 0 3 2
41	153	1 0 3 2		194	2 0 3 1	60	235	1 0 3 2
	154	1 0 2		195	1 0 3 2	61	236	1 0 3 2
	155	1 0 3 2		196	1 0 3 2		237	1 0 3 2
	156	1 0 3 2		197	1 0 3 2		238	1 0 3 2
	157	1 0 3 2		198	1 0 3 2		239	1 0 3 2
42	158	1 0 3 2		199	1 0 3 2	62	240	1 0 3 2
	159	1 0 3 2	48	200	2 0 3 1		241	1 0 2 1
	160	1 0 3 2		201	1 0 3 2	64	242	1 2 0
	161	1 0 3 2		202	1 0 3 2	65	243	1 0 3 0 2
	162	1 0 3 2		203	1 0 3 2		244	1 2 0
	163	1 0 3 2		204	1 0 3 2	66	245	1 2 0
	164	1 0 3 2		205	1 0 3 2			

Figure 4.4: DTA contour primes from beginning to all 244 pitches of Kazuo Fukushima's *Mei* (cont).

	Total Occurrence	Occurrence Percentage
0 1 =	19	8%
0 2 1 =	15	6%
1 2 0 =	8	3%
0 1 0 =	1	0%
1 3 0 2 =	9	4%
1 0 2 =	14	6%
1 0 3 2 =	147	60%
2 3 0 1 =	1	0%
2 0 3 1 =	25	10%
1 0 2 1 =	3	1%
1 2 0 1 =	1	0%
1 0 3 0 2 =	1	0%

Figure 4.5: Percentage of contour primes occurring diachronically in Kazuo Fukushima's *Mei*.

## 4.2 Kazuo Fukushima: Requiem (1966)

*Requiem* is the result of Fukushima's study of 12-tone music (Lee 2010, 39). The piece is intended to calm and settle the wandering spirits and souls of the dead (Lee 2010, 40). The piece's structure contains overlapping musical components within the monophonic texture. The individual layers of pitch content, contour, segmentation, and the placement of *ma* overlap to create a stratification of structural processes.

The complete score of *Requiem* is shown in **Figure 4.6**. The piece is composed in three sections. These sections are delineated when *ma*, twelve-tone row termination, change of diachronic contour shade, and *sequence* termination coincide. The strata will be individually examined and the overall form will be considered based on their coordination.

### Pitch Class Content

The first stratum for discussion is pitch-class content as it is organized into twelve-tone rows. The twelve-tone row used for melodic construction is: <2564307e1t98>. Fukushima utilizes only the prime, inversion, retrograde, and retrograde inversion forms of the twelve-tone row. The use of the twelve-tone row is as follows.

Measures 1-4: P (through Ab5).

Measures 4-6: P (D6 through Ab4).

Measures 6-18: P (D4 through Ab4).

Measures 19-31: I (D5 through Ab6).

Measures 32-38: I (D5 through Ab5).

Measures 38-41: R (Ab5 [shared with previous twelve-tone row] through D6).

Measures 41-47: RI (Ab5 through D4).

The figure displays five staves of musical notation for a solo flute, each representing a different section of the piece. The notation includes various musical symbols such as notes, rests, and dynamic markings, along with labels for twelve-tone row construction and TG segmentation.

**Staff 1 (Measures 1-4):** Labeled "Section: 1", "Segment: 1", "Sequence: 1", and "Clang: 1". It features a "Prime Row" and a "poco rit..." marking. Dynamic markings include *mp*, *mp*, *mf*, and *mf*. Clang labels include "Clang: 2" and "Clang: 3".

**Staff 2 (Measures 5-8):** Labeled "Sequence: 2", "Clang: 5", "Clang: 6", and "Clang: 7". It features a "Prime Row" and a "rit..." marking. Dynamic markings include *mp*, *mp*, *pp*, and *p*. Clang labels include "Clang: 4" and "Clang: 7".

**Staff 3 (Measures 9-10):** Labeled "Sequence: 3", "Clang: 8", "Clang: 9", and "Clang: 10". Dynamic markings include *mf*, *pp*, *f*, *sf*, *f*, *mp*, and *pp*. Clang labels include "Clang: 8", "Clang: 9", and "Clang: 10".

**Staff 4 (Measures 11-13):** Labeled "Segment: 2", "Sequence: 4", "Clang: 11", "Clang: 12", "Clang: 13", and "Clang: 14". It features a "Flatt." marking. Dynamic markings include *ppp*, *mf*, *ff*, *sf*, *ff*, and *fff*. Clang labels include "Clang: 11", "Clang: 12", "Clang: 13", and "Clang: 14".

**Staff 5 (Measures 14-16):** Labeled "Sequence: 5", "Clang: 15", "Clang: 16", "Clang: 17", "Clang: 18", and "Clang: 19". Dynamic markings include *ff* and *ff*. Clang labels include "Clang: 15", "Clang: 16", "Clang: 17", "Clang: 18", and "Clang: 19".

Figure 4.6: Kazuo Fukushima's *Requiem* for solo flute labeled with twelve-tone row construction and TG segmentation.

© Sugarmusic S.p.A. - Suvini Zerboni, Milano (Italy)

Sequence: 6  
Clang: 20

Clang: 21

Flatt.

Inverted Row →

Sequence: 7  
Clang: 22

*fff* *f* *p* *fff* *ppp* *mp*

Clang: 23

Clang: 24

Segment: 3  
Sequence: 8  
Clang: 25

*mf* *mp* *f*

Clang: 26

Clang: 27

Clang: 28

Clang: 29

Clang: 30

Sequence: 9  
Clang: 31

*ff*

Clang: 32

Clang: 33

Sequence: 10

Clang: 34

Clang: 35

Clang: 36

*ff* *cresc.*

Sequence: 4  
Sequence: 11  
Clang: 37

Clang: 38

Clang: 39

Clang: 40

Clang: 41

Sequence: 12  
Clang: 42

Clang: 43

*ff* *fff*

*accel.* *rit.*

Figure 4.6: Kazuo Fukushima's *Requiem* for solo flute labeled with twelve-tone row construction and TG segmentation (cont).

31 Clang: 44 Clang: 45 Inverted Row Sequence: 13 Clang: 46 Clang: 47  
 fff mp

34 Clang: 48  
 p

37 Section: 2 Segment: 5 Sequence: 14 Clang: 49 Clang: 50 Retrograded Row  
 3

40 Sequence: 15 Clang: 51 Clang: 52 Sequence: 16 Clang: 53 Clang: 54 Sequence: 17 Clang: 56 Clang: 56  
 3 3 p mp p

42 Clang: 57 Clang: 58 Clang: 59  
 pp ppp mp

45 Clang: 60

Figure 4.6: Kazuo Fukushima's *Requiem* for solo flute labeled with twelve-tone row construction and TG segmentation (cont).

Statement 1 of the twelve-tone row occurs in measures 1-4 (ending on beat 3.33). It is stated without any pitch-class repetitions. Statement 2, measures 4 (beat 3.67) through 8, restates the prime form of the twelve-tone row but at different pitch levels. Statement 3, measures 6-18, is much longer, due to pitch-class repetition. Whereas the first two twelve-tone row statements last six measures combined, this statement lasts twelve measures.

The first alteration of the twelve-tone row occurs in statement 4, beginning in measure 19 and lasting until measure 31. The twelve-tone row is inverted. Again, this statement is elongated by pitch-class repetition. Statement 5 occurs in measures 32-38. The last pitch-class of this row becomes the first pitch-class of the next statement. Statements 6 and 7 proceed to the end of the piece without further overlap or repetition. Statement 6 is retrograded in measures 38-41 and statement 7 is retrograded and inverted in measures 41-47.

Fukushima studied the twelve-tone method as described by Ernst Krenek in his treatise *Studies in Counterpoint Based on the Twelve-Tone Technique* (Lee 2010, 39). The rules which Fukushima seems to have utilized to are:

Avoid more than two major or minor triads formed by a group of three consecutive tones (Krenek 1940, 1).

Once begun, continue the series until its conclusion (Krenek 1940, 31).

Repetitions of tones is permitted before the next tone of the series is sounded (Krenek 1940, 33).

Repetition of tones after the next tone has been sounded is permitted in certain situations, namely trills, tremolos, tremolo-like formations, and in groups where one note can be considered an auxiliary note (Krenek 1940, 34).

A theme is generally not equated with a series. The commencement of a theme should not coincide with the entrance of a series (Krenek 1940, 32).

The first rule listed above concerns the use of consecutive major or minor triads.



Nowhere in *Requiem* does Fukushima outline consecutive triads. The prime form of the twelve-tone row contains a c-minor triad consisting of pcs 3, 0, and 7. When the twelve-tone row is inverted, an A-major triad is created in measures 28 (beginning on C#5) through 29 (ending on A5), in measures 34 (beginning on C#4) through 37 (ending on A5), and in measures 42 (beginning on A5) through 43 (ending on C#5). The c-minor triad occurs again in measure 40 (beginning on G5 and ending on Eb6).

The next rule concerns the formation of the series. Fukushima carries out this rule in statements 1, 2, 6, and 7 of the twelve-tone row. The others statements, however contain repetitions of pitch-classes during the unfolding of the row. While these pitch-class repetitions often fall within the standards of the next two rules, this slower unfolding of pitch-classes is a more organic technique of twelve-tone composition which allows for a less systematic method of melodic production which, as seen in *Mei*, emphasizes the scale content of *Noh* drama and the characteristic intervals of the *noh-kan*.

The final rule listed above explains that beginnings of themes should not coincide with the beginning of twelve-tone row statements. Fukushima often offsets the beginning of the twelve-tone row from the beginning of a new phrase. For example, the first instance of *ma* is a *fermata* over an eighth-rest in measure 3. The next statement of the twelve-tone row does not begin until measure 4 on D6. He reverses this process in measure 6, where the statement of the twelve-tone row ends prior to the placement of *ma* as a *fermata* over the barline. Twelve-tone row statements 6 and 7 continue this overlapping trend. However, statements 4 and 5 both coincide with not only placement of *ma*, but also with breaks in other strata, as will be discussed below.

## Segmentation

The next stratum is segmentation based on Tenney and Polansky's (T & P) tem-

poral gestalt (TG) segmentation theory. For segmentation, pitches were entered as semitones and durations were related to quarter-note beats. Grace notes or *acciacatura* were entered as a fraction of the beat, .01, an arbitrary analytical approximation of the value of a grace note determined by the author. This value of .01 was subtracted from the value of the previous pitch. A time value of one beat was added for each *fermata*. These values were then normalized to a range between 0-1. Pitch was normalized by dividing every pitch value by the number 34 (the range of the piece being 0-33) and duration values were divided by the number 5.99 (the absolute value of the difference between the longest and shortest durations). The data for segmentation is available at [kate.cooleysekula.com/java](http://kate.cooleysekula.com/java). The complete segmentation of the score is shown in **Figure 4.6**.

Segmentation reveals 60 *clangs*, 17 *sequences*, 5 *segments*, and 2 *sections*. The *clang* and *sequence* structure will be discussed in more detail during the discussion concerning the piece's overall form.

## ***Ma***

An aspect of Japanese art that is especially prevalent in the works of Kazuo Fukushima is the use of *ma*. The author interprets *ma* as the combination of several musical factors, the most obvious being rests. But there are musical characteristics other than literal silence that add to release of tension and sense of repose. Fukushima commonly employs *decrendos* on longer pitch values, often with the addition of a *fermata*. These characteristics create a sense of musical space, offering a relaxation on a single pitch which is often paired with a subsequent rest.

The instances of *ma* in *Requiem* are as follows:

Measure 3, beat 4.5: *fermata* over eighth-rest.

Measure 6, beats 2-4: *fermata* over D4 marked *piano* followed by *fermata* over right barline.

Measure 10, beats 3-4: *fermata* over E4 marked *pianissimo* followed by quarter rest.

Measure 18: *fermata* over A $\flat$ 4 marked *pianissimo* followed by quarter rest.

Measure 31, beat 4: *fermata* over A $\flat$ 6 marked *fortissississimo* followed by half rest.

Measure 35, beat 4: *fermata* over quarter rest.

## Contour

Contour analysis reveals interesting musical facets at many structural levels. Overall, the synchronic melodic contour is <2 3 0 1>. This contour represents the pitches D5 (measure 1), A6 (measure 17), C $\sharp$ 4 (measure 35), and D4 (measure 46-47). Synchronic contour data is shown in **Figure 4.7**, which provides contour information about the TG segmentation as well as the row construction and the piece's sections (to be discussed below).

<u>Sequence</u>	<u>Contour</u>	<u>Pitches</u>	<u>Set Class</u>	<u>Normal Form</u>
1	2031	14 4 19 9	4-23	2 4 7 9
2	2301	20 26 17 18	4-12	2 5 6 8
3	102	28 11 33	3-9	9 e 4
4	201	8 2 6	3-8	2 6 8
5	120	5 11 0	3-5	e 0 5
6	102	11 7 33	3-6	7 9 e
7	021	8 14 10	3-8	8 t 2
8	102	14 10 16	3-8	t 2 4
9	10	21 15	2-6	3 9
10	102	21 15 32	3-5	3 8 9
11	201	14 1 4	3-2	1 2 4
12	2301	21 29 3 11	4-25	3 5 9 e
13	1032	19 16 29 20	4-3	4 5 7 8
14	1032	7 6 28 24	4-z29	0 4 6 7
15	120	10 11 2	3-3	0 1 4
<u>Segment</u>				
1	102	14 4 33	3-9	2 4 9
2	102	8 2 33	3-5	8 9 2
3	01	8 32	N/A	8 8
4	2031	41 1 28 2	3-2	1 2 4
<u>Piece</u>				
	2 3 0 1	14 33 1 2	3-4	9 1 2
<u>Tone Row</u>				
P	102	14 4 20	3-8	2 4 8
P	120	26 33 8	3-5	8 9 2
P	021	2 11 8	3-10	8 e 2
PI	102	14 10 32	3-8	8 t 2
PI	1032	14 1 29 19	4-z15	1 2 5 7
R	1032	20 9 29 26	4-18	2 5 8 9
RI	120	20 28 2	3-8	2 4 8
<u>Section</u>				
1	2301	14 33 2 8	3-5	8 9 2
2	102	14 10 32	3-8	8 t 2
3	2031	14 1 28 2	3-2	1 2 4

Figure 4.7: Synchronic contour results for *Requiem*. The third column displays pitches as integers representing the range of the flute where C4=0 and C7=36.

Fukushima often uses the contour  $\langle 2\ 3\ 0\ 1 \rangle$  as a melodic motive, as shown in **Figure 4.8**. The contour  $\langle 2\ 3\ 0\ 1 \rangle$  functions at a deeper structural level and highlights formal division of the piece. It is the most common contour as the DTA unfolds, representing 37% of the contours, as shown in **Figure 4.10**. The data for DTA is shown in **Figure 4.9**, which displays the contour from the opening pitch to any pitch within the piece. It is color-coded according to the recurrence of contour primes. The color coding is arbitrary. Patterns emerge based on the evolution of the overall contour from beginning to end. The color-coding shows where the predominant contour(s) change. As shown in **Figure 4.9**, the predominant coloring of pitches 1 through 23 (D4 in measure 1 through A $\flat$ 4 in measure 6), contains mostly the contours  $\langle 1\ 0\ 2 \rangle$ ,  $\langle 2\ 0\ 3\ 1 \rangle$ , and  $\langle 1\ 0\ 3\ 2 \rangle$ , represented by the colors green, blue, and purple. In measure 6, the predominant contour primes become  $\langle 1\ 2\ 0 \rangle$  and  $\langle 2\ 3\ 0\ 1 \rangle$ , represented by the colors yellow and orange. These contour primes are predominant until the C $\sharp$ 5 in measure 13. This change in predominant contour prime is referred to as a change in *contour shade*. In *Requiem*, contour shade changes at the following points:

Measure 6, after A $\flat$ 4.

Measure 13, after C $\sharp$ 5.

Measure 17, after the half-note A6.

Measure 28, after C $\sharp$ 5.

Measure 31, after A $\flat$ 6.

The figure displays a musical score for Kazuo Fukushima's Requiem, specifically focusing on the surface occurrences of the contour prime <2 3 0 1>. The score is written in 4/4 time and features various dynamic markings and performance instructions.

**Measure 1:** Starts with *Lento rubato*. The contour prime <2 3 0 1> is indicated above the first four notes. Dynamics include *mp* and *mf*. A *poco rit.* instruction is present.

**Measure 4:** The contour prime <2 3 0 1> is indicated above the first four notes. Dynamics include *mp*, *pp*, and *p*. A *rit.* instruction is present.

**Measure 7:** Dynamics include *mf*, *pp*, *f*, *sf*, *f*, *mp*, and *pp*.

**Measure 11:** Dynamics include *ppp*, *mf*, *ff*, *sf*, *ff*, and *fff*. A *Flatt.* instruction is present.

**Measure 14:** Dynamics include *ff*, *sf*, *ff*, and *cresc.* (crescendo). A *3* (triple) is indicated.

**Measure 17:** The contour prime <2 3 0 1> is indicated above the first four notes. Dynamics include *fff*, *sf*, *p*, *ff*, *fff*, *ppp*, and *mp*. A *Flatt.* instruction is present.

**Measure 21:** Dynamics include *mf*, *mp*, and *f*.

Figure 4.8: Surface occurrences of contour prime <2 3 0 1> in Kazuo Fukushima's *Requiem*.

© Sugarmusic S.p.A. - Suvini Zerboni, Milano (Italy)

2

25 *ff* *accel.*

27 *ff* *cresc.* *3*

29 *ff* *fff* *accel.* *rit.*

31 *ff* *fff* *mp*

34 *p* *< 2 3 0 1 >*

37 *p* *< 2 3 0 1 >*

40 *p* *mp* *p* *< 2 3 0 1 >*

42 *pp* *ppp* *mp* *< 2 3 0 1 >*

45

Figure 4.8: Surface occurrences of contour prime  $\langle 2\ 3\ 0\ 1 \rangle$  in Kazuo Fukushima's *Requiem* (cont).

Measure	Pitch location from 1-54	Contour Prime	Measure	Pitch location from 55-108	Contour Prime	Measure	Pitch location from 109-162	Contour Prime
1	1	0		55	1 0 2		109	1 3 0 2
	2	0 1		56	1 0 3 2		110	1 3 0 2
	3	0 1		57	1 0 2		111	1 3 0 2
2	4	1 2 0		58	1 0 3 2		112	1 3 0 2
	5	1 3 0 2	17	59	2 0 3 1		113	1 3 0 2
	6	2 3 0 1		60	1 3 0 2		114	1 3 0 2
	7	1 0 2		61	1 0 2		115	1 3 0 2
3	8	2 0 3 1		62	1 0 2	31	116	1 3 0 2
	9	2 0 3 1	18	63	2 3 0 1		117	1 3 0 2
	10	2 0 3 1	19	64	1 2 0 1		118	1 3 0 2
4	11	1 0 2	20	65	2 3 0 1		119	1 3 0 2
	12	1 0 2	21	66	2 3 0 1		120	1 3 0 2
	13	1 0 3 2		67	2 3 0 1		121	1 3 0 2
	14	1 0 3 2		68	2 3 0 1	32	122	2 3 0 1
5	15	1 0 2	22	69	1 2 0 1		123	2 3 0 1
	16	1 0 3 2		70	2 3 0 1	33	124	2 3 0 1
	17	1 0 3 2	23	71	2 3 0 1		125	2 3 0 1
	18	1 0 3 2	24	72	2 3 0 1	34	126	2 3 0 1
	19	2 0 3 1		73	2 3 0 1		127	2 3 0 1
	20	2 0 3 1		74	2 3 0 1		128	1 2 0
	21	1 0 3 2	25	75	2 3 0 1		129	1 2 0
	22	1 0 2		76	2 3 0 1	35	130	2 3 0 1
6	23	2 0 3 1		77	1 2 0 1	36	131	1 3 0 2
	24	1 2 0		78	2 3 0 1	37	132	1 3 0 2
7	25	2 3 0 1		79	1 2 0 1		133	2 3 0 1
8	26	2 3 0 1		80	2 3 0 1	38	134	1 3 0 2
	27	2 3 0 1	26	81	1 2 0 1		135	1 3 0 2
9	28	2 3 0 1		82	2 3 0 1		136	1 3 0 2
	29	2 3 0 1		83	1 2 0 1		137	2 3 0 1
	30	2 3 0 1		84	2 3 0 1	39	138	2 3 0 1
10	31	2 3 0 1		85	2 3 0 1	40	139	2 3 0 1
	32	2 3 0 1	27	86	1 2 0 1		140	2 3 0 1
11	33	2 3 0 1		87	2 3 0 1		141	1 3 0 2
	34	1 2 0		88	1 2 0 1		142	1 3 0 2
12	35	1 2 0		89	2 3 0 1		143	1 3 0 2
	36	1 2 0		90	1 2 0 1		144	1 3 0 2
	37	2 3 0 1		91	2 3 0 1		145	1 3 0 2
	38	1 2 0		92	1 2 0 1		146	1 3 0 2
13	39	2 3 0 1		93	2 3 0 1	41	147	1 3 0 2
	40	2 3 0 1		94	2 3 0 1		148	1 3 0 2
	41	2 3 0 1		95	2 3 0 1		149	2 3 0 1
	42	1 3 0 2	28	96	2 3 0 1		150	2 3 0 1
14	43	1 0 2		97	2 3 0 1		151	1 3 0 2
	44	1 0 2		98	2 3 0 1		152	1 3 0 2
	45	1 0 3 2		99	1 3 0 2	42	153	1 3 0 2
	46	1 0 2	29	100	1 3 0 2		154	1 3 0 2
	47	1 0 3 2		101	1 3 0 2		155	2 3 0 1
15	48	1 0 2		102	1 3 0 2	43	156	1 3 0 2
	49	1 0 3 2		103	1 3 0 2		157	2 3 0 1
	50	1 3 0 2	30	104	1 3 0 2	44	158	2 3 0 1
	51	1 0 2		105	1 3 0 2		159	2 3 0 1
	52	1 0 3 2		106	1 3 0 2	45	160	2 3 0 1
	53	1 0 2		107	1 3 0 2	46	161	2 3 0 1
16	54	1 0 3 2		108	1 3 0 2			

Figure 4.9: Diachronic contour primes from beginning to all 162 pitches of Kazuo Fukushima's *Requiem*.



	Total Occurences	Occurrence Percentage
2301 =	60	37%
120 =	8	5%
1302 =	45	28%
102 =	15	9%
2031 =	7	4%
1032 =	13	8%
1201 =	11	7%
01 =	2	1%
0 =	1	1%

Figure 4.10: Percentage of contour primes occurring diachronically in Kazuo Fukushima's *Requiem*.

### Formal Division

The strata of twelve-tone row construction, sequence construction, placement of *ma*, and diachronic transformational contour divide *Requiem* into three sections. This analysis coincides with that of Chung-Lin Lee (Lee 2010, 43). While there are many instances where the end of a twelve-tone row, *sequence*, placement of *ma* or change in contour shade occur, it is when they occur concurrently that large-scale form is delineated.

The first occurrence is in measure 18. *Sequence* 6/*clang* 20 end on the barline between measures 18 and 19. The twelve-tone row ends on the downbeat Ab4. Contour shade changes from predominantly <1 0 2> and <1 0 3 2> to <2 3 0 1> and <1 2 0 1> between A6 of measure 17 and Ab4 of measure 18. Ab4 obtains a sense of closure and release of tension due to the dynamic level (*ppp*) (a stark contrast to the previous *fff*), a *fermata*, and a quarter-note rest. The intense *crescendo* on A6 leads directly and without pause into the, albeit quieter, Ab4. This pitch releases musical tension with decreased dynamic level, extension via *fermata*, and the additional silence of the proceeding quarter rest.

The next section begins in measure 32. *Sequence* 12/*clang* 45 ends in measure 32,

as does a twelve-tone row (emphasized with a *fermata*). *Ma* is placed prior to the beginning of the next section and the contour shade changes from <2 3 0 1> and <1 2 0 1> to <1 3 0 2> and <2 3 0 1>. The only difference here is that the sense of space is created by the juxtaposition of Ab6 and its dynamic level of *fortississimo* with a change in tessitura to D5 in measure 32 and a change in dynamic level to *mezzo piano*.

Each section has a contour-relation to the overall piece or to the diachronic contour analysis (see **Figure 4.7**). Section 1 has a contour of <2 3 0 1>, the overall contour of the piece and an important melodic contour motive. Section 2 has a contour of <1 0 2> which, as seen in **Figure 4.10**, is one of only eight contour primes presented diachronically throughout the entire piece. Section 3 has a contour of <2 0 3 1>, another example of the eight diachronic contour primes.

In summary, *Requiem* is composed in 3 sections as follows:

Section 1: Measures 1-18.

Section 2: Measures 19-31.





Section 3: Measures 32-47.

The work is composed from a single twelve-tone row which is inverted and retrograded. The contour prime <2 3 0 1> is predominant at both the diachronic and synchronic levels. TG segmentation, placement of *ma*, twelve-tone row reiteration, contour, and articulation combine to reflect points of formal division.

### 4.3 Tōru Takemitsu: Itinerant (1989)

Tōru Takemitsu (1930-1996) was a Japanese composer influenced by Olivier Messiaen and Claude Debussy (Koozin 2002, 18-19). He was a member of the Jikken Kōbō (experimental workshop), the Japanese version of the Darmstadt School, a group of experimental composers who used Western-influenced harmonizations with actual or invented Japanese folk tunes.<sup>6</sup> The two works discussed in this document are *Itinerant* and *Air*, both written for solo flute. In both works, Takemitsu blends Japanese and Western music techniques. Similar to Kazuo Fukushima, Takemitsu’s works mimic the sound of a Japanese flute, in this case a *shakuhachi*, and incorporate the use of *ma*. Additionally, he uses Debussy’s concept of gestural imagery and Messiaen’s *modes of limited transposition* (or “modes”) as building blocks for his compositions.<sup>7</sup> *Itinerant* was composed in memory of sculptor Isama Noguchi, a friend of Takemitsu, who traveled extensively throughout his lifetime: hence the title *Itinerant* (Burt 2006, 230).


A note about the musical examples: the author has updated some elements of the original score for readability and adherence to modern notation standards. First, a change is made to Takemitsu’s trill indications. If a pitch is to be trilled it is shaded gray on the fingering diagram. The fingering diagrams are also updated pictures of the keys, instead of numeric indications. For example, Takemitsu’s original fingering


diagram  is replaced with . Second, the *fermata* signs are updated to standardized versions to discriminate between short, regular, and long *fermatas*. Therefore, Takemitsu’s *fermatas* (shown in increasing duration)  are now displayed as .

6. For a detailed discussion of Takemitsu’s time with the Jikken Kōbō, see Burt 2006, 39-49.

7. For more information on Takemitsu’s musical background and influences, see Burt 2006, Koozin 1989, and 2002.

One final note about musical symbols: Takemitsu provides a key for some further unconventional symbols, which are listed below.

*Accelerando* = 

*Ritardando* = 

N.V. = non vibrato

H.T. = Hollow tone

norm. = Normal playing (fingering)

Takemitsu evokes Japanese traditional music by indicating a variety of extended techniques which mimic the sound of the *shakuhachi*. Microtones imitate the instrument's scale system. He provides detailed instructions for timbral changes, such as H.T. for "hollow tone" and N.V. for "no vibrato." Additionally, he asks for *portamenti*, flutter tonguing, multiphonics, and non-traditional fingerings.

## Segmentation

Based on the location of *ma*, *fermatas*, and *decrescendos*, the piece can be initially divided into 25 phrases (see **Figure 4.15**). The exception to this criteria is between phrases 10 and 11. Future discussion of phrase ideas will reveal a change from a tritone oscillation phrase idea to a *portamento* phrase idea.

For temporal gestalt (TG) segmentation, pitches were entered as semitones and durations as sixteenth-note beats to the TG spreadsheet. An additional time value of 1-3 beats was added for each *fermata*, depending on whether it was short, regular or long.<sup>8</sup> Duration was then converted to represent the beat indication of a dotted quarter-note by dividing the original sixteenth-note value by six.<sup>9</sup> These values were then normalized to a range between 0-1. Pitch was normalized by dividing every

---

8. For this piece, the resultant TG segmentation was the same regardless of the addition of time values for *fermatas*.

9. The opening tempo is *Lento Misterioso* with the indication of a dotted quarter-note = "ca. 30 flexible."

pitch value by the number 37 (the range of the piece being 0-36) and duration values were divided by the number 2.055 (the absolute value of the difference between the longest and shortest duration). For unconventional trills, the notated pitch was used for the pitch data point. For multiphonics the predominant, resultant sounding pitch was used as the pitch data point. TG segmentation is shown in **Figure 4.11**.

TG segmentation divides the piece into sixty-eight *clangs*, twenty-one *sequences*, and six *segments*. The complete TG spreadsheet data is available online at [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). Division at the *sequence* level regularly correlates with the twenty-five phrases discussed above. Additionally, *clangs* and *sequences* show division within phrases and *sequence* termination correlates with rests between phrases. *Sequence* terminations provide evidence for reevaluation of the initial twenty-five phrase structure.

TG segmentation helps define more ambiguous phrasing, such as the beginning of phrase 20 (see **Figure 4.15**). While a long *fermata* on C4 followed by *ma* indicates that phrase 20 may not begin until after the sixteenth and thirty-second-rests which follow it, TG segmentation reveals a greater disjunction occurring between E6, at the end of the previous stanza, and C4. In this regard, TG segmentation helps locate the termination of phrase 19. It is combined with a typical analysis of the musical surface to provide verification or clarification of the segmentation results.

The musical score is divided into five systems, each representing a segment of the piece. The notation includes treble clefs, key signatures, and various musical symbols for clangs, sequences, and dynamics.

**System 1:** Section: 1, Segment: 1, Sequence: 1, Clang: 1. Dynamics: *p*, *fp*, *p*, *mf*, *p*, *mf* poco, *pp* al niente. Includes Clang: 2, Clang: 3, Clang: 4, Clang: 5, and Sequence: 2.

**System 2:** Sequence: 3, Clang: 6, Flutt. norm., Clang: 7, Clang: 8, Clang: 9. Dynamics: *mf*, *ff*, *p* al niente.

**System 3:** Segment: 2, Sequence: 4, Clang: 10, Clang: 11, Clang: 12, Clang: 13. Dynamics: *pp*, *mf*, *p*.

**System 4:** N.V., Clang: 14, Sequence: 5, Clang: 15, Clang: 16, Clang: 17. Dynamics: *p*, *p*, *pp*, *mf* poco, *f*.

**System 5:** Sequence: 6, Clang: 18, H.T. Flutt., Clang: 19, Clang: 20. Dynamics: *p*, *p*, *f*, *pp*, *sf*, *pp*, *f*, *p*. Includes markings for *norm.* and *port.*

Figure 4.11: Segmentation of Tōru Takemitsu's *Itinerant* for solo flute.

Takemitsu ITINERANT in Memory of Isamu Noguchi, SJ 1055

Copyright © 1989 by Schott Music Co. Ltd., Japan

All Rights Reserved

Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan

Sequence: 7  
Clang: 21

much air pressure

Clang: 22 Clang: 23 Clang: 24

*sf* *p* *ff* *mf* *molto cresc.* *ff* *p sub.* *pp*

Calm  
N.V.

Section: 2  
Segment: 3  
Sequence: 8  
Clang: 25

Clang: 26

Sequence: 9  
Clang: 27 Clang: 28 Clang: 29 Clang: 30

*p* *ppp* *p* *sf ppp poco* *p* *mf* *ff* *f*

H.T.

Clang: 31

norm. H.T.

port.

H.T.

tr

Sequence: 10  
Clang: 32

*ff* *pp* *pp* *sf* *p* *ppp* *p* *f* *ff*

Clang: 33 Clang: 34

Sequence: 11  
Clang: 35

Clang: 36

norm.

H.T. (N.V.)

port.

*> mf* *ff* *f* *fff* *ff* *p sub.* *pp*

The image displays four staves of musical notation for solo flute. Each staff contains various musical notations including notes, rests, and dynamic markings. Above the staves, there are labels for 'Sequence' and 'Clang' numbers, as well as performance instructions like 'much air pressure', 'Calm N.V.', 'H.T.', 'norm. H.T.', 'port.', 'tr', and 'H.T. (N.V.)'. Below the staves, there are dynamic markings such as *sf*, *p*, *ff*, *mf*, *molto cresc.*, *ff*, *p sub.*, *pp*, *p*, *ppp*, *sf ppp poco*, *f*, *ff*, *pp*, *sf*, *p*, *ppp*, *p*, *f*, *ff*, *> mf*, *ff*, *f*, *fff*, *ff*, *p sub.*, and *pp*. The notation includes various musical symbols like treble clefs, key signatures, and time signatures.

Figure 4.11: Segmentation of Tōru Takemitsu's *Itinerant* for solo flute (cont).

The musical score is divided into five systems, each representing a segment of Tōru Takemitsu's *Itinerant* for solo flute. The notation includes various dynamic markings, sequence and clang labels, and performance instructions.

**System 1:** L.H. 3, Segment: 4, Sequence: 12, Clang: 37, Clang: 38. Dynamics: *pp*, *sf*, *f*, *ff*, *p*.

**System 2:** Sequence: 13, Clang: 39, Clang: 40. Dynamics: *p*, *mf*, *fp*, *f*, *p*, *mp*, *pp*.

**System 3:** Sequence 14, Clang: 41, Clang: 42, Clang: 43. Dynamics: *f*, *pp* sub., *mf*, *ff*, *p*, *al niente*.

**System 4:** Segment: 5, Sequence: 15, Clang: 44, Clang: 46, Clang: 45, Clang: 47, Sequence: 16, Clang: 48, Clang: 49. Dynamics: *ppp*, *f*, *p*, *f*, *mf*, *ff*, *f*, *p*, *p*, *pp*, *al niente*.

**System 5:** Sequence: 17, Clang: 50, Clang: 51, Clang: 52, Clang: 53. Dynamics: *ppp*, *mf*, *mf*, *ppp*, *mf*, *mf*, *ff*.

Figure 4.11: Segmentation of Tōru Takemitsu's *Itinerant* for solo flute (cont).



N.V.

Segment: 6  
Sequence: 18  
Clang: 54

Clang: 55

*pp* *p* *ppp* *p*

Sequence: 19  
Clang: 56

much air pressure *ffp* *>* non tonguing *poco mfz* *>* *p* *ppp* *p* *ff* *>* *molto fff*

Clang: 57 Clang: 58 Clang: 59 Clang: 60

Sequence: 20  
Clang: 61

H.T. *port.* Clang: 62 Clang: 63 Clang: 64

*fff* *p* *sub.* *al niente* *pp* *p* *molto dim.*

Sequence: 21  
Clang: 66

Clang: 65 *poco* *p* *mf* *al niente* *mf* *poco* *pp* *f*

Clang: 67 Clang: 68

*pp* *poco mfz* *f* *fff*

Figure 4.11: Segmentation of Tōru Takemitsu's *Itinerant* for solo flute (cont).

## Pitch Class Content

In Elizabeth Robinson’s 2011 dissertation “*Voice, Itinerant and Air: A Performance and Analytical Guide to the Solo Flute Works of Tōru Takemitsu*,” she states the following:

Though Takemitsu does not reuse melodic material, he creates continuity through other means. Phrases are uneven in length, sometimes relating to phrases before and after through pitch content, but more often possessing no obvious relationship. Pitch content does not appear to have been chosen based on pitch-class sets or serial devices, but rather to support movement through Takemitsu’s emphasized intervals (Robinson 2011, 71).

The following analysis will show that Takemitsu, in fact, reuses melodic material and that pitch-class content relies largely on pitch-class set collections. Takemitsu was influenced by the compositional techniques of Olivier Messiaen. This is reflected in *Itinerant* through Takemitsu’s use of referential pitch-class pitch-class set collections.<sup>10</sup> These sets come from Messiaen’s “mode of limited transpositions.” For this discussion several pitch-class set collection abbreviations are employed:

OCT I = octatonic scale, transposition I (0134679t)

OCT II = octatonic scale, transposition II (124578te)

OCT III = octatonic scale, transposition III (0235689e)

WT I = whole tone scale, transposition I (02468t)

WT II = whole tone scale, transposition II (13579e)

VII-I = mode VII, transposition I (012346789t)

VII-II = mode VII, transposition II (e12345789t)

Throughout the piece, Takemitsu relies on modes I (whole tone), II (octatonic), and VII (two chromatic pentachords separated by a whole step). The occurrence of each

---

10. For more on Olivier Messiaen’s compositional techniques, see Messiaen 1956.

mode is shown in **Figure 4.15**. Within these referential pitch-class set collections, Takemitsu utilizes set class motivic cells, primarily the SC trichords 3-2, 3-3, and 3-8. SC 3-8, a trichord associated with mode I, is especially prominent and is often found at the end of phrases. Set class use is also shown in **Figure 4.15**.


### Motivic and Gestural Ideas

Takemitsu was also influenced by Claude Debussy's concept of gesture.<sup>11</sup> Some gestures in *Itinerant* are simply rhythmic, but he also combines intervallic content, primarily the tritone and augmented fifth, with ascending or descending motion. These short, motivic cells are labeled “W,” “X” and “Y.”

W = Two tritones, often separated by a pitch related to one of the tritones by half- or whole-step. The “W” motive often results in SC 4-28 (the “fully diminished” tetrachord), 5-28 or 5-31.

X = Motion by *portamento*.

Y = A tritone plus an augmented fifth (or vice versa), often resulting in SC 3-8 or 4-24.

Other gestural repetition occurs in the form of oscillation between two pitches forming a tritone, sweeping ascending or descending melodic motion and repetition of the rhythmic cell: . Synchronic contour gestures, to be discussed later, highlight a predominance of ascending motion which also exists at the deeper, diachronic level.

Motivic cells and gestures combine to form five phrase ideas, which can stand alone as individual phrases or combine to form longer phrases:

Sedentary: phrases that often contain occurrences of the W or Y motive that, while


---

11. Roger Graybill uses the term “gesture” to describe a grouping structure with a distinguishing internal dynamic shape. See Graybill, Atlas, and Cherlin 1994.

not musically inactive, do not contain the range or sweeping motion of ascent that other phrase ideas have.

Oscillation: phrases that often involve motion between the two pitches of a tritone.

The oscillation can be written out or involve the use of a *tremolo* indication.

Ascent: phrases that contain the rhythmic cell  and a single upward melodic motion.

Double Ascent: phrases that contain two melodic ascents, the second always going higher than the first.

*Portamento*: phrases utilizing motive X.

As mentioned earlier, these phrase ideas help locate points of segmentation. While TG segmentation locates the beginning of *sequence* 5 at one position, location of phrase ideas highlight that the B $\flat$ 4 and F $\sharp$ 5 at the start of *sequence* 5 are repetitions of the previous two pitches, which result in the completion of the ‘Y’ motive followed by a double ascent. In this respect, gestural analysis has trumped TG analysis.

## Contour

Takemitsu maintains the use of seven different synchronic contour primes throughout the 25 phrases:  $\langle 1\ 0\ 3\ 2 \rangle$ ,  $\langle 1\ 3\ 0\ 2 \rangle$  (which is also the overall synchronic contour of the piece),  $\langle 2\ 0\ 1 \rangle$ ,  $\langle 1\ 0\ 2 \rangle$ ,  $\langle 0\ 2\ 1 \rangle$ ,  $\langle 0\ 1 \rangle$ , and  $\langle 1\ 0 \rangle$ . Of these contour primes, only two are overall descending:  $\langle 2\ 0\ 1 \rangle$  and  $\langle 0\ 1 \rangle$ . Speaking diachronically, two contour primes are the most predominant:  $\langle 1\ 3\ 0\ 2 \rangle$  represents 28% of all diachronic contours and  $\langle 1\ 0\ 3\ 2 \rangle$  represents 31% of all diachronic contours. Contour data for *Itinerant* is shown in **Figures 4.12** through **4.14**.

The idea of ascending/descending motion is important to the overall concept of the piece. The idea of ascent has already been noted as a phrase idea. Contour also provides evidence for the importance of ascent. The majority of synchronic contours

are ascending. This is also true of the diachronic contour, where 75% of the contours are ascending. This represents a deeper, underlying motion of ascent within the piece.

A complete motivic, phrase idea, contour and pitch content analysis of *Itinerant* is shown in **Figure 4.15**. At the beginning of each phrase, pertinent information is listed in the following order: “mode,” synchronic contour, phrase idea(s).

<u>Phrase</u>	<u>Contour</u>	<u>Ascending/ Descending</u>	<u>Pitches</u>	<u>Set Class</u>	<u>Normal Form</u>
1	1 0 3 2	A	5 1 19 8	4-z29	0 1 3 7
2	1 3 0 2	A	2 17 0 11	4-13	e 0 2 5
3	2 0 1	D	22 3 14	3-4	t 2 3
4	1 0 2	A	1 0 18	3-5	0 1 6
5	1 0 2	A	4 1 24	3-3	0 1 4
6	0 2 1	A	1 9 0	3-3	9 0 1
7	0 2 1	D	8 16 14.5	N/A	2.5 4 8
8	0 2 1	D	0 31 3	3-11	0 3 7
9	0 1	A	8 35	2-3	8 e
10	0 2 1	A	3 32 6	3-7	3 6 8
11	1 0 2	A	6 0 35	3-5	6 e 0
12	1 0	D	9 8.5	2-1	1 0
13	0 1	A	2 30	2-4	2 6
14	0 1	A	4 33	2-5	4 9
15	1 3 0 2	A	4 19 2 13	4-13	1 2 4 7
16	1 0 2	A	33 4 36	3-11	9 0 4
17	1 0 2	A	33 32 34	3-1	8 9 t
18	1 0 2	A	7 4 36	3-11	0 4 7
19	0 1	A	1 28	2-3	1 4
20	0 2 1	A	0 36 35	2-1	e 0
21	1 0 2	A	10 0 28	3-8	t 0 4
22	0 1	A	0 13	2-1	0 1
23	0 1	A	33 33.5	N/A	
24	1 0 2	A	0 1 2	3-1	0 1 2
25	0 1	A	3 34	2-4	t 3

Figure 4.12: Synchronic contour results for *Itinerant*.

Stanza	Pitch Location from 1-38	Contour Prime	Stanza	Pitch Location from 39-76	Contour Prime	Stanza	Pitch Location from 77-114	Contour Prime
1	1	0	3	39	2 0 3 1		77	1 0 3 2
	2	0 1		40	1 2 0		78	1 0 3 2
	3	1 2 0		41	2 3 0 1		79	1 0 3 2
	4	1 0 2		42	1 3 0 2		80	1 0 3 2
	5	1 2 0		43	1 3 0 2		81	1 0 3 2
	6	1 0 2		44	1 0 3 2		82	2 0 3 1
	7	1 0 2		45	1 0 3 2		83	1 0 3 2
	8	1 0 3 2		46	2 0 3 1		84	2 0 3 1
	9	1 0 3 2		47	1 0 3 2		85	1 0 3 2
	10	1 0 3 2		48	1 0 3 2		86	2 0 3 1
	11	1 0 3 2		49	1 2 0		87	1 0 3 2
	12	1 0 3 2		50	1 0 3 0 2		88	1 0 3 2
	13	1 0 2		51	1 0 2		89	1 0 3 2
	14	1 0 3 2		52	1 2 0	7	90	1 0 3 2
	15	1 0 3 2		53	2 0 3 0 1		91	1 0 3 2
	16	1 0 3 2		54	1 0 3 0 2		92	1 0 3 2
	17	2 0 3 1		55	1 0 3 2		93	1 0 3 2
	18	1 0 3 2		56	1 2 0		94	1 0 3 2
	19	1 0 3 2		57	1 3 0 2		95	1 0 3 2
2	20	1 0 3 2		58	1 3 0 2		96	2 0 3 1
	21	1 0 3 2		59	1 3 0 2		97	1 0 3 2
	22	1 0 3 2		60	1 3 0 2		98	1 0 3 2
	23	1 0 3 2		61	1 2 0		99	1 0 3 2
	24	1 0 3 2	5	62	1 2 0	8	100	1 0 3 2
	25	1 0 3 2		63	1 0 3 2		101	2 0 3 1
	26	2 0 3 1		64	1 0 3 2		102	1 0 3 2
	27	1 0 3 2		65	2 0 3 1		103	1 0 3 2
	28	1 2 0		66	1 0 3 2		104	1 0 3 2
	29	2 3 0 1		67	1 0 3 2		105	1 2 0
	30	1 3 0 2		68	1 0 3 2		106	1 3 0 2
	31	1 0 2		69	1 0 3 2		107	1 3 0 2
	32	1 0 3 2		70	1 2 0		108	1 3 0 2
	33	2 0 3 1		71	1 0 2	9	109	1 3 0 2
	34	1 0 3 2		72	1 0 3 2		110	2 3 0 1
	35	1 0 3 2		73	1 0 3 2		111	1 3 0 2
	36	1 0 3 2		74	1 0 3 2		112	2 3 0 1
	37	1 0 3 2	6	75	2 0 3 1		113	1 2 0 1
	38	1 0 3 2		76	1 0 3 2		114	2 3 0 1

Figure 4.13: Diachronic contour primes from beginning to all 266 pitches of Tōru Takemitsu's *Itinerant*.

Stanza	Pitch Location from 115-152	Contour Prime	Stanza	Pitch Location from 153-190	Contour Prime	Stanza	Pitch Location from 191-226	Contour Prime
	115	1302		153	1302		191	1032
	116	1302		154	1302		192	102
	117	1302	12	155	1302		193	1032
	118	1302		156	120		194	1032
	119	2301		157	2301		195	102
	120	1302		158	2301		196	102
	121	1302		159	1201		197	102
	122	1302		160	1302	15	198	102
	123	2301		161	1302		199	1032
	124	1302		162	021		200	1032
10	125	1302		163	1302		201	1032
	126	1201	13	164	021		202	1032
	127	1302		165	021		203	1032
	128	1302		166	021		204	1302
	129	1302		167	120		205	120
	130	2301		168	021		206	1302
	131	1302		169	021		207	1302
	132	1302		170	021		208	2301
	133	1302		171	102		209	1302
	134	1302		172	120		210	1302
	135	1302		173	10	16	211	1302
	136	1302		174	01		212	1302
	137	1302		175	01		213	120
	138	2301		176	01		214	1302
	139	1302		177	01		215	1201
11	140	2301	14	178	120		216	1302
	141	1201		179	120		217	1302
	142	1302		180	120		218	1302
	143	1302		181	120	17	219	1302
	144	1302		182	120		220	1302
	145	1302		183	120		221	2301
	146	1302		184	120		222	120
	147	1302		185	120		223	2301
	148	1302		186	102		224	2301
	149	1302		187	102		225	1302
	150	1302		188	1032		226	1302
	151	2301		189	2031			
	152	1302		190	1021			

Figure 4.13: Diachronic contour primes from beginning to all 266 pitches of Tōru Takemitsu's *Itinerant* (cont).

	Total Occurrence	Occurrence Percentage
1 2 0 =	25	11%
1 0 2 =	15	7%
1 0 3 2 =	70	31%
2 3 0 1 =	17	8%
1 3 0 2 =	63	28%
2 0 3 1 =	13	6%
0 2 1 =	7	3%
1 2 0 1 =	4	2%
1 0 2 1 =	1	0%
1 0 3 0 2 =	2	1%
2 0 3 0 1 =	1	0%
0 =	1	0%
0 1 =	5	2%
1 0 =	1	0%

Descending contours =	25%
Ascending contours =	75%

Figure 4.14: Percentage of diachronic contour primes occurring diachronically in Tōru Takemitsu's *Itinerant*.

Scholar Timothy Koozin's research has been invaluable to this analysis. His research on Tōru Takemitsu has revealed general characteristics of Takemitsu's work which are apparent in both *Itinerant* and *Air*, noticeably:

1. Use of referential sets associated with Messiaen's *modes of limited transposition*.
2. A pattern of action and repose.
3. Recurring motives which gradually reveal the harmonic source from which they derive.
4. Focus on the tritone.
5. Use of SC 3-8 to migrate between octatonic and whole-tone regions.
6. Use of mode VI as an intermediary between octatonic and whole-tone progressions.
7. Use of the "fully diminished" tetrachord 4-28.
8. Use of chromatic sets that exceed mode boundaries.<sup>12</sup>

---

12. For more background on the harmonic language of Tōru Takemitsu, see Koozin 1989 and 2002.



VII-II (when combined with Phrase 3)  
 <1 0 3 2>  
 Sedentary

'W' SC 5-31 using E5 (2458e)  
 'W' SC 5-31 {578e2}

OCT-III  
 <1 3 0 2>  
 Sedentary

'W' SC 4-25 {0268}  
 'Y' SC 4-18 {e036}  
 SC 3-3 {e03}  
 Rhythmic Cell

VII-II (when combined with Phrase 1)  
 <2 0 1>  
 Oscillation

SC 3-8 {te3} TT oscillation

OCT-I  
 <1 0 2>  
 Ascent

Rhythmic cell  
 Y' SC 3-8 {46t}  
 SC 3-3 {347} SC 3-3 {913}

OCT-I  
 <1 0 2>  
 Double Ascent

SC 3-3 {347} SC 3-3 {913}

WT-I  
 <0 2 1>  
 Portamento

X  
 T<sub>7</sub>  
 Rhythmic Cell  
 SC 4-25 {248t}

WT-II  
 <0 2 1>  
 Portamento

X  
 SC 4-25 {1379}

Figure 4.15: Analysis of Tōru Takemitsu's *Itinerant* for solo flute (dynamics and text markup removed for legibility).

Takemitsu ITINERANT in Memory of Isamu Noguchi, SJ 1055

Copyright © 1989 by Schott Music Co. Ltd., Japan

All Rights Reserved

Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan

OCT-III  
<0 2 1>  
Double Ascent  
Sedentary

SC 3-8 {9e3} SC 3-8 {026}

⑧

4:3 4:3

'W' SC 5-31 {03569}

'W' SC 4-28 {02369}

'W' SC 4-28 {0369}

'Y' SC 3-8 {9e3}

norm.

OCT-III  
<0 1>  
Ascent

⑨

SC 3-8 {48t}

SC 3-8 {35e}

'W' SC 5-7 {te345}

OCT-III  
<0 2 1>  
Oscillation  
Sedentary

⑩

Tritone Oscillation

OCT-III  
<1 0 2>  
Portamento  
Oscillation  
Double Ascent

⑪

SC 3-8

H.T.

'W' SC 4-25 {0268}

'X'

TT Oscillation

port.

H.T.

OCT-III  
<1 0>  
Portamento norm.  
H.T. (N.V.)

⑫

SC 4-28 {9034}

'X'

port.

Figure 4.15: Analysis of Tōru Takemitsu's *Itinerant* for solo flute (cont).

13 <0 1>  
Oscillation  
Double Ascent

SC 5-23 {24579} diatonic

WT-I SC 6-35 {02468t}

'Y' SC 4-24 {268t}

14 <0 1>  
Ascent

Chromatic SC 4-1 {4567}

'Y' SC 3-8 {791}

15 OCT-II  
<1 3 0 2>  
Sedentary

Rhythmic Cell

'W' SC 4-18 {1257}

16 OCT-II  
<1 0 2>  
Sedentary  
Ascent

Rhythmic Cell  
SC 3-3 {034}

SC 3-2 {023}

WT-II SC 4-25 {359e}

norm.

'Y' SC 4-8 {0235}

SC 3-3 {125}

17 <1 0 2>  
Sedentary

SC 3-1 {89t}

18 OCT-I  
<1 0 2>  
Double Ascent

SC 3-2 {9t0}

Figure 4.15: Analysis of Tōru Takemitsu's *Itinerant* for solo flute (cont).

19  $\langle 0\ 1 \rangle$   
Ascent

SC 3-8 {913} SC 3-8 {48t}

20  $\langle 0\ 2\ 1 \rangle$   
Sedentary  
Ascent

SC 3-8 {680} SC 3-8 {359} SC 3-2 {t01} SC 3-2 {467} SC 3-2 {9t0}

'W' SC 5-31 {68903}

VII-III  
21  $\langle 1\ 0\ 2 \rangle$   
Portamento  
Double Ascent

H.T. port.

SC 3-1 {89t} 'Y' SC 4-18 {e036} SC 3-8 {t24}

'X' SC 3-8 {35t}

22  $\langle 0\ 1 \rangle$   
Ascent poco

SC 3-8 {791}

Rhythmic Cell

23  $\langle 0\ 1 \rangle$   
Portamento

'X'


24  $\langle 1\ 0\ 2 \rangle$   
Sedentary

25  $\langle 0\ 1 \rangle$   
Ascent

SC 6-1 {te0123}

SC 3-1 {012}

Figure 4.15: Analysis of Tōru Takemitsu's *Itinerant* for solo flute (cont).

The analysis shown in **Figure 4.15** reveals the piece’s complete motivic and gestural saturation. Tritone saturation permeates the deepest level. The tritone becomes a motivic seed for larger motivic cells, like the W and Y motives, and is a member of the most prominently used set classes, SCs 3-3, 3-8, 4-28, and 5-31. Takemitsu uses referential pitch-class set collections representative of Messiaen’s “modes,” specifically modes I, II, and VII. While large-scale pitch/rhythmic repetition does not occur, the rhythmic cell , and similar rhythms, are repeated. The largest-scale repetition occurs in phrase 15, which is a rhythmic and set class repetition of phrase 2.

## Summary

At first glance, Tōru Takemitsu’s *Itinerant* seems to be composed of phrases “more often possessing no obvious relationship” and to have pitch content that “does not appear to have been chosen based on pitch-class sets (Robinson 2011, 71).” A deeper investigation of pitch-class content, segmentation, and contour reveals a saturation of motivic and gestural material. The use of “modes,” a rhythmic cell, and phrase ideas unify the work and expose Takemitsu’s use of precompositional cells and referential pitch-class sets.

The investigation of pitch-class content, segmentation, and contour reveals a saturation of motivic and gestural material. The use of modes, a rhythmic cell, and phrase ideas unify the work and expose Takemitsu’s use of precompositional cells and referential set complexes.

## 4.4 Tōru Takemitsu: *Air* (1995)

*Air* was Takemitsu's last completed score (Burt 2006, 218). The title refers not only to the idea of his own impending last breath, but also to his emphasis on melody, giving a nod to the English air (Burt 2006, 218). Takemitsu utilizes many of the same techniques used in *Itinerant*, but chooses not to use some of the more extended performance techniques, such as multiphonics. He also uses far fewer microtones and alternate fingerings than in *Itinerant*. There are no *fermatas*. He does, however, utilize a lower range, often using the pitch B3 associated with the B-footjoint on the flute.

### Segmentation

Based on the location of *ma* and *decrescendos*, the piece can be divided into 6 sections. While the piece contains no *fermatas*, there are five 2/4 measures of rest preceded by *decrescendos* which assist the argument for this division. Beyond the appearance of *ma* there are audible repetitions of musical material which further divide the music into three recurring subsections:

- A: The combination of musical ideas with the same rhythm and interval content as measures 1-6.
- B: The combination of musical ideas with the same melodic arch motion as measures 7-14.
- C: The combination of musical ideas with the same rhythm and interval content as measures 23-27.

These subsections may return in full or only part of the original statement. The phrases within each subsection may be rearranged or transposed. The formal division of the piece, based on these criteria, is shown in **Figure 4.16**. Each subsection can be further divided into motives which are inherently tied to synchronic contour. These motives will be discussed in the next section.

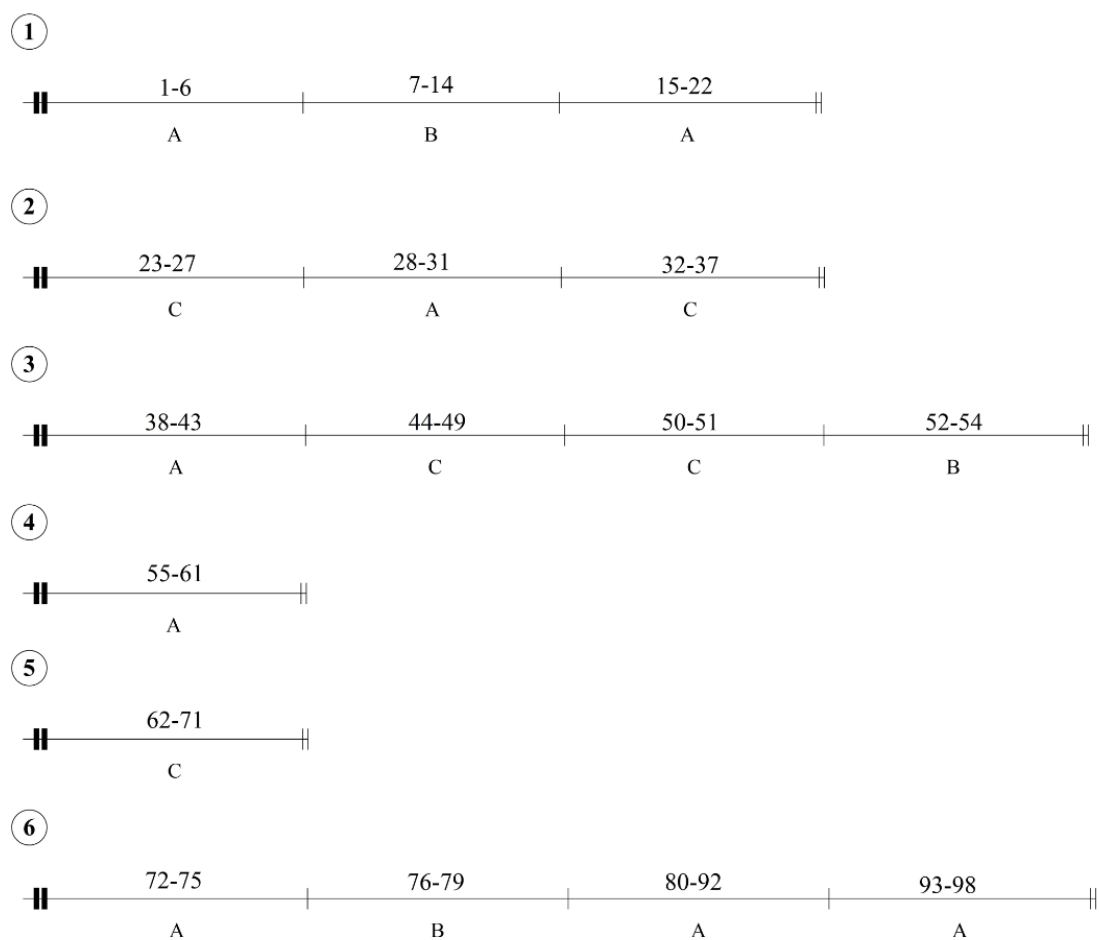


Figure 4.16: Form diagram of sections and subsections for Tōru Takemitsu's *Air* for solo flute. Sections are labeled 1-6. Subsections are labeled A, B, or C. Measure numbers are given for each subsection.

Temporal gestalt (TG) segmentation is shown in **Figure 4.17**. For segmentation, pitches were entered as semitones and duration as quarter-note beats into the TG spreadsheet. These values were then normalized to a range between 0-1. Pitch was normalized by dividing every pitch value by the number 30 (the range of the piece being -1 through 28) and duration values were divided by the number 5.91667 (the absolute value of the difference between the longest and shortest duration). TG segmentation divides the piece into 139 *clangs*, 50 *sequences*, 15 *segments* and 6 *sections*. Complete TG segmentation data for *Air* is available online at [kate.cooleysekula.net/java](http://kate.cooleysekula.net/java). TG segmentation is least-effective on this piece. The *sequence* level has the greatest correlation with the subdivisions of the subsections as described above, but the remaining levels do not correlate with the location of subsections and sections show in **Figure 4.16**. *Air* is the least post-tonal of the works discussed in this document, so it makes sense that a technique meant for post-tonal analysis might not offer satisfying results. While pitch-class content reflects use of pitch-class set collections, the prominent repetition of subsections makes audible division of the work less complicated.



The musical score is divided into four systems, each with a key signature change and a time signature change. The notation includes various musical symbols such as notes, rests, and dynamic markings. Above the staff, labels indicate the segmentation of the piece into sequences and clangs. Below the staff, dynamic markings and performance instructions are provided.

**System 1 (Measures 1-4):** Key signature: one flat, Time signature: 3/4. Labels: Section: 1, Segment: 1, Sequence: 1, Clang: 1, Clang: 2, Clang: 3, Sequence: 2, Clang: 4, Clang: 5, Clang: 6. Dynamics: *pp cresc.*, *f*, *p*, *mf*, *poco*, *p*, *mf*. Performance instructions: *sostenuto*.

**System 2 (Measures 5-8):** Key signature: two flats, Time signature: 3/4. Labels: Sequence: 3, Clang: 7, Clang: 8, Clang: 9, Clang: 10. Dynamics: *p*, *ff*, *f*, *mf*, *f*, *p*, *pp*, *p*. Performance instructions: *in Tempo*, *poco rit.*, *poco a poco accel.*.

**System 3 (Measures 9-12):** Key signature: two flats, Time signature: 3/4. Labels: Clang: 11, Sequence: 5, Clang: 12, Clang: 13, Sequence: 6, Clang: 14, Clang: 15, Sequence: 7, Clang: 16, Clang: 17. Dynamics: *p*, *mf*, *p*, *p*, *f*, *f*, *mf*. Performance instructions: *poco rall.*, *in Tempo*, *poco rit.*, *in Tempo*, *poco*, *in Tempo*.

**System 4 (Measures 13-16):** Key signature: one flat, Time signature: 3/4. Labels: Clang: 18, Sequence: 8, Clang: 19, Clang: 20, Clang: 21, Sequence: 9, Clang: 22, Clang: 23. Dynamics: *p*, *mf*, *p*, *ppp*, *pp*, *p*, *(mf)*. Performance instructions: *poco*, *sostenuto*, *in Tempo lontano*, *normal*.

**System 5 (Measures 17-20):** Key signature: one flat, Time signature: 3/4. Labels: Clang: 24, Clang: 25, Sequence: 10, Clang: 26, Clang: 27, Clang: 28. Dynamics: *sub.*, *pp*, *mf*, *p*, *mf*, *poco*, *mf*, *p*, *p*, *mf*. Performance instructions: *sub.*, *poco*, *poco*, *sostenuto*.

Figure 4.17: Segmentation of Tōru Takemitsu's *Air* for solo flute.

Takemitsu AIR dedicated to Auréle Nicolet for his 70th birthday, SJ 1096

Copyright © 1995 by Schott Music Co. Ltd., Japan

All Rights Reserved

Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan

2

Sequence: 11  
Clang: 29

Clang: 30

in Tempo

lontano  
as echo

19

*p* *f* *mf* *p* *mf* *pp*

Segment: 3  
Sequence: 12  
Clang: 31

Clang: 32

Clang: 33

Sequence: 13  
Clang: 34

quasi flutt.

Clang: 35

ord.

Clang: 36

*pp* *poco sfz* *p* *sempre* *mf* *p* *mf*

Segment: 4  
Sequence: 14  
Clang: 37

in Tempo

Clang: 38

Sequence: 15  
Clang: 39

Clang: 40

*p* *f* *mf* *mf* *p* *f*

Sequence: 16  
Clang: 41

Clang: 42

*poco rit.*

Sequence: 17  
Clang: 43

in Tempo

Clang: 44

Clang: 45

*sub. p* *p* *mf* *pp* *mf* *pp*

Sequence: 18  
Clang: 46

Clang: 47

*poco rit.*

Section: 2  
Segment: 5  
Sequence: 19  
Clang: 48

in Tempo

Clang: 49

Sequence: 20  
Clang: 50

*p* *mf* *p* *f* *mf* *p*

Figure 4.17: Segmentation of Tōru Takemitsu's *Air* for solo flute (cont).

158

52

Sequence: 29  
Clang: 77

Clang: 78

Sequence: 30  
Clang: 79

Clang: 80

Clang: 81

*pp* *mf* *p* *f* *f*

55

in Tempo

Sequence: 31  
Clang: 82

Clang: 83

Clang: 84

Segment: 9  
Clang: 86

Sequence: 32  
Clang: 85

Sequence: 33  
Clang: 87

*p* *mf* *p* *mf* *p* *mf* *p* *f*

*poco rit.* *in Tempo* *rit.* *in Tempo*

58

Sequence: 34  
Clang: 89

Clang: 88

Clang: 90

Sequence: 35  
Clang: 91

*poco* *mf* *pp* *mf* *p* *pp* *al niente* *mf* *p*

62

Clang: 92

Clang: 93

Clang: 94

Clang: 95

*piu p* *mf* *sub. pp* *al niente* *pp* *p* *pp* *mf* *p* *p*

65

sostenuto

Clang: 96

Section: 4  
Segment: 10  
Sequence: 36  
Clang: 97

in Tempo

Clang: 98

Sequence: 37  
Clang: 99

Clang: 100

*p* *mf* *pp* *sub. f* *mf* *p*

Figure 4.17: Segmentation of Tōru Takemitsu's *Air* for solo flute (cont).

Segment: 11  
Sequence: 38  
Clang: 101

in Tempo ord.

Clang: 102

69

*pp* *al niente* *ppp* *pp* *p* *mf*

Sequence: 39  
Clang: 104  
Clang: 105

Clang: 103

73

*pp* *mf p* *mf* *mf*

*poco rit.* *in Tempo* *rit. poco* *in Tempo*

Sequence: 41  
Clang: 111  
Clang: 112

Sequence: 42  
Clang: 114  
Clang: 115

Sequence: 43  
Clang: 116

Clang: 108 Clang: 109 Clang: 110

76

*più p* *mf* *p* *f* *mf* *f* *f* *ff* *p*

*poco* *poco*

Clang: 117 Clang: 118 Clang: 119

Clang: 120

Clang: 121

80

*p* *f* *(p)* *mf* *p* *p*

*poco* *poco*

Section: 5  
Segment: 13  
Sequence: 44  
Clang: 122

Sequence: 45  
Clang: 124

Clang: 125

in Tempo *lontano* *lontano*

Clang: 123

83

*p* *pp* *ff* *f* *(ppp)* *mf* *p*

*poco*

Figure 4.17: Segmentation of Tōru Takemitsu's *Air* for solo flute (cont).

6

Segment: 14  
Sequence: 46  
Clang: 126

Sequence: 47  
Clang: 128

Clang: 127

*poco sostenuto* in Tempo *ord.*

Clang: 129

87

*al niente* *pp* *p* *mf* *ppp* *p* *mf*

Clang: 130

*sostenuto*

Clang: 131

*poco* *mf* *p* *pp* *al niente*

90

Section: 6  
Segment: 15  
Sequence: 48  
Clang: 132

*in Tempo*  $\bullet = \text{ca. } 54$

Clang: 133

Clang: 134

Sequence: 49  
Clang: 135

Clang: 136

Clang: 137

93

*pp cresc* *f* *p* *mf* *p* *p* *mf*

Sequence: 50  
Clang: 138

*in Tempo*

Clang: 139

*poco rit.*

96

*p* *ff* *f* *mf* *poco f* *p* *al niente*

Figure 4.17: Segmentation of Tōru Takemitsu's *Air* for solo flute (cont).

## Pitch Class Content

Several additional abbreviations are employed (in addition to those used in the analysis of *Itinerant*) for the analysis of *Air*:

Mode III-I = Messiaen's mode III, transposition I (01245689t)

Mode III-II = Messiaen's mode III, transposition II (1235679te)

Mode III-III = Messiaen's mode III, transposition III (234678te0)

Mode III-IV = Messiaen's mode III, transposition IV (01345789e)

Mode VI-I = Messiaen's mode VI, transposition I (0135679e)

Mode VI-II = Messiaen's mode VI, transposition II (0124678t)

Mode VI-III = Messiaen's mode VI, transposition III (1235789e)

Mode VI-IV = Messiaen's mode VI, transposition IV (234689t0)

*Air* for solo flute is another example of Messiaen's and Debussy's influence on Takemitsu. In this piece, Takemitsu relies on modes II (octatonic), III (three chromatic trichords separated by a whole-step), and VII (two chromatic hexachords separated by a whole step). The location of each mode's use is shown in **Figure 4.18**. This analysis is by no means the only possible explanation for pitch-class set collection use in this piece. Within these larger pitch-class set collection, Takemitsu weaves use of the octatonic and whole tone collections.

Certain pitch-classes are more prominent than others, especially pc 9, or A, which represents 20% of the piece's duration, measured in quarter notes. The next longest combined pitch duration is pc 5 at 8%. Pc 9 is often paired with the pitches E and E $\flat$ , a motive which can be transformed to spell *S-E-A* (borrowing from the German Es to form *E  $\flat$ -E-A*) (Takemitsu et al. 1995, 112). This motive appears throughout Takemitsu's works. It recurs throughout *Air*, often embellished, such as in gesture "d" (described below) in measures 4-6, or respelled enharmonically, again as in measures 4-6, or with pc 9 transposed down a half-step to pc 8.

Figure 4.18: Analysis of Tōru Takemitsu’s *Air* for solo flute (dynamics and text markup removed for legibility). Pitch-class set collection use is marked above the staff. Motivic gestures a-d are labeled below the staff.

Takemitsu AIR dedicated to Auréle Nicolet for his 70th birthday, SJ 1096

Copyright © 1995 by Schott Music Co. Ltd., Japan

All Rights Reserved

Used by Permission of European American Music Distributors Company, Sole U.S. and Canadian agent for Schott Music Co. Ltd., Japan



2

19

d

d

3

d

OCT III

23

6

a

b

3

3

Mode III-IV

OCT-I

26

d

5

a

a

WT-II

OCT-III

29

3

5

a

b

6

b

6

Mode III-IV

Mode III-III

Mode VI-III

Mode III-III

31

3

c

6

c

b

5

a

5

Figure 4.18: Analysis of Tōru Takemitsu's *Air* for solo flute (cont).

OCT-III

34

a b c

Mode VI-III

37

a d

Mode III-IV

41

a a a

Mode III-I

OCT I

45

b a a a

Mode III-II

49

a b a

Detailed description: The image shows a musical score for solo flute, specifically an analysis of Tōru Takemitsu's *Air*. The score is presented in five systems, each with a label above indicating a mode or section. Measure numbers are placed to the left of each system. Brackets labeled with letters 'a', 'b', 'c', and 'd' are used to group specific melodic phrases or techniques. Fingerings (3, 5, 6) and slurs are indicated throughout the score.

- System 1 (Measures 34-36):** Labeled "OCT-III". Measures 34 and 35 contain phrases grouped under brackets 'a' and 'b' respectively. Measure 36 contains a phrase grouped under bracket 'c'. Fingerings 3, 5, and 6 are indicated.
- System 2 (Measures 37-38):** Labeled "Mode VI-III" and "Mode III-IV". Measure 37 contains a phrase grouped under bracket 'a'. Measure 38 contains a phrase grouped under bracket 'd'. A fingering of 5 is indicated in measure 38.
- System 3 (Measures 41-43):** Labeled "Mode III-I". Measure 41 contains a phrase grouped under bracket 'a'. Measure 42 contains a phrase grouped under bracket 'a'. Measure 43 contains a phrase grouped under bracket 'a'. Fingerings 3 and 6 are indicated.
- System 4 (Measures 45-48):** Labeled "OCT I" and "Mode III-I". Measure 45 contains a phrase grouped under bracket 'b'. Measure 46 contains a phrase grouped under bracket 'a'. Measure 47 contains a phrase grouped under bracket 'a'. Measure 48 contains a phrase grouped under bracket 'a'. A fingering of 5 is indicated in measure 46.
- System 5 (Measures 49-51):** Labeled "Mode III-II". Measure 49 contains a phrase grouped under bracket 'a'. Measure 50 contains a phrase grouped under bracket 'b'. Measure 51 contains a phrase grouped under bracket 'a'. Fingerings 6 and 5 are indicated.

Figure 4.18: Analysis of Tōru Takemitsu's *Air* for solo flute (cont).

4

Mode VII-II

52

b b

OCT-III WT-I Mode III-IV

55

a b a

Mode VII-II Mode III-IV

58

c b d

62

a b a a

65

a a b

Mode III-I

Figure 4.18: Analysis of Tōru Takemitsu's *Air* for solo flute (cont).

The musical score is divided into five systems, each with a key signature of one flat (B-flat) and a treble clef. The measures are numbered 69, 73, 76, 80, and 83 at the beginning of their respective systems.

- System 1 (Measures 69-72):** Labeled "Mode VII-II". Measure 69 contains a half note G4, a half note F4, and a half note E4. Measure 70 is a whole rest. Measure 71 is a whole rest. Measure 72 contains a half note G4, a half note F4, and a half note E4. A bracket labeled "a" spans measures 71 and 72.
- System 2 (Measures 73-75):** Labeled "OCT-I". Measure 73 contains a half note G4, a half note F4, and a half note E4. Measure 74 contains a half note G4, a half note F4, and a half note E4. Measure 75 contains a half note G4, a half note F4, and a half note E4. Brackets labeled "c" and "b" are placed under measures 73-74 and 74-75 respectively.
- System 3 (Measures 76-79):** Labeled "OCT-I", "Mode III-III", and "Mode III-IV". Measure 76 contains a half note G4, a half note F4, and a half note E4. Measure 77 contains a half note G4, a half note F4, and a half note E4. Measure 78 contains a half note G4, a half note F4, and a half note E4. Measure 79 contains a half note G4, a half note F4, and a half note E4. Brackets labeled "b" and "b" are placed under measures 77-78 and 78-79 respectively.
- System 4 (Measures 80-83):** Labeled "Mode III-I". Measure 80 contains a half note G4, a half note F4, and a half note E4. Measure 81 contains a half note G4, a half note F4, and a half note E4. Measure 82 contains a half note G4, a half note F4, and a half note E4. Measure 83 contains a half note G4, a half note F4, and a half note E4. Brackets labeled "a", "a", and "a" are placed under measures 80-81, 81-82, and 82-83 respectively.
- System 5 (Measures 83-86):** Labeled "Mode III-IV". Measure 83 contains a half note G4, a half note F4, and a half note E4. Measure 84 contains a half note G4, a half note F4, and a half note E4. Measure 85 contains a half note G4, a half note F4, and a half note E4. Measure 86 contains a half note G4, a half note F4, and a half note E4. Brackets labeled "c", "d", and "d" are placed under measures 83-84, 84-85, and 85-86 respectively.

Figure 4.18: Analysis of Tōru Takemitsu's *Air* for solo flute (cont).

6

OCT-III

87

a b d

Mode III-IV

90

d b

Mode VI-IV Mode III-IV

93

a b c c

96

d d

Figure 4.18: Analysis of Tōru Takemitsu's *Air* for solo flute (cont).

## Motivic and Gestural Ideas

*Air*, like *Itinerant* is highly gestural. The gestures are much clearer than those in *Itinerant* and exact repetition is often used. There are four main gestures, each identified by a specific surface contour (i.e., a contour apparent on the the musical surface without reduction) and its related truncations, transpositions or inversions:

a = Contour  $\langle 5\ 2\ 4\ 1\ 3\ 0 \rangle$  and its retrogression and inversion as well as the truncation  $\langle 0\ 2\ 1\ 3 \rangle$  and its retrogression and inversion.

b = Ascent/Descent contour  $\langle 0\ 1\ 2\ 3\ 2\ 1\ 0 \rangle$  or ascent contour  $\langle 0\ 1\ 2\ 3 \rangle$  or descent contour  $\langle 3\ 2\ 1\ 0 \rangle$ .

c = Contour  $\langle 0\ 3\ 2\ 5\ 1\ 4 \rangle$ .

d = Contour  $\langle 3\ 0\ 2\ 1 \rangle$  or its retrograde inversion  $\langle 2\ 1\ 3\ 0 \rangle$ .

Gesture “a” generally results in SC 6z-19 or, in its truncated version, SC 4-19. Gesture “c” generally results in SC 6-z39 and gesture “d” generally results in SC 4-z29 or 4-12. Gesture “b” produces a wider array of set classes, ranging from three to six pitches. Sometimes it is presented as a complete arch of ascent/descent. At other times only an ascending or descending motion is used. The complete use of these gestures is shown in **Figure 4.18**.

## Contour

The use of small-scale surface contour analysis is integral to the identification of musical gesture. Small, synchronic contours are tied to each gestural idea, as discussed above. Speaking diachronically, one contour prime is the most predominant:  $\langle 1\ 0\ 3\ 2 \rangle$  represents 34% of all contours (note that this same contour was also predominant in *Itinerant*, representing 31% of all contours). It is also the overall contour prime of the piece. Contour data for *Air* is shown in **Figures 4.19** through **4.20**.

Measure	Pitch location from 1-40	Contour Prime
1	1	0
	2	1 0
	3	1 0 2
	4	1 0 2 1
	5	1 0 2
	6	1 0 3 2
2	7	1 0 2
	8	1 0 3 2
	9	1 0 3 2
	10	1 0 3 2
	11	1 0 3 2
	12	1 0 2 1
3	13	1 2 0
	14	2 0 3 0 1
	15	2 0 3 1
	16	1 0 3 2
	17	2 0 3 1
	18	1 0 2 1
	19	1 0 3 2
	20	1 0 3 2
	21	1 0 3 2
4	22	1 0 2 1
	23	1 0 2
	24	1 0 3 2
	25	1 0 3 2
5	26	1 0 3 2
	27	1 0 2
	28	1 0 3 2
	29	1 0 2
6	30	1 0 2
7	31	1 0 3 2
	32	1 0 3 2
	33	1 0 2
	34	1 0 2
	35	1 0 2
	36	1 0 2
	37	1 0 2
	38	1 0 3 2
8	39	1 0 2 1
	40	1 0 3 2

Measure	Pitch location from 41-80	Contour Prime
	41	1 0 2
	42	1 0 2
	43	1 0 2
	44	1 0 3 2
	45	1 0 3 2
	46	1 0 3 2
9	47	1 2 0
	48	1 0 3 0 2
	49	0 1 2 1 0
	50	1 0 3 0 2
	51	1 0 3 2
	52	1 0 3 0 2
10	53	1 2 0
	54	2 3 0 1
	55	2 3 0 1
	56	1 3 0 2
	57	2 3 0 1
	58	1 3 0 2
11	59	1 3 0 2
	60	1 3 0 2
	61	1 3 0 2
	62	1 3 0 2
	63	1 2 0 1
	64	1 3 0 2
12	65	2 3 0 1
	66	1 2 0
	67	1 2 0 1
13	68	2 3 0 1
	69	1 3 0 2
14	70	1 3 0 2
	71	1 3 0 2
	72	1 3 0 2
15	73	2 3 0 1
	74	2 3 0 1
	75	2 3 0 1
	76	1 3 0 2
	77	2 3 0 1
	78	1 2 0 1
16	79	1 3 0 2
	80	1 3 0 2

Measure	Pitch location from 81-120	Contour Prime
	81	1 3 0 2
	82	1 2 0 1
17	83	1 3 0 2
	84	1 3 0 2
	85	1 3 0 2
	86	1 3 0 2
	87	1 3 0 2
	88	1 2 0 1
18	89	2 3 0 1
	90	2 3 0 1
	91	2 3 0 1
	92	1 3 0 2
	93	2 3 0 1
	94	1 2 0 1
	95	1 3 0 2
	96	1 3 0 2
	97	1 3 0 2
19	98	1 2 0 1
	99	1 0 2
	100	1 0 3 2
	101	1 0 3 2
20	102	1 0 3 2
	103	1 0 2
	104	1 0 3 2
	105	1 3 0 2
21	106	1 3 0 2
22	107	1 2 0
	108	2 3 0 1
	109	2 3 0 1
	110	2 3 0 1
	111	2 3 0 1
	112	2 3 0 1
	113	2 3 0 1
23	114	1 3 0 2
	115	1 3 0 2
	116	1 2 0 1
	117	2 3 0 1
	118	2 3 0 1
	119	1 2 0
24	120	2 3 0 1

Figure 4.19: Diachronic contour primes from beginning to all 472 pitches of Tōru Takemitsu's *Air*.

Measure	Pitch location from 121-160	Contour Prime	Measure	Pitch location from 161-200	Contour Prime	Measure	Pitch location from 201-240	Contour Prime
	121	2 3 0 1		161	2 0 1		201	1 2 0
25	122	1 2 0		162	1 0 2		202	0 1
26	123	1 0 2		163	2 0 1	38	203	0 1
	124	1 0 3 2		164	1 0 1		204	0 2 1
	125	1 0 3 2		165	2 0 1		205	0 2 1
	126	1 0 3 2	31	166	1 0 2		206	0 2 1
	127	1 0 3 2		167	1 0 2		207	0 2 1
	128	1 0 3 2		168	1 0 2		208	1 2 0
	129	1 0 3 2		169	1 0 2	39	209	0 2 1
	130	2 0 1		170	2 0 1		210	1 2 0
27	131	2 0 3 1		171	1 0 1		211	1 2 0 1
	132	1 0 2		172	1 0 2		212	2 3 0 1
	133	1 0 1		173	1 0 2		213	2 3 0 1
	134	1 0 2		174	1 0 2	40	214	2 3 0 1
	135	1 0 3 2	32	175	2 0 1		215	2 3 0 1
	136	1 0 2		176	2 0 1		216	2 3 0 1
28	137	1 0 3 2		177	2 0 1		217	2 0 3 1
	138	1 0 3 2		178	1 0 2		218	2 0 3 1
	139	1 0 3 2	33	179	1 0		219	2 0 3 1
	140	1 0 3 2		180	2 0 1		220	2 0 3 1
	141	1 0 3 2		181	2 0 1		221	1 0 3 2
	142	1 0 2 1		182	2 0 1		222	1 0 2 1
	143	1 0 3 2		183	2 0 1		223	1 0 3 2
	144	1 0 3 2		184	2 0 1		224	1 0 3 2
	145	1 0 3 2		185	2 0 1		225	1 0 3 2
	146	1 0 2 1		186	1 0 2	41	226	2 0 3 1
29	147	1 0 3 2	34	187	1 0 2		227	1 0 3 2
	148	1 0 3 2		188	0 2 1		228	2 0 3 1
	149	1 0 3 2		189	0 2 1		229	1 0 3 2
	150	1 0 3 2		190	0 2 1		230	1 0 3 2
	151	1 0 3 2		191	0 2 1		231	1 0 3 2
	152	1 0 3 2	35	192	0 2 1	42	232	1 0 3 2
	153	1 0 2		193	0 1 0	43	233	2 0 3 1
	154	1 0 2		194	0 2 1		234	1 0 3 2
	155	1 0 2		195	0 2 1		235	1 0 3 2
	156	1 0 2		196	0 2 1		236	1 0 3 2
	157	1 0 2		197	0 2 1	44	237	2 0 3 1
	158	1 0 1	37	198	1 2 0		238	2 0 3 1
30	159	2 0 1		199	1 2 0		239	1 0 2 1
	160	2 0 1		200	1 2 0		240	1 0 3 2

Figure 4.19: Diachronic contour primes from beginning to all 472 pitches of Tōru Takemitsu's *Air* (cont).



Measure	Pitch location from 241-280	Contour Prime	Measure	Pitch location from 281-320	Contour Prime	Measure	Pitch location from 321-360	Contour Prime
	241	1 0 3 2		281	1 0 3 2		321	2 0 3 1
45	242	2 0 3 1		282	1 0 3 2		322	1 0 3 2
	243	2 0 3 1		283	1 0 2		323	1 0 3 2
	244	2 0 3 1	54	284	1 0		324	1 0 3 2
	245	1 0 2 1		285	1 0 1		325	1 0 3 2
46	246	2 0 3 1		286	2 0 1		326	1 0 3 2
	247	2 0 3 1		287	1 0 2	63	327	1 0 3 2
	248	2 0 3 1		288	2 0 1		328	1 0 3 2
	249	1 0 2 1		289	1 0 2		329	1 0 3 2
47	250	1 0 3 2		290	1 0 2		330	1 0 3 2
	251	1 0 3 2	55	291	1 0 2		331	1 0 3 2
48	252	1 0 3 2		292	1 0 2	64	332	2 0 3 1
	253	1 0 2		293	2 0 1		333	1 0 3 2
49	254	1 2 0		294	2 0 1		334	1 0 3 2
	255	2 0 1	56	295	1 0 2	65	335	1 0 3 2
	256	2 0 1		296	1 0 3 2	66	336	1 0 3 2
	257	2 0 1		297	1 0 3 2		337	1 0 3 2
	258	1 0 2		298	1 0 3 2		338	1 0 3 2
	259	1 0 3 2		299	1 0 3 2		339	1 0 3 2
	260	2 0 3 1		300	2 0 3 1		340	1 0 3 2
	261	1 2 0	57	301	1 2 0	67	341	1 2 0
	262	1 0 2		302	2 3 0 1		342	2 3 0 1
	263	1 0 2 1		303	2 3 0 1		343	2 3 0 1
	264	1 0 3 2		304	2 3 0 1		344	2 3 0 1
50	265	2 0 3 1		305	2 0 3 1		345	1 3 0 2
51	266	1 2 0		306	2 0 3 1	68	346	1 3 0 2
	267	2 0 1	58	307	1 0 3 2	71	347	1 0 2
	268	2 0 1		308	1 0 3 2		348	0 2 1
	269	1 0 1		309	2 0 3 1		349	0 2 1
	270	1 0 2		310	2 0 3 1		350	0 2 1
	271	1 0 2	59	311	1 0 3 2		351	1 2 0
	272	1 0 3 2		312	1 0 3 2	72	352	1 2 0
	273	2 0 3 1		313	1 0 3 2		353	2 3 0 1
	274	2 0 3 1		314	1 0 3 2		354	2 0 3 1
52	275	2 0 3 1	61	315	2 0 3 1		355	2 0 3 1
	276	2 0 3 1		316	1 0 3 2		356	2 0 3 1
	277	1 0 3 2		317	1 0 3 2		357	2 0 3 1
	278	1 0 3 2		318	1 0 3 2		358	1 0 3 2
	279	1 0 3 2	62	319	2 0 3 1		359	1 0 3 2
	280	1 0 3 2		320	2 0 3 1		360	2 0 3 1

Figure 4.19: Diachronic contour primes from beginning to all 472 pitches of Tōru Takemitsu's *Air* (cont).

Measure	Pitch location from 361-400	Contour Prime	Measure	Pitch location from 401-440	Contour Prime	Measure	Pitch location from 441-472	Contour Prime
73	361	2 0 3 1	81	401	1 0 3 2		441	1 0 3 2
74	362	1 0 3 2		402	1 0 3 2	91	442	1 0 3 2
	363	1 0 3 2		403	1 0 3 2	92	443	1 0 2 1
	364	2 0 3 1		404	1 0 3 2		444	2 0 3 1
	365	2 0 3 1		405	1 0 3 2		445	1 0 3 2
75	366	1 2 0		406	1 0 2 1		446	1 0 2 1
	367	1 2 0	82	407	2 0 3 1		447	1 0 3 2
	368	2 3 0 1		408	2 0 3 1		448	1 0 3 2
	369	2 3 0 1		409	2 0 3 1	93	449	1 0 3 2
	370	2 3 0 1		410	1 0 3 2		450	1 0 3 2
	371	2 3 0 1		411	2 0 3 1		451	1 0 3 2
	372	2 3 0 1		412	1 0 2 1		452	1 0 3 2
	373	1 3 0 2	83	413	1 0 2		453	1 0 3 2
76	374	2 3 0 1		414	1 0 3 2		454	1 0 2 1
	375	2 3 0 1		415	1 0 3 2	94	455	2 0 3 1
	376	2 3 0 1	84	416	1 0 3 2		456	2 0 3 1
	377	1 3 0 2	85	417	1 0 2		457	2 0 3 1
	378	1 3 0 2		418	1 0 3 2		458	1 0 3 2
	379	1 3 0 2		419	1 0 3 2		459	2 0 3 1
77	380	0 2 1		420	1 0 3 2		460	1 0 2 1
	381	0 2 1	87	421	1 0 3 2		461	1 0 3 2
	382	0 2 1		422	1 0 2 1		462	1 0 3 2
	383	1 3 0 2		423	1 0 3 2		463	1 0 3 2
	384	1 3 0 2		424	2 0 3 1	95	464	1 0 2 1
78	385	1 0 2		425	2 0 3 1		465	1 0 2
	386	0 2 1		426	2 0 3 1		466	1 0 3 2
	387	0 2 1		427	1 0 2 1		467	1 0 3 2
	388	1 2 0		428	1 0 3 2	96	468	1 0 3 2
79	389	2 3 0 1		429	1 0 3 2		469	1 0 2
	390	1 3 0 2	88	430	1 0 3 2		470	1 0 3 2
	391	1 2 0 1		431	1 0 3 2		471	1 0 3 2
	392	1 3 0 2		432	1 0 3 2	97	472	1 0 3 2
	393	1 0 3 2		433	1 0 3 2			
	394	1 3 0 2		434	1 0 3 2			
	395	1 0 3 2	89	435	1 0 3 2			
80	396	1 0 3 2		436	1 0 3 2			
	397	1 0 3 2		437	1 0 3 2			
	398	1 0 3 2	90	438	1 0 3 2			
	399	1 0 3 2		439	1 0 3 2			
	400	1 0 2 1		440	1 0 3 2			

Figure 4.19: Diachronic contour primes from beginning to all 472 pitches of Tōru Takemitsu's *Air* (cont).

	Total Occurrence	Occurrence Percentage
01 =	2	0%
120 =	23	5%
102 =	55	12%
1032 =	160	34%
2301 =	44	9%
1302 =	39	8%
2031 =	54	11%
021 =	22	5%
1021 =	22	5%
10 =	3	1%
201 =	24	5%
1201 =	10	2%
101 =	6	1%
20301 =	1	0%
10302 =	3	1%
01210 =	1	0%
010 =	1	0%

Figure 4.20: Percentage of contour prime occurring diachronically in Tōru Takemitsu's *Air*

## Summary

*Air* for solo flute is a piece composed of musical gestures. In total, there are four different gestures which Takemitsu replicates throughout the work. These replications can be exact, as in the final two stanzas of the work, or can include any number of alterations, such as rearrangement of the gestures, shortened gestures or presentation at a different pitch level. While the TG segmentation does not offer any enlightenment to the division of the work, the use of small-scale surface contour analysis is integral to the identification of musical gesture.

# Chapter 5

## Conclusion

This dissertation set out to demonstrate the collaborate relationship between computer programming and music analysis. This discussion began with a review of previous and current scholarly activity on the concepts of contour reduction, segmentation theory and computer analysis. Specifically, this document used the methods of Robert Morris, Rob Schultz, James Tenney and Larry Polansky.

Robert Morris introduced the terms of c-space, cp, cseg, etc. He combined the functionality of a COM-matrix and  $INT^+$  with transformational properties to determine equivalence between contours of the same cardinality. In a later article, he analyzed contours of any length with the CRA. Through employment of the CRA he determined different hierarchical contour levels and contour complexity. Rob Schultz reconfigured the CRA to repair methodological errors. He then combined contour theory with phenomenology and genealogy to produce a diachronic analysis of contour which reveals the transformational nature of a contour through time.

Segmentation is the procedure for determining how a musical work is divided into structurally significant musical units. It is a bottom-up, linear process which creates the smallest units first, and then seeks out relationships between those units to create higher levels of segmentation. It is a highly context-sensitive process of

trial and error, in which analysis of the various parameters of a musical surface must be paired with the listener's aural interpretation. Based on experimentation with the work of Larry Tenney and James Polansky, the author determined a reworking of the temporal gestalt segmentation algorithm which provided both clarification and simplification of the original algorithm. The parameters of intensity and tempo, previously experimented with by other authors, proved to have no significant influence on the segmentation of the works discussed herein.

Two computer programs were created for music analysis, each using a different platform. First, a segmentation Microsoft Excel spreadsheet was created that determined the temporal gestalt segmentation of a piece once initial pitch and time data is entered by the user. Then, a Java World Wide Web-based application was created to determine contour reduction.

In the future, the author plans to continue to develop open source applications for music analysis, and to branch out into music theory education applications. While this dissertation only selected four works for analysis, continued analysis of all works for solo flute could reveal large-scale relationships between these and other twentieth-century pieces. This extended analytical work could then be paired with performance practice to help students and performers alike in their understanding of such difficult repertoire.

The works analyzed in this dissertation coordinated the four areas of pitch-class set theory, temporal gestalt segmentation, contour reduction and general analysis of the musical surface to create a well-informed description of each piece. General analysis of the musical surface of all four pieces revealed divisions based on silence, use of *fermatas* and placement of *decrescendos*. It also revealed relationships based on repetition and use of gesture.

Use of pitch-class set theory revealed relationships based on pitch-class set collections, modes of limited transposition and seed motives related to the trichord.

Temporal gestalt segmentation was based on the parameters of pitch and duration and led to the determination of disjunction values. These values determined segmentations at the levels of *clang*, *sequence*, *segment* and *section*. These segmentations were then used to confirm or deny divisions based on analysis of the musical surface.

Contour analysis took on many guises. Surface synchronic contours were found to correlate with musical gestures. Contour analysis also revealed relationships which moved from the smallest, micro level to the entire piece. Instances of contour embedment revealed relationships between overall synchronic contours and the contours of sections and subsections. Two new contour tools proved useful: determination of diachronic contour percentage and . Determining diachronic contour percentages found further relationships between overall synchronic contours and the rate of occurrence of diachronic contours. Contour shade may yet be shown to be a useful analytical tool. In Kazuo Fukushima's *Requiem* it helped determine large-scale division of the piece.

When used in concert, these analytical tools have provided a satisfactory formal and pitch-class analysis of these works. The use of computing to automate the tasks of segmentation and contour reduction allowed the author to focus primarily on analysis of the music rather than on the tedious sub-processes which lead to the eventual acts of segmentation and contour reduction. This has resulted in the combination of computer programming with segmentation, contour reduction and pitch-class set analysis leading to the production of informed musical interpretations of each piece.

# Glossary

## Glossary

**3-Window Algorithm:** A contour reduction process developed by Mustafa Bor in which each contour pitch in a contour segment is evaluated in comparison to the preceding and succeeding contour pitches.

**5-Window Algorithm:** A contour reduction process developed by Mustafa Bor in which each contour pitch in a contour segment is evaluated in comparison to the two preceding and two succeeding contour pitches.

**access modifier:** A modifier that controls the level of access the given source code has to other areas of source code in a computer program.

**acciaccatura:** A short grace note, typically notated with a slash through the stem or flag, that is a non-harmonic note sounded at the same time as the primary harmonic note or notes and either quickly resolves to the main note or is immediately released.

**add-on:** An additional computer program that enhances the computer program it is added to.

**adjusted contour mutual embedding:** (ACMEMB) A tally of the total number of mutually embedded contour subsegments within any two contour segments represented as a value between 0 to 1.

**aggregate:** A collection of twelve chromatic pitches.

**algorithm:** A finite step-by-step process for solving a problem that frequently involves repetition of an operation.

**Apache Commons:** A collection of open source reusable Java components, including tools for reading file bytecode and for managing computer input/output.

**application:** (app) Short for application software. A computer program that performs a task on a computer.

**array:** An ordered arrangement of data elements of the same type.

**ascent relation:** (C<sup>+</sup>) The relationship (higher, lower, same) between two adjacent contour pitches.

**ascent relation matrix:** (C<sup>+</sup> matrix) Ian Quinn's version of a COM-matrix. It is the same as a COM-matrix, except it only represents whether there was an ascent (1) or no ascent (0).

**ascent relation similarity:** (C<sup>+</sup>SIM) determined by computing the similarity function between a contour member and a contour group's average. Contours with a C<sup>+</sup>SIM above a certain threshold, determined by the analyst, are considered to have a strong relationship to the average contour.

**associative orientation:** The orientation concerned with relational properties conferred by repetition, equivalence, and similarity.

**atonal:** see post-tonal.

**average transformed voice pair interval class set:** (average-TVPicset) The determination of the compactness of a vertical sonority.

**boolean:** A logical combinatorial system that represents relationships symbolically by the logical operators AND, OR, and NOT between entities.

**boundary interval:** (BI) The difference between the final *element* of a *clang* (or any other higher level) and the initial *element* of the next *clang* (or any other higher level).

**build automation:** The automatic completion of compiling, packaging, testing and



deploying source code.

**build system:** A set of software tools designed to automate the process of computer program compilation with the objective of efficiently creating an executable.

**byte:** Short for binary table. The common unit of computer storage from desktop computer to mainframe.

**bytecode:** An intermediate language that is executed by a runtime computer program. Human-readable source code is encoded to bytes which are then read by the Java virtual machine.

**call:** A statement in source code that requests services from another computer program.

**cardinality:** The number of items in a pitch-class set.

**cascading style sheet:** (CSS) A programming language used for describing the look and formatting of a document written in a HyperText Markup Language.

**central processing unit:** (CPU) The hardware within a computer that carries out the instructions of the computer program by performing the basic arithmetical, logical, and input/output operations of the system.

**chain:** A series of like orientations such as “up,” “down,” and “same” that describe the movement from one interval pitch to another.

**clang:** A temporal gestalt unit that is made up of two or more *elements*.

**class:** A programmer-defined data type that defines a collection of objects that share the same characteristics, implements methods, and can create objects of a given type.

**class-based:** A style of object-oriented computer programming in which inheritance is achieved by defining classes of objects.

**client:** An end user’s computer, generally a Windows, Mac, or Linux desktop or a laptop, smartphone, or tablet, that is connected to a network.

**coding:** Writing statements in a programming language. Synonymous with computer

programming.

**coincidence:** When two or more criteria identify a segment.

**combinatorial:** A term used in serial music for an aggregate created of subsets, one of which is considered a prime and the others which are derived from the prime by the application of basic twelve-tone operations.

**comparison matrix:** (COM-matrix) A two-dimensional representation of the interval succession/contour adjacency series results that compares a contour segment to itself.

**compile:** The act of transforming source code written in a computer programming language into another computer language, most commonly to create an executable computer program.

**complement:** Given a universal set of all twelve pitch-class integers and a selection from said set of a cardinality less than twelve, the complement is the remainder of pitch-classes not chosen from the universal set.

**composite segment:** A segment formed by contour subsegments that are contiguous or that are otherwise linked in some way.

**computer program:** (program) A set of instructions that tells a computer what to do.

**computer programming:** (programming) The process that leads from an original formulation of a computing problem to an executable computer program.

**computer programming language:** (language) A formal vocabulary designed to communicate instructions to a machine, particularly a computer. Computer programming languages can be used to create computer programs that control the behavior of a machine and/or to express algorithms precisely.

**computing:** A goal-oriented activity requiring computers.

**concurrent:** A form of computing in which computer programs are designed as collections of interacting computational processes that may be executed in

parallel.

**contextual criterion:** A basic type of criterion that responds to repetitions in a certain respect between two or more groupings of notes within a specific musical context.

**contextual domain:** The domain concerned with repetition, association, and categorization.

**contour:** The function that tracks a musical parameter over time.

**contour adjacency series:** (CAS) An ordered series of +s and -s corresponding to moves upward and downward in a musical unit.

**Contour Analysis Tools:** (CAT) A web application designed by Kate Sekula for the analysis and reduction of contours.

**contour class:** (CC) The position of contour pitches (high or low) in relation to each other.

**contour depth:** The number of times one traverses through the contour reduction algorithm until the contour prime is obtained.

**contour embedding:** (CEMB) A tally of the number of times a smaller contour segment occurs as a contour subsegment of a larger contour segment represented as a value between 0 and 1 where 0 indicates oppositeness and 1 indicates equivalence.

**contour mutual embedding of  $n$  cardinality:** (CMEMB <sub>$n$</sub> ) An adjustment of CEMB to measure similarity between contour segments of equal or unequal cardinality where  $n$  equals a certain cardinality and CMEMB <sub>$n$</sub>  measures how many times a contour subsegment of  $n$  cardinality occurs in a contour segment.

**contour pitch:** (cp) An integer representing the height of a pitch relative to those around it.

**contour prime:** A contour segment's fundamental structure.

**contour reduction algorithm:** (CRA) A process for recursively pruning maxima and minima from a contour segment until its contour prime is revealed.

**contour segment:** (cseg) An ordered set of contour pitches.

**contour shade:** The result of a change in the predominant diachronic contour prime during the process of Diachronic Transformational Contour Analysis.

**contour similarity:** (CSIM) A measure of the similarity of two contour segments of equal cardinality determined by tallying the number of equivalent entries in the upper right-hand triangles of the COM-matrices of the contour segments being compared. This number is then divided by the total possible number of entries and returns a value between 0 and 1. 0 indicates oppositeness. 1 indicates equivalence.

**contour similarity of Adams's type:** (CSIM-AT) The measure of contour similarity between any pair of Adams's contour types.

**contour smooth network:** A uniformly balanced contour web, similar to a transformational network, which connects all contours that differ by only one contour pitch.

**contour space:** (c-space) The ranking of contour pitches from low to high, ignoring the exact intervals between contour pitches; the lowest contour pitch in a specified region is 0 and the highest is  $n - 1$ .

**contour subsegment:** (csubseg) Any ordered sub-grouping of contour pitches of a given contour segment.

**contour subset:** The division of a contour segment into smaller subsegments, allowing for hierarchical and embedding relationships between contours.

**criterion:** A rationale for the cognitive grouping of musical events or segmentation.

**Darmstadt School:** A designation associated primarily with serial music written in the 1950s by Luigi Nono, Bruno Maderna, Karlheinz Stockhausen and Pierre Boulez and promoted by them in the 1950s at the Darmstadt summer courses.

**debug:** The methodical process of finding and reducing the number of defects in a computer program.

**dependency:** When computer source code depends on other source code for its full meaning and implementation.

**deploy:** The process of installing and running a computer program.

**development kit:** A set of software routines and utilities used to help programmers write an application. For graphical user interfaces, it provides the tools and libraries for creating menus, dialog boxes, fonts and icons. It provides the means to link the application to libraries of software routines and to the operating environment.

**diachronic:** Of, relating to, or dealing with phenomena as they occur or change over a period of time.

**Diachronic Transformational Contour Analysis:** (DTA) A contour analysis that repetitively determines the contour prime for a piece as each pitch is performed. A contour analysis that deals with contour prime as it changes over a period of time.

**dimension:** See parameter.

**discontinuity to event ratio:** (DTER) The ratio of discontinuities within a parameter to the total number of events; higher ratios represent an approximation of the likelihood that a particular parameter will yield a productive segmentation.

**disjunction:** A single measure of change/difference between two musical units.

**disjunctive orientation:** The orientation concerned with difference and the magnitude of change. Disjunctions separate musical events from one another and lift segments from their surroundings.

**distance interval:** The sum of the absolute value of the distance between *elements* of the same parameter.

**domain:** See parameter.

**double:** A floating type with precision up to 15 digits.

***element:*** A temporal gestalt unit that is not further divisible.

**embed:** The inclusion of a set class of smaller cardinality within a set class of larger cardinality.

**encode:** The act of assigning source code to represent data.

**end user:** (user) The target individual for a computer program.

**esthetic level:** An analyst's perception of a musical work.

**executable:** A file that causes a computer to perform an indicated task(s) according to encoded instructions.

**execute:** To run a computer program, which causes the computer to carry out its instructions.

**extensible markup language:** (xml) A language for annotating a document that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

**floating type:** (float) A variable used to define numbers with fractional parts.

**fuzzy logic:** Logic based on approximation rather than a binary true/false relationship.

**generative-transformational grammar:** The theory developed by Noam Chomsky which focuses on unconscious knowledge which is the innate knowledge we already have that allows us to learn language.

**genosegment:** A potentially perceptible grouping of pitches (or other sound events) supported by exactly one sonic criterion or contextual criterion.

**gestalt psychology:** The study of perception of an individual's response to how things are put together which rejects analysis into discrete events of stimulus, percept, and response.

**GFL-reflective:** When the contour adjacency series of a contour segment has a deceleration or a lengthening duration at its conclusion.

**GlassFish:** The Java library that contains all of the support for Java web applications, such as servlets and JavaServer Pages.

**Google Web Toolkit:** (gwt) An open source software development kit from Google for creating browser-based applications.

**graphical user interface:** (GUI) A computer program that allows a person to work easily with a computer by using a mouse to point to small pictures and other elements on the screen.

**group final lengthening:** (GFL) A deceleration occurring at the end of a contour segment.

**grouping structure:** A listener's segmentation of a musical piece.

**HashMap:** A type of map with key-value pairings that can be easily searched. Duplicate entries are not allowed.

**HashMultiMap:** A HashMap that can have multiple values for any key and does not allow duplicate key-value pairs.

**hyperlink:** A highlighted word or picture in a document or World Wide Web page that you can click on with a computer mouse to go to another place in the same document, a different document, or a World Wide Web page.

**HyperText Markup Language:** (HTML) The standard document format for World Wide Web pages.

**imbrication:** The process of segmenting a composite segment into smaller, overlapping contour subsegments.

**inheritance:** The ability of one class of objects to receive properties from a higher class.

**inlay:** The placement of shorter set class statements within longer statements of another set class.

**instantiation:** A single criterion that delineates a segment.

**integer:** Any number that is not a fraction or decimal; Any whole number or its negative.

**integer variable:** (int) A structure that holds data, in this case that data must be an integer.

**integrated development environment:** (IDE) A software application that provides comprehensive facilities to computer programmers for software development, normally consisting of a source code editor, build automation tools, and a debugger.

**Internet:** (Net) A global internet comprising nearly a billion World Wide Web, e-mail, and related servers in more than 100 countries.

**internet:** A large network made up of smaller networks.

**intersection:** Given equivalent pitch-class sets A and B, a third set C is determined by the elements that are both in A and in B.

**interval:** The distance between two pitches.

**interval class:** (ic) The interval formed by determining the distance, in semitones, between two pitch-class integers. Interval classes range from 0 to 6.

**interval class vector:** (icv) The description of the total interval content of any given pitch-class set displayed as an ordered array.

**interval succession:** (INT<sup>+</sup>) see contour adjacency series.

**inversion:** (I) Taking the inverse of each element in a segment. The inverse of an element x is that element x' that when added to x gives 0:  $x + x' = 0$ .

**Java:** A general-purpose, concurrent, class-based, object-oriented computer programming language that is designed to have as few implementation dependencies as possible.

**Java archive:** (JAR) The single archive file format into which multiple Java files are bundled.



**Java development kit:** (jdk) A Java software development environment. It includes the Java virtual machine, compiler, debugger and other tools for developing Java applets and applications..

**Java package:** (package) A mechanism for organizing Java classes.

**Java virtual machine:** (JVM) Software that converts a computer program in Java bytecode (intermediate language) into machine language and executes it.

**javascript:** A programming language that is embedded in most World Wide Web pages which enables interactive functions to be added to World Wide Web pages which otherwise lack interactive capabilities.

**JavaServer Page:** (jsp) An extension to the Java servlet technology that allows HTML to be combined with Java on the same World Wide Web page. The Java provides the processing and the HTML provides the layout on the World Wide Web page.

**Jikken Kōbō:** An interdisciplinary experimental workshop similar to the Darmstadt School, existing in Japan from 1951 to approximately 1958, which sought to combine Japanese artistic tradition with Western harmony.

**library:** In computer programming, a collection of implementations of behavior, written in terms of a computer programming language, that is a well-defined interface by which the behavior is invoked.

**Lilypond:** An open source, text-input, music engraving computer program.

**LinkedHashMap:** A HashMap that stores entries in the order in which they were added.

**list:** An arranged set of data, often in row and column format.

**loop:** A repetition within a computer program.

**ma:** In Japanese are, this concept is interpreted as emptiness or space. In the works of Kazuo Fukushima, it refers to the expressive space between musical phrases.

**macro language:** (macro) A set of instructions that causes a computer to perform a series of tasks.

**manifest:** A list of the contents of a computer program's distribution file.

**map:** An object that assigns a relationship between two items.

**maximum:** A local high point of a contour segment.

**meta-information:** Also known as meta-data. Data that provides information about one or more aspects of data, such as means of creation, time and date of creation, and programming author information.

**method:** A procedure, associated with a class, which implements a routine as coded by the programmer.

**Microsoft Excel:** A spreadsheet application that features calculation, graphing tools, tables, and macro language.

**microtone:** Any musical interval or difference of pitch distinctly smaller than a semitone.

**minimum:** A local low point of a contour segment.

**mode of limited transposition:** (mode) Term used by Olivier Messiaen in his *Technique de Mon Language Musical* (Paris, 1944; Eng. trans., 1956, chap.16) for a scale which can be transposed by a semitone fewer than 11 times before the original set of notes reappears.

**modular arithmetic:** Arithmetic that deals with whole numbers where the numbers are replaced by their remainders after division by a fixed number.

**multiphonic:** Sounds generated by a normally monophonic instrument in which two or more pitches can be heard simultaneously.

**multiple linear regression:** The attempt to model the relationship between two or more variables and a response variable by fitting a linear equation to observed data.

**Musical Instrument Digital Interface:** (MIDI) An established norm that de-

scribes a system of rules for message exchange within or between computers and allows a wide variety of electronic musical instruments and computers to communicate with each other.

**MusicXML:** A digital sheet music interchange and distribution format with the goal of creating a universal format for common Western music notation.

**nesting:** The systematic joining of elements of a segment.

**network:** A system that transmits data between users. It includes the network operating system in the client and server machines, the cables connecting them, and all supporting hardware in between. In wireless systems, antennas and towers are also part of the network.

**neutral level:** The exhaustive description and inventory of all possibly conceivable configurations in a musical score.

**Noh:** A Japanese form of drama which originated in the 14th century as an exhibition of talent combining elements of dance, drama, music, and poetry into one highly aesthetic art form, comparable to the Western musical.

**Noh-kan:** A Japanese transverse flute associated with *Noh* theater. It is made of 100-year-old smoked bamboo and has seven finger holes and a mouth hole.

**non-parametric variable:** A term used by Nicholas Ruwet to refer to a musical aspect which is not constant throughout a piece, such as pitch, duration, intensity, or scale.

**normal form:** An ordering of the pitches within a set class that spans the smallest possible interval.

**normalized weighting value:** (NWW) The result of the process of normalizing all of the parameters' relative weights are so that they sum to 1.0.

**object:** A self-contained module of data and its associated processing.

**object-oriented:** A computer programming paradigm that represents concepts as objects that have data fields and associated procedures known as methods.

**open source:** Software that is distributed with its source code so that end user organizations and vendors can modify it for their own purposes.

**operating system:** (OS) A computer's master control program.

**orientation:** A perceptual or cognitive strategy.

**package:** A software that has been built from source code and typically contains compiled source code with additional meta-information such as a package description, package version, or dependencies.

**parameter:** (domain, dimension) Any of a set of compositional or auditory properties whose values determine the characteristics or behavior of a piece of music.

**phenomenology:** The way in which one perceives and interprets events and one's relationship to them in contrast both to one's objective responses to stimuli and to any inferred unconscious motivation for one's behavior.

**phenosegment:** A readily perceptible musical segment supported by at least one sonic criterion or contextual criterion (and perhaps also structural criteria).

**pitch-class:** (pc) The collection of all pitches that are enharmonically equivalent and/or related by octave transposition and are labeled from 0 through 11.

**pitch-class set:** (pcset) An unordered collection of pitch-classes without replication.

**pitch-class set complex:** (set complex) A collection of pitch-classes made up of related smaller pitch-class sets.

**pitch-class set theory:** A method of music analysis in which musical sets of unordered pitch-classes are compared to other sets and in which relationships are determined by transposition, inversion, retrograde, , interval content, symmetry, embedment, similarity, and set complexes.

**plug-in:** Software that is installed into an existing application in order to enhance its capability.

**poietic level:** The compositional procedures and intentions of a composer.

**portable document format:** (PDF) The de facto standard for electronic document

publishing from Adobe, the leading multimedia software company.

**post-tonal:** A type of music composition in which there is no hierarchy of functional harmony, no distinction between consonance and dissonance, for which a traditional tonal analysis or pitch-class set analysis does not yield satisfying analytical results, or for which a pitch-class set analysis alone is not sufficient to illuminate the structure of the music.

**preference rules:** (PRs) Grouping rules which specify groups that correspond to an experienced listener's hearing of a particular musical piece.

**primary segment:** A musical unit demarcated by conventional means, such as by rests.

**prime form:** The normal form of a pitch-class set transposed to begin on pitch-class 0.

**programming language:** See computer programming language.

**prune:** The recursive elimination of non-maxima and non-minima from a contour segment.

**public:** An access modifier that allows all parts of the computer program access to the source code in question.

**Python:** A popular, object-oriented scripting language used for writing system utilities and Internet scripts.

**quarter-tone:** An interval half the size of a semitone.

**realization:** A type of coincidence in which one criterion is a structural criterion and the other is a contextual criterion, generating a stronger argument for segmentation.

**resource consumption:** The extent to which a computer program uses computer memory and central processing unit time.

**retrogression:** (R) An order operator used to reverse the order of elements in a segment.

**robust:** Source code that is capable of performing without failure under a wide range of conditions.

**script:** A computer program written in a general-purpose computer programming language.

**section (temporal gestalt):** The temporal gestalt unit at the level between *segment* and piece.

**segment (temporal gestalt):** The temporal gestalt unit at the level between *sequence* and *section*.

**segment:** A grouping of notes (or other sound-events) that constitutes a significant musical object in analytic discourse.

**segmentation:** The procedure for determining how a musical work is divided into structurally significant musical units.

**sequence:** A succession of two or more *clangs*.

**server:** A computer system in a network that is shared by multiple end users.

**servlet:** A Java application that runs in a World Wide Web server or application server and provides server-side processing.

**set class:** (SC) The collection of pitch-class sets related by any combination of transposition and inversion.

**set theory:** A branch of mathematics or of symbolic logic that deals with the nature and relations of sets.

**shakuhachi:** An end-blown flute (open or stopped) from Japan with a V- or U-shaped notch cut or burnt into its upper rim to facilitate tone production.

**software:** Instructions for the computer.

**software architecture:** The design of application software that allows for interacting with other computer programs and for future flexibility and expandability.

**sonic criterion:** A basic type of criterion that responds to disjunctions in the attribute-values of individual sounds and silences within a single psychoa-

coustic musical dimension.

**sonic domain:** The psychoacoustic aspect of music.

**source code:** (code) Any collection of computer instructions (possibly with comments) written using some human-readable computer programming language, usually as text.

**source code editor:** A computer program designed specifically for editing source code text of computer programs by programmers.

**spreadsheet:** A software that simulates a paper spreadsheet, in which columns of numbers are summed for budgets and plans.

**static:** A modifier that states that there is only one such object of this name in a class.

**structural criterion:** A basic type of criterion that assumes a theoretic orientation and indicates an interpretation supported by a specific orienting theory.

**structural domain:** The realm of interpretation shaped by active reference to a theory of musical structure or syntax chosen by the analyst.

**subroutine:** A group of instructions that perform a specific task.

**synchronic:** Concerned with events existing in a limited time period and ignoring historical antecedents; concurrent.

**temporal gestalt unit:** (TG) The grouping of musical objects based on the principles of similarity and proximity.

**theory:** A coherent system of rules or principles, commonly regarded as correct, that can be used as principles for explanation and prediction for a class of phenomena.

**theory orientation:** The orientation that identifies or interprets musical events in terms of a specific theory of musical structure, its theoretical framework, and theoretic entities.

**Total Segmentation Value:** (TSV) The sum of the normalized weighting values.

**transformed interval class:** (tic) The average of the sum of the interval classes between all pairs of adjacent pitches in a vertical sonority.

**transposition:** (T) The process of adding  $n$  to each element in a segment.

**trichord:** A pitch-class set of cardinality 3.

**twelve-tone row:** (row) A segment of twelve pitch-classes without replication.

**Universal Tree Diagram:** A diagram of transformational contour chains showing all possible contours as a melody proceeds from left to right through time.

**version:** (ver) The form of a computer program that is different in some way from previous forms.

**Visual Basic for Applications:** (VBA) The name of the programming language for Microsoft Excel.

**Web browser:** The application computer program that serves as the primary method for accessing the World Wide Web.

**well-formedness rules:** (WFRs) Grouping rules which specify structural descriptions.

**word processor:** A software application that is used to create and edit text documents.

**World Wide Web:** (web) An Internet-based system, accessed through a graphical user interface and containing documents with hyperlinks, that enables an individual or a company to communicate or advertise itself to the world.

**World Wide Web-based application:** (web application) Any application that uses a Web browser as a client.

**Z-chain:** A repetitive inspection of a melody for a chain with a specific orientation.



# Appendices

# Appendix A

## Source Code: Robert Morris CRA

```
package net.cooleysekula.CAT;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashSet;

public class Methods extends ContourApp {

    public static ArrayList getMorris(ArrayList list) {
        ArrayList<Double> data = new ArrayList(list);

        ArrayList<Double> maxlist = new ArrayList(data);
        ArrayList<Double> minlist = new ArrayList(data);

        int i = 0;
        int j = 0;
        int k = 0;

        //Find max points of original data set. Call it maxlist
        //Find min points of original data set. Call it minlist
```

```

int depth;
int foundIt = 0;
int cont;
int cont2;

////////////////////////////////////
// CONTOUR REDUCTION ALGORITHM //
// Given a contour C and a variable N //
// STEP 0: Set N to 0 //
////////////////////////////////////

depth = 0;

////////////////////////////////////
// CONTOUR REDUCTION ALGORITHM //
// STEP 1: Flag all maxima in C //
////////////////////////////////////

for (i = 0; i < data.size() - 2; i++) {
    if ((data.get(i + 1) >= data.get(i) &&
        data.get(i + 1) >= data.get(i + 2)))
    {
    }
    else {
        maxlist.remove((i + 1) - j);
        j++;
    }
}

////////////////////////////////////
// CONTOUR REDUCTION ALGORITHM //
// STEP 2: Flag all minima in C //
////////////////////////////////////

for (i = 0; i < data.size() - 2; i++) {
    if (data.get(i + 1) <= data.get(i) &&
        data.get(i + 1) <= data.get(i + 2))

//Iterate through original data list ...
// ... and compare each element to elements on either side

```

```

{
}
else {
    minList.remove((i + 1) - k);
    k++;
}
}

////////////////////////////////////
//      CONTOUR REDUCTION ALGORITHM      //
// STEP 4: Delete all non-flagged pitches in C //
////////////////////////////////////

ArrayList<Double> depth1 = new ArrayList(data);

j = 0;

for (i = 0; i < data.size() - 2; i++) {
    if ((data.get(i + 1) >= data.get(i)
        && data.get(i + 1) >= data.get(i + 2)) |
        (data.get(i + 1) <= data.get(i)
        && data.get(i + 1) <= data.get(i + 2)))
    {
    }
    else {
        depth1.remove((i + 1) - j);
        j++;
    }
}

////////////////////////////////////
//      CONTOUR REDUCTION ALGORITHM      //
// STEP 3: If all pitches of C are flagged, go to Step 9 //
////////////////////////////////////

```

```

//If central element is smaller, do nothing

//Else if central element is greater, remove it

//Remove non maxima\minima from original data set to determine the elements of depth1
//Create new ArrayList 'depth1' which contains all elements of 'data'

//Iterate through original data set

//Compare each element to elements on either side

//If central element is a maxima or minima, do nothing

//If central element is not a maxima or minima, remove it

```

```

for ( i = 0; i < depth1.size(); i++) {
    if (depth1.size() == data.size()) {
        foundt = 1;
    }

    if (foundt == 1) {
        break;
    } else {
        continue;
    }
}

////////////////////////////////////
// CONTOUR REDUCTION ALGORITHM //
// STEP 5: N is incremented by 1 //
////////////////////////////////////

depth = 1;

do
{
    cont = 0;
    cont2 = 0;

    //////////////////////////////////////
    // CONTOUR REDUCTION ALGORITHM //
    // STEP 6: Flag all maxima in max-list //
    // For any string of equal and adjacent maxima in max-list, either //
    // 1) flag only one of them; or //
    // 2) if one pitch in the string is the first or last pitch of C, flag only it; or //
    // 3) if both the first and last pitch of C is in the string, //
    // flag (only) both the first and last pitch of C. //
    // //
    // CONTOUR REDUCTION ALGORITHM //
    // STEP 7: Flag all minima in max-list //
    // For any string of equal and adjacent minima in min-list, either //

```

```

// 1) flag only one of them; or //
// 2) if one pitch in the string is the first or last pitch of C, flag only it; or //
// 3) if both the first and last pitch of C is in the string, //
// flag (only) both the first and last pitch of C. //
////////////////////////////////////

//Find and remove any repeated data points in maxlist1
for (i = 0; i < maxlist.size() - 2; i++) {
    if (maxlist.get(i + 1).equals(maxlist.get(i)) ||
        maxlist.get(i + 1).equals(maxlist.get(i + 2))) {
        maxlist.remove(i + 1);
    }
}

//Find and remove any repeated data points in minlist1
for (i = 0; i < minlist.size() - 2; i++) {
    if (minlist.get(i + 1).equals(minlist.get(i)) ||
        minlist.get(i + 1).equals(minlist.get(i + 2))) {
        minlist.remove(i + 1);
    }
}

////////////////////////////////////
// CONTOUR REDUCTION ALGORITHM //
// STEP 4: Delete all non-flagged pitches in C //
////////////////////////////////////

ArrayList<Double> maxlist1 = new ArrayList(maxlist);
j = 0;

//Iterate through original data list
for (i = 0; i < maxlist.size() - 2; i++) {
    if (maxlist.get(i + 1) >= maxlist.get(i) &&
        maxlist.get(i + 1) >= maxlist.get(i + 2)) //Compare each element to elements on either side
    {
    }
    //If central element is greater, do nothing
    else {

```

```

        maxlist1.remove((i + 1) - j);
        j++;
        cont = 1;
    }
}

//Else if central element is smaller, remove it

//Drop elements that do not qualify as minima of minlist1. Call it minlist2.
ArrayList<Double> minlist1 = new ArrayList(minlist);
j = 0;
for (i = 0; i < minlist.size() - 2; i++) {
    //Iterate through original data list
    if (minlist.get(i + 1) <= minlist.get(i) &&
        minlist.get(i + 1) <= minlist.get(i + 2)) //Compare each element to elements on either side
    {
        //If central element is greater, do nothing
    }
    else {
        minlist1.remove((i + 1) - j);
        j++;
        cont2 = 1;
    }
}

//Find and remove any repeated data points in maxlist1
for (i = 0; i < maxlist1.size() - 2; i++) {
    if (maxlist1.get(i + 1).equals(maxlist1.get(i)) ||
        maxlist1.get(i + 1).equals(maxlist1.get(i + 2))) {
        maxlist1.remove(i + 1);
    }
}

//Find and remove any repeated data points in minlist1
for (i = 0; i < minlist1.size() - 2; i++) {
    if (minlist1.get(i + 1).equals(minlist1.get(i)) ||
        minlist1.get(i + 1).equals(minlist1.get(i + 2))) {
        minlist1.remove(i + 1);
    }
}

//Combine the newly reduce maxlist and minlist and compare to depth 1

```

```

//Delete pitches from depth1 that do not occur in the new maxlist or minlist

////////////////////////////////////
//      CONTOUR REDUCTION ALGORITHM      //
// STEP 3: If all pitches of C are flagged, go to Step 9 //
////////////////////////////////////
if (cont != 0 | cont2 != 0) {

////////////////////////////////////
//      CONTOUR REDUCTION ALGORITHM      //
// STEP 5: N is incremented by 1 //
////////////////////////////////////
depth++;
maxlist = maxlist1;
minlist = minlist1;
continue;
} else {
maxlist = maxlist1;
minlist = minlist1;
break;
}
} while ((cont != 0 | (cont2 != 0)); //End of while loop

////////////////////////////////////
//      CONTOUR REDUCTION ALGORITHM      //
// STEP 9: End. N is the "depth" of the original contour C //
////////////////////////////////////

maxlist.removeAll(minlist);
minlist.addAll(maxlist);
depth1.retainAll(minlist);

//Find union of maxlist and minlist

//Remove immediate duplicates from new depth1
//Iterate through maxlist1
for (i = 0; i < depth1.size() - 2; i++) {
if (depth1.get(i + 1).equals(depth1.get(i)) ||
depth1.get(i + 1).equals(depth1.get(i + 2))) {
depth1.remove(i + 1);
}
}

```



```

    }

    ArrayList<Double> finalist = new ArrayList();
    //Create a new list to place elements existing in both minlist and depth1 into

    for (i = 0; i < depth1.size(); i++) {
        if (minlist.contains(depth1.get(i))) {
            finalist.add(depth1.get(i));
        }
    }

    final ArrayList hs = new ArrayList(new LinkedHashSet(finalist));

    //Make sure first and last elements of original data set are still first and last

    for (i = 0; i < hs.size() - 1; i++) {
        if (hs.get(i).equals(depth1.get(depth1.size() - 1))) {
            Collections.swap(hs, i, hs.size() - 1);
        }
        if (hs.get(i).equals(depth1.get(0))) {
            Collections.swap(hs, i, 0);
        }
    }
    return hs;
} //end getMorris method
}

```

# Appendix B

## Source Code: main.js

```
function tabs(a, g, j) {
    document.body.className = "js-on";
    var g = a.getElementsByTagName(g);
    d = [];
    c;
    this.active;
    this.total = g.length;
    this.container = a;
    e = a.insertBefore(document.createElement("nav"), g[0]), change = function (f) {
        if (typeof this.active !== "undefined") {
            d[this.active].className = g[this.active].className = "n"
        }
        d[f].className = g[f].className = "active";
        this.active = f
    }, clickEvent = function (h, f) {
        h.onclick = function () {
            change(f);
        }
    }
```

```

        return false
    }
};
for (var b = 0; b < g.length; b++) {
    d[b] = e.appendChild(document.createElement("a"));
    d[b].href = "#";
    c = [g[b].getAttribute("data-title"), g[b].getElementsByTagName(j)[0]];
    d[b].innerHTML = c[0] !== null ? c[0] : c[1] ? c[1][ "innerText" || "textContent" ] : b + 1;
    new clickEvent(d[b], b)
}
change(0)
}
tabs.prototype.change = function (b) {
    change(b - 1)
};
tabs.prototype.next = function (b) {
    active === this.total - 1 ? change(0) : change(active + 1)
};
tabs.prototype.prev = function (b) {
    active === 0 ? change(this.total - 1) : change(active - 1)
};
tabs.prototype.response = function (d, c) {
    nav = document.createElement("nav");
    nav.id = "mobiles";
    nav.innerHTML = '<a href="#'+onclick+'"+d+'prev(); return false'>' + c.prev + '</a><a href="#'+onclick+'"+d+'next(); return false'>' + c.next + '</a>';
    this.container.insertBefore(nav, this.container.firstChild);
    return this
};
};

```

# Appendix C

## Source Code: style.css

```
/* Structure */
html{
  background: #eee;
}
body {
  padding: 0px;
  background: #fff;
  color : #333;
  margin: 0 auto;
  max-width: 900px;
  font: 1em/1.5 "Helvetica Neue", Helvetica, Arial, sans-serif;
}

a {
  color : #105672;
}
```

```

header {
    background: #171e2e;
    padding: .5em 3em;
    color : #fff;
    line-height: 1;
}

header h1{
    margin-bottom: 0;
}

header h1 span{
    display: inline;
    color : rgba(255,255,255,.4);
}

header span{
    display: block;
    color : rgba(255,255,255,.2);
    font-weight: 300;
    margin-bottom: 1.6em
}

header nav{
    float : right;
    text-align: right
}

header nav div {
    font-size: .8em;
}

header nav div a {
    font-weight: 300;
    padding: .3em .5em
}

header nav a {
    color : #fff;
    display: inline-block;

```

```

padding: .3em .8em
}

header nav a:hover, header nav a:focus{
color : rgba(255,255,255,.6)
}

[role=main]{
padding:1.5em 3em;
}
article {
padding: 1em 0;
}
footer{
background: #333;
color : #fff;
padding: .1em 3em;
}

/* Typography */

p{
font: 1em/1.5 Palatino, "Palatino Linotype", Georgia, Times, "Times New Roman", serif;
}

img{
max-width: 100%;
height: auto;
}

blockquote{
float : left ;
margin: 1em 3em;
}

```

```

blockquote p{
    font-size: 1.4em;
    line-height: 1.2;
    font-weight: 700;
    font-style: italic ;
}
a{
    font: 700 1em/1.5 "Helvetica Neue", Helvetica, Arial, sans-serif;
    text-decoration: none
}
a:hover, a:focus{
    color : #000;
}
a:active{
    position: relative ;
    top:1px;
}

/* Tabs */

.js-on #tabs article
{
    display:none
}

#tabs, #tabs nav a:active{
    background: #ddd;
    color : #111;
}

#tabs nav
{
    position: relative ;
    overflow: hidden;
    display: table;
    background: #bbb;
}

```

```

#tabs nav a
{
    width:150px;
    display:table-cell;
    padding:1em;
    text-align:center;
    color : #333;
}

#tabs nav a:hover,#tabs nav a:focus
{
    background:#eee
}

#tabs article
{
    padding:2em;
}

.js-on #tabs article.active
{
    display:block;
}

#tabs #mobiles{
    display:none;
    border-radius: 0;
}

#tabs #mobiles a, #tabs #mobiles a:first-child, #tabs #mobiles a:last-child{
    width:300px;
    border-radius: 0;
}

/* Media queries */

```



```

@media screen and (min-width:900px)
{
    body{font-size: 1.1em;}
}

@media screen and (max-width:600px)
{
    #tabs nav{
        display: none;
        position: relative;
    }
    #tabs #mobiles{
        display: block;
    }
    #tabs article {
        display: block;
    }
}

@media screen and (max-width:480px)
{
    blockquote{
        float : none;
    }

    header nav a{
        padding:.7em .8em
    }
    header nav{
        float : none;
        margin: -.5em -3em 0;
        background: #000;
        overflow: hidden;
        text-align: left
    }
    header nav a{
        border-right: 1px solid #222
    }
}

```

```
[role=main]{  
  padding:1.5em 2em;  
}  
header nav div{  
  display: none;  
}  
}
```

# Appendix D

## Source Code: index.jsp

214

```
<%--
    Document    : index
    Created on : Jan 9, 2014, 1:23:37 PM
    Author     : katesekula
    --%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="<c:url value ="style.css"/>" type="text/css" >
    <meta name="gwt:module" content="net.cooleysekula.Main=net.cooleysekula.Main" >
    <meta name="author" content="Kate Sekula" >
    <title>Tools for Music Analysis</title>
    <script type="text/javascript" src="net.cooleysekula.Main/net.cooleysekula.Main.nocache.js" ></script>
</head>
```

```

<body>
<header>
<nav> <a href="kate.cooleysekua.net">Home</a> <a href="#">A bout</a> <a href="#">Contact</a>
<div> <a href="https://www.facebook.com/katesekula">Facebook</a> <a href="http://www.youtube.com/katesekula">YouTube</a> <a href="http://www.music.uconn.edu/">
UConn Music</a> <a href="usao.edu">USAO</a></div>
</nav>
<h1>Tools for <span>Music Analysis</span></h1>
<span>a subtitle goes here</span> </header>
<section id="tabs">
<article>
<h2>Contour</h2>
<p>Input pitches as integers or decimals (i.e. 7 7 2 7 2 7 11 14)</p>
<form method="POST" action="Results" id="useform">
<p>Enter Pitches: <input type="text" name="pitches" size="63">
<input type="submit" value="Submit" name="B1"></p>
</form>
<textarea style="margin: 2px; width: 813px; height:160px;" name="result" form="useform">${todo}</textarea>

<form method="POST" action="GetMid" enctype="multipart/form-data" id="useform2">
<input type="file" value="Upload Midi" name="uploadmidi" onclick="ClearFields()">
<input type="submit" value="Upload File">
<textarea style="margin: 2px; width: 813px; height:40px;" id="derp" name="midresult" form="useform2">${todo2}</textarea><br>
</form>
Copy and paste upload results into box labeled "Enter pitches."
<script>
function ClearFields() {
    document.getElementById("derp").value = "";
}
</script>

<p> The application will return the contour reduction according to Rob Schultz Contour Reduction Algorithm, depth, prime and normal forms, set class info and the contour prime</p>
</article>
<article>
<h2>More Tools</h2>
<p>More tools to come soon</p>

```

```

</article>
<article>
  <h2>More Tools</h2>
  <p>More tools to come soon</p>
</article>
<article>
  <h2>More Tools</h2>
  <p>More tools to come soon</p>
</article>
</section>
<footer>
  <p>&copy; Copyright 2014 – CooleySekula | Website Template By <a target="_blank" href="http://mlb.li/">Matthias Le Brun</a></p>
</footer>
<script src="main.js"></script>
<script>
  var myTabs = new tabs(document.getElementById("tabs"), "article", "h2").responsive("myTabs", {
    prev: "Previous",
    next: "Next"
  });
</script>
</body>
</html>

```

## Appendix E

### Source Code: methods.class

```
package net.cooleysekula.CAT;

import com.google.common.collect.HashMultimap;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

/**
 *
 * @author katesekula
 */
public class Methods {

    public static ArrayList rotateLeft(ArrayList list) { //create method rotateLeft()
```

```

ArrayList<Double> data = new ArrayList(list);

//create ArrayList 'data' with contents of 'list'
data.add(data.get(0));
data.remove(0);

return data;
}

public static ArrayList getMod12(ArrayList list) {
    ArrayList<Double> data = new ArrayList(list);
    int i;

    for (i = 0; i < data.size(); i++) {
        if (data.get(i) >= 0 && data.get(i) < 12) {
        } else {
            data.set(i, data.get(i) % 12);
        }
    }
    return data;
}

public static ArrayList CheckIntegrity(ArrayList list) {
    Set se = new HashSet(list);
    list.clear();
    list = new ArrayList(se);
    return list;
}

public static ArrayList toZero(ArrayList list) {
    ArrayList<Double> data = new ArrayList(list);
    ArrayList<Double> moddata = new ArrayList();
    for (int i = 0; i < data.size(); i++) {
        moddata.add(data.get(i) - data.get(0));
    }

    for (int i = 0; i < moddata.size(); i++) {
        if (moddata.get(i) < 0) {
            //transpose data so it begins on zero, called toZero()
            //create ArrayList of contents of list
            //create new ArrayList to put results into
            for all elements in 'data' ...
            //... add all of them to new list 'moddata'
            //end forloop
            //above process can result in negatives. Let's fix that.
            //look through each element in 'moddata'
            //if an element in 'moddata' is less than zero ...

```

```

        moddata.set(i, moddata.get(i) + 12);
    }
    return moddata;
}

public static Double sum(ArrayList<Double> list) {
    Double sum = 0.0;
    for (Double i : list) {
        sum = sum + i;
    }
    return sum;
}

public static ArrayList getNormal(ArrayList list) { //create method getNormal(form
    ArrayList<Double> data = new ArrayList(list); //declare all lists , maps and variables
    HashMap<Double, ArrayList> map = HashMap.create();
    HashMap<Double, ArrayList> map2 = HashMap.create();
    HashMap<Double, ArrayList> map3 = HashMap.create();
    HashMap<Double, ArrayList> map4 = HashMap.create();
    HashMap<Double, ArrayList> map5 = HashMap.create();
    ArrayList<Double> datarotate = new ArrayList();
    ArrayList<Double> mapToArray = new ArrayList();
    double count = 0.0;
    double count2 = 0.0;
    double d;
    double c;
    int b;
    int j;
    boolean done = false;
    double minValue;

    Collections.sort(data);

    //STEP 1: Sort 'data' into numerical order
    //STEP 2: List all rotations
    //If 'data' had only one element...
    // ... add that element to 'mapToArray'...
    // ... else ...
    if (data.size() == 1) {
        mapToArray.add(0.0);
    } else {

```



```

while (count < data.size()) {
    // ... create while statement to act on counter
    //While counter is less than size of 'data' ...
    c = (data.get(data.size() - 1) - data.get(0)); //get distance between first and last pitch
    map.put(c, data);
    //place info into map (distance, data)
    datarotate = Methods.rotateLeft(data); //call method rotateLeft to rotate 'data'
    d = datarotate.get(datarotate.size() - 1) + 12.0; //add 12 to the number at far right
    b = datarotate.indexOf(datarotate.get(datarotate.size() - 1)); //get index of number at far right
    datarotate.set(b, d); //replace element at far right with new value
    data = datarotate; //replace original 'data' with new 'datarotate'
    count++; //increase counter by 1
} //end STEP 2

minValue = (Collections.min(map.keys())); //***STEP 3: Determine minimum distance
//set variable 'minValue' to determine min value in a HashMap

for (ArrayList key : map.get(minValue)) {
    map2.put(count2, key); //Find the minimum value in HashMap 'map'
    count2++; //Place this value into a new map, map2
} //this counter ensures that all values have an individual key

if (map2.size() == 1) {
    for (ArrayList value : map2.values()) {
        mapToArray.addAll(value); //if 'map2' has only one value...
        // ... then get the value from 'map2'...
        // ... and add it to ArrayList 'mapToArray'...
    }
    done = true; // ... and set variable 'done' to true. End process.
}

do {
    for (j = 1; j < count; j++) {
        for (ArrayList<Double> diff : map2.values()) { //iterate through all the distance values
            d = (diff.get(j) - diff.get(0)); //find min distance between first and second(third, fourth, etc.) numbers.
            map3.put(d, diff); //place these new distance values into ArrayList 'map3'
        }
    }

    if (map3.size() == 1) {
        for (ArrayList value : map3.values()) { //if 'map3' has only one value...
            // ... then get the value from 'map2'...
            mapToArray.addAll(value); // ... and add it to ArrayList 'mapToArray'...
        }
    }
}

```

```

        done = true;
    }
}
count2 = 0.0;
minValue = (Collections.min(map3.keys())); //set variable 'minValue' to determine min value in a HashMap

for (ArrayList value : map3.get(minValue)) { //Find the minimum value in HashMap 'map3'
    map4.put(count2, value);
    count2++;
}

//this counter ensures that all values have an individual key

if (map4.size() == 1) {
    //if 'map2' has only one value...
    for (ArrayList value : map4.values()) { //... then get the value from 'map2'...
        mapToArray.addAll(value);
        done = true;
    }
} else if (data.size() < 3) {
    //else, if 'data' has only 2 elements...
    Double doubleValue = 0.0;
    for (Map.Entry<Double, ArrayList> entry : map4.entrySet()) { //...look through all 'map4' elements...
        if (entry.getKey().equals(doubleValue)) { //...and if any key is 'map4' is zero
            mapToArray = entry.getValue(); //...replace the entry at that key with the value ...
            done = true;
        }
    }
    //End first half if--statement in Step 2
} else {
}
}
if (j == count) {
    //break any ties
}

for (ArrayList<Double> value : map4.values()) { //find the sum of all values in 'map4'
    map5.put(Methods.sum(value), value); //Place value results into a new map, 'map5'. The value with the lowest sum wins
}

minValue = (Collections.min(map5.keys())); //set variable 'minValue' to determine min value in a HashMap
for (ArrayList key : map5.get(minValue)) { //Find the minimum value in HashMap 'map5'
    mapToArray = key;
}

done = true;
//Set variable 'done' to true. End process.

```

```

    }
    map2 = map4;
    map3.clear();

    //reset conditions for reiteration of do...while loop

    if (done == true) {
        break;
    }
    } while (done == false);
}
return Methods.getMod12(mapToArray);
}

//end method getNormal

public static ArrayList getPrime(ArrayList list) {
    ArrayList<Double> data = new ArrayList(list);
    double d;
    double e;
    ArrayList<Double> datanorm = new ArrayList(Methods.getNormal(data)); //get normal form of data
    ArrayList<Double> datanormzero = new ArrayList(Methods.toZero(datanorm)); //set the normalform to zero
    //get normal form of the inversion of datanorm
    //find inversion by subtracting 12 from each value
    ArrayList<Double> prime = new ArrayList();
    for (int i = 0; i < datanorm.size(); i++) {
        prime.add(Math.abs(data.get(i) - 12));
    }
    //close forloop
    //this may result in answers of 12 instead of zero
    //convert any answers of 12 to zero
    for (int i = 0; i < prime.size(); i++) {
        if (prime.get(i) == 12) {
            prime.set(i, 0.0);
        }
    }
    //close if statement
    //close forloop
    ArrayList<Double> dataprime = new ArrayList(Methods.getNormal(prime)); //get normal form of prime
    ArrayList<Double> dataprimezero = new ArrayList(Methods.toZero(dataprime)); //transpose dataprime to zero

    //determine which is smaller, datanorm or dataprime
    d=net.cooleysekula.CAT.Methods.sum(datanormzero); //get sum of all values in 'datanormzero'
    e=net.cooleysekula.CAT.Methods.sum(dataprimezero); //get sum of all values in 'dataprimezero'

```

```

        if (d < e) {
            return datanormzero;
        } else {
            return dataprimezero;
        }
    }
} //end getPrime

public static ArrayList getContourPrime(ArrayList list) { //create method call; getContourPrime
    ArrayList<Double> data = new ArrayList(list); //declare all lists and variable
    ArrayList<Double> dataDup = new ArrayList(data);
    ArrayList<Double> dataFinal = new ArrayList(data);
    int minIndex;
    double minValue;
    double lastValue = 0.0;
    double placeholder = 0.0;

    for (int j = 0; j < data.size(); j++) { //for every element in 'data'
        minIndex = dataDup.indexOf(Collections.min(dataDup)); //find the index of the smallest value
        minValue = dataDup.get(Integer.valueOf(minIndex)); //add this value to 'minValue'

        if (data.contains(minValue)) { //if 'data' contained the 'minValue' ...
            for (int i = 0; i < data.size(); i++) { //then look through every element in 'data'
                if (data.get(i).equals(minValue)) { //and whenever an element matching MinValue is found
                    dataFinal.set(data.indexOf(data.get(i)), placeholder); //replace it with the value of placeholder
                }
            }
        }

        dataDup.remove(minValue); //remove the lowest value we just used
        placeholder++; //increase the placeholder by 1
    }

    return dataFinal; //return 'dataFinal'
} //end getContourPrime

} //End public class Methods
}

```

# Appendix F

## Source Code: forte.class

```
package net.cooleysekula.CAT;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 *
 * @author katesekula
 */

public class Forte {
    public static String getForteNumber(ArrayList list) {
        //create public class Forte
        //create method for getForteNumber
        //The data in HashMap was taken from Jay Tonlin's code for his set theory calculator.
        //The data was reviewed and edited for correctness
        HashMap<String, String> fortelhash = new HashMap();
    }
}
```

fortehash.put("0,1,2", "3-1");  
fortehash.put("0,1,3", "3-2");  
fortehash.put("0,1,4", "3-3");  
fortehash.put("0,1,5", "3-4");  
fortehash.put("0,1,6", "3-5");  
fortehash.put("0,2,4", "3-6");  
fortehash.put("0,2,5", "3-7");  
fortehash.put("0,2,6", "3-8");  
fortehash.put("0,2,7", "3-9");  
fortehash.put("0,3,6", "3-10");  
fortehash.put("0,3,7", "3-11");  
fortehash.put("0,4,8", "3-12");  
fortehash.put("0,1,2,3", "4-1");  
fortehash.put("0,1,2,4", "4-2");  
fortehash.put("0,1,3,4", "4-3");  
fortehash.put("0,1,2,5", "4-4");  
fortehash.put("0,1,2,6", "4-5");  
fortehash.put("0,1,2,7", "4-6");  
fortehash.put("0,1,4,5", "4-7");  
fortehash.put("0,1,5,6", "4-8");  
fortehash.put("0,1,6,7", "4-9");  
fortehash.put("0,2,3,5", "4-10");  
fortehash.put("0,1,3,5", "4-11");  
fortehash.put("0,2,3,6", "4-12");  
fortehash.put("0,1,3,6", "4-13");  
fortehash.put("0,2,3,7", "4-14");  
fortehash.put("0,1,4,6", "4-Z15");  
fortehash.put("0,1,5,7", "4-16");  
fortehash.put("0,3,4,7", "4-17");  
fortehash.put("0,1,4,7", "4-18");  
fortehash.put("0,1,4,8", "4-19");  
fortehash.put("0,1,5,8", "4-20");  
fortehash.put("0,2,4,6", "4-21");  
fortehash.put("0,2,4,7", "4-22");  
fortehash.put("0,2,5,7", "4-23");  
fortehash.put("0,2,4,8", "4-24");  
fortehash.put("0,2,6,8", "4-25");

```

fortehash.put("0,3,5,8", "4-26");
fortehash.put("0,2,5,8", "4-27");
fortehash.put("0,3,6,9", "4-28");
fortehash.put("0,1,3,7", "4-Z29");
fortehash.put("0,1,2,3,4", "5-1");
fortehash.put("0,1,2,3,5", "5-2");
fortehash.put("0,1,2,4,5", "5-3");
fortehash.put("0,1,2,3,6", "5-4");
fortehash.put("0,1,2,3,7", "5-5");
fortehash.put("0,1,2,5,6", "5-6");
fortehash.put("0,1,2,6,7", "5-7");
fortehash.put("0,2,3,4,6", "5-8");
fortehash.put("0,1,2,4,6", "5-9");
fortehash.put("0,1,3,4,6", "5-10");
fortehash.put("0,2,3,4,7", "5-11");
fortehash.put("0,1,3,5,6", "5-Z12");
fortehash.put("0,1,2,4,8", "5-13");
fortehash.put("0,1,2,5,7", "5-14");
fortehash.put("0,1,2,6,8", "5-15");
fortehash.put("0,1,3,4,7", "5-16");
fortehash.put("0,1,3,4,8", "5-Z17");
fortehash.put("0,1,4,5,7", "5-Z18");
fortehash.put("0,1,3,6,7", "5-19");
fortehash.put("0,1,5,6,8", "5-20");
fortehash.put("0,1,4,5,8", "5-21");
fortehash.put("0,1,4,7,8", "5-22");
fortehash.put("0,2,3,5,7", "5-23");
fortehash.put("0,1,3,5,7", "5-24");
fortehash.put("0,2,3,5,8", "5-25");
fortehash.put("0,2,4,5,8", "5-26");
fortehash.put("0,1,3,5,8", "5-27");
fortehash.put("0,2,3,6,8", "5-28");
fortehash.put("0,1,3,6,8", "5-29");
fortehash.put("0,1,4,6,8", "5-30");
fortehash.put("0,1,3,6,9", "5-31");
fortehash.put("0,1,4,6,9", "5-32");
fortehash.put("0,2,4,6,8", "5-33");

```

```

fortehash.put("0,2,4,6,9", "5-34");
fortehash.put("0,2,4,7,9", "5-35");
fortehash.put("0,1,2,4,7", "5-Z36");
fortehash.put("0,3,4,5,8", "5-Z37");
fortehash.put("0,1,2,5,8", "5-Z38");
fortehash.put("0,1,2,3,4,5", "6-1");
fortehash.put("0,1,2,3,4,6", "6-2");
fortehash.put("0,1,2,3,5,6", "6-Z3");
fortehash.put("0,1,2,4,5,6", "6-Z4");
fortehash.put("0,1,2,3,6,7", "6-5");
fortehash.put("0,1,2,5,6,7", "6-Z6");
fortehash.put("0,1,2,6,7,8", "6-7");
fortehash.put("0,2,3,4,5,7", "6-8");
fortehash.put("0,1,2,3,5,7", "6-9");
fortehash.put("0,1,3,4,5,7", "6-Z10");
fortehash.put("0,1,2,4,5,7", "6-Z11");
fortehash.put("0,1,2,4,6,7", "6-Z12");
fortehash.put("0,1,3,4,6,7", "6-Z13");
fortehash.put("0,1,3,4,5,8", "6-14");
fortehash.put("0,1,2,4,5,8", "6-15");
fortehash.put("0,1,4,5,6,8", "6-16");
fortehash.put("0,1,2,4,7,8", "6-Z17");
fortehash.put("0,1,2,5,7,8", "6-18");
fortehash.put("0,1,3,4,7,8", "6-Z19");
fortehash.put("0,1,4,5,8,9", "6-20");
fortehash.put("0,2,3,4,6,8", "6-21");
fortehash.put("0,1,2,4,6,8", "6-22");
fortehash.put("0,2,3,5,6,8", "6-Z23");
fortehash.put("0,1,3,4,6,8", "6-Z24");
fortehash.put("0,1,3,5,6,8", "6-Z25");
fortehash.put("0,1,3,5,7,8", "6-Z26");
fortehash.put("0,1,3,4,6,9", "6-27");
fortehash.put("0,1,3,5,6,9", "6-Z28");
fortehash.put("0,2,3,6,7,9", "6-Z29");
fortehash.put("0,1,3,6,7,9", "6-30");
fortehash.put("0,1,4,5,7,9", "6-31");
fortehash.put("0,2,4,5,7,9", "6-32");

```



```

fortehash.put("0,2,3,5,7,9", "6-33");
fortehash.put("0,1,3,5,7,9", "6-34");
fortehash.put("0,2,4,6,8,10", "6-35");
fortehash.put("0,1,2,3,4,7", "6-36");
fortehash.put("0,1,2,3,4,8", "6-37");
fortehash.put("0,1,2,3,7,8", "6-38");
fortehash.put("0,2,3,4,5,8", "6-39");
fortehash.put("0,1,2,3,5,8", "6-40");
fortehash.put("0,1,2,3,6,8", "6-41");
fortehash.put("0,1,2,3,6,9", "6-42");
fortehash.put("0,1,2,5,6,8", "6-43");
fortehash.put("0,1,2,5,6,9", "6-44");
fortehash.put("0,2,3,4,6,9", "6-45");
fortehash.put("0,1,2,4,6,9", "6-46");
fortehash.put("0,1,2,4,7,9", "6-47");
fortehash.put("0,1,2,5,7,9", "6-48");
fortehash.put("0,1,3,4,7,9", "6-49");
fortehash.put("0,1,4,6,7,9", "6-50");
fortehash.put("0,1,2,3,4,5,6", "7-1");
fortehash.put("0,1,2,3,4,5,7", "7-2");
fortehash.put("0,1,2,3,4,5,8", "7-3");
fortehash.put("0,1,2,3,4,6,7", "7-4");
fortehash.put("0,1,2,3,5,6,7", "7-5");
fortehash.put("0,1,2,3,4,7,8", "7-6");
fortehash.put("0,1,2,3,6,7,8", "7-7");
fortehash.put("0,2,3,4,5,6,8", "7-8");
fortehash.put("0,1,2,3,4,6,8", "7-9");
fortehash.put("0,1,2,3,4,6,9", "7-10");
fortehash.put("0,1,3,4,5,6,8", "7-11");
fortehash.put("0,1,2,3,4,7,9", "7-12");
fortehash.put("0,1,2,4,5,6,8", "7-13");
fortehash.put("0,1,2,3,5,7,8", "7-14");
fortehash.put("0,1,2,4,6,7,8", "7-15");
fortehash.put("0,1,2,3,5,6,9", "7-16");
fortehash.put("0,1,2,4,5,6,9", "7-17");
fortehash.put("0,1,4,5,6,7,9", "7-18");
fortehash.put("0,1,2,3,6,7,9", "7-19");

```

```

fortehash.put("0,1,2,5,6,7,9", "7-20");
fortehash.put("0,1,2,4,5,8,9", "7-21");
fortehash.put("0,1,2,5,6,8,9", "7-22");
fortehash.put("0,2,3,4,5,7,9", "7-23");
fortehash.put("0,1,2,3,5,7,9", "7-24");
fortehash.put("0,2,3,4,6,7,9", "7-25");
fortehash.put("0,1,3,4,5,7,9", "7-26");
fortehash.put("0,1,2,4,5,7,9", "7-27");
fortehash.put("0,1,3,5,6,7,9", "7-28");
fortehash.put("0,1,2,4,6,7,9", "7-29");
fortehash.put("0,1,2,4,6,8,9", "7-30");
fortehash.put("0,1,3,4,6,7,9", "7-31");
fortehash.put("0,1,3,4,6,8,9", "7-32");
fortehash.put("0,1,2,4,6,8,10", "7-33");
fortehash.put("0,1,3,4,6,8,10", "7-34");
fortehash.put("0,1,3,5,6,8,10", "7-35");
fortehash.put("0,1,2,3,5,6,8", "7-Z36");
fortehash.put("0,1,3,4,5,7,8", "7-Z37");
fortehash.put("0,1,2,4,5,7,8", "7-Z38");
fortehash.put("0,1,2,3,4,5,6,7", "8-1");
fortehash.put("0,1,2,3,4,5,6,8", "8-2");
fortehash.put("0,1,2,3,4,5,6,9", "8-3");
fortehash.put("0,1,2,3,4,5,7,8", "8-4");
fortehash.put("0,1,2,3,4,6,7,8", "8-5");
fortehash.put("0,1,2,3,5,6,7,8", "8-6");
fortehash.put("0,1,2,3,4,5,8,9", "8-7");
fortehash.put("0,1,2,3,4,7,8,9", "8-8");
fortehash.put("0,1,2,3,6,7,8,9", "8-9");
fortehash.put("0,2,3,4,5,6,7,9", "8-10");
fortehash.put("0,1,2,3,4,5,7,9", "8-11");
fortehash.put("0,1,3,4,5,6,7,9", "8-12");
fortehash.put("0,1,2,3,4,6,7,9", "8-13");
fortehash.put("0,1,2,4,5,6,7,9", "8-14");
fortehash.put("0,1,2,3,4,6,8,9", "8-Z15");
fortehash.put("0,1,2,3,5,7,8,9", "8-16");
fortehash.put("0,1,3,4,5,6,8,9", "8-17");
fortehash.put("0,1,2,3,5,6,8,9", "8-18");

```

```

    fortehash.put(" 0,1,2,4,5,6,8,9 ", "8-19");
    fortehash.put(" 0,1,2,4,5,7,8,9 ", "8-20");
    fortehash.put(" 0,1,2,3,4,6,8,10 ", "8-21");
    fortehash.put(" 0,1,2,3,5,6,8,10 ", "8-22");
    fortehash.put(" 0,1,2,3,5,7,8,10 ", "8-23");
    fortehash.put(" 0,1,2,4,5,6,8,10 ", "8-24");
    fortehash.put(" 0,1,2,4,6,7,8,10 ", "8-25");
    fortehash.put(" 0,1,3,4,5,7,8,10 ", "8-26");
    fortehash.put(" 0,1,2,4,5,7,8,10 ", "8-27");
    fortehash.put(" 0,1,3,4,6,7,9,10 ", "8-28");
    fortehash.put(" 0,1,2,3,5,6,7,9 ", "8-Z29");
    fortehash.put(" 0,1,2,3,4,5,6,7,8 ", "9-1");
    fortehash.put(" 0,1,2,3,4,5,6,7,9 ", "9-2");
    fortehash.put(" 0,1,2,3,4,5,6,8,9 ", "9-3");
    fortehash.put(" 0,1,2,3,4,5,7,8,9 ", "9-4");
    fortehash.put(" 0,1,2,3,4,6,7,8,9 ", "9-5");
    fortehash.put(" 0,1,2,3,4,5,6,8,10 ", "9-6");
    fortehash.put(" 0,1,2,3,4,5,7,8,10 ", "9-7");
    fortehash.put(" 0,1,2,3,4,6,7,8,10 ", "9-8");
    fortehash.put(" 0,1,2,3,5,6,7,8,10 ", "9-9");
    fortehash.put(" 0,1,2,3,4,6,7,9,10 ", "9-10");
    fortehash.put(" 0,1,2,3,5,6,7,9,10 ", "9-11");
    fortehash.put(" 0,1,2,4,5,6,8,9,10 ", "9-12");
    fortehash.put(" 0,1,2,3,4,5,6,7,8,9 ", "10-1");
    fortehash.put(" 0,1,2,3,4,5,6,7,8,10 ", "10-2");
    fortehash.put(" 0,1,2,3,4,5,6,7,9,10 ", "10-3");
    fortehash.put(" 0,1,2,3,4,5,6,8,9,10 ", "10-4");
    fortehash.put(" 0,1,2,3,4,5,7,8,9,10 ", "10-5");
    fortehash.put(" 0,1,2,3,4,6,7,8,9,10 ", "10-6");
    fortehash.put(" 0,1,2,3,4,5,6,7,8,9,10 ", "11-1");
    fortehash.put(" 0,1,2,3,4,5,6,7,8,9,10,11 ", "12-1");
    fortehash.put("None", "N/A");
}

ArrayList<Double> data = new ArrayList(list);
List<Integer> fortенumber = new ArrayList();

if (data.size() > 2 && data.size() < 13) {
    //ends creation of fortehash
    //create ArrayList 'data' to place user data into
    //create List 'fortenumber' to add SC info to

    //if 'data' is greater than 2 and less than 13...

```

```

for (int i = 0; i < data.size(); i++) {
    fortunumber.add(data.get(i).intValue());
}

String string = fortunumber.toString().replace("[", "").replace("]", "").replace(" ", "");
return fortunumber.get(string);
} else {
    //if 'data' is less than 2 or greater than 12...
    return fortunumber.get("None");
}

//end method getForteNumber()
//end class Forte

```

# Appendix G

## Source Code: schultz.class

```
package net.cooleysekula.CAT;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 *
 * @author katesekula
 */
public class Schultz {

    public static ArrayList getSchultzCont (ArrayList list) {
        ArrayList<Double> data = new ArrayList(list);
        ArrayList<Double> valuesmap = new ArrayList();
        List<Integer> keysmap = new ArrayList();

        //Create class named 'Schultz'
        //Create method named 'getSchultzCont'
        //Declare all ArrayLists and variables used
```

```

ArrayList<Integer> keymap2 = new ArrayList();
LinkedHashMap<Integer, Double> hashdata = new LinkedHashMap();
LinkedHashMap<Integer, Double> maxlist = new LinkedHashMap();
LinkedHashMap<Integer, Double> minlist = new LinkedHashMap();
LinkedHashMap<Integer, Double> unflagged = new LinkedHashMap();
LinkedHashMap<Integer, Double> test = new LinkedHashMap();
HashMap<Integer, Double> hashmapfinal = new HashMap();

int N = 0; //N is the variable for measuring contour depth
int i; //i is a variable for looping
int j = 0;
int next = 0; //Variable for generating keys for hashmap of the data input
int yes;
int no;
int first ;
int last ;
int endloop;
int maxKey;
int minKey;
double maxVal;
double minVal;

for (i = 0; i < data.size(); i++) {
    hashdata.put(next, data.get(i));
    next++;
}

for (i = 0; i < data.size(); ) {

    next = 0;
    maxlist.put(next, data.get(0));
    next++;
    for (i = 0; i < data.size() - 2; i++) {
        if ((data.get(i + 1) >= data.get(i)
            && data.get(i + 1) >= data.get(i + 2))) {
            //FORLOOP for steps 1-5
            //***Step 1: Flag all maxima in C upwards; call the result 'max-list'
            //Iterate through 'hashdata.' Find maxima.
            //Place all non-maxima into the hashmap 'unflagged.'
            //Place all maxima into the hashmap 'maxlist,'
            //Every time we loop, reset integer 'next' to zero
            //Guarantees that the first pitch from 'data' will be retained in 'hashmax.'
            //Increment 'next' by 1
            //Iterate through the data list
            //Compare each pitch value to the pitch values on either side

```

```

        maxlist.put(next, data.get(i + 1));
    } else {
    }
    next++;
}

maxlist.put(next, (data.get(data.size() - 1)));

next = 0;
minlist.put(next, data.get(0));
next++;
for (i = 0; i < data.size() - 2; i++) {
    if ((data.get(i + 1) <= data.get(i)
        && data.get(i + 1) <= data.get(i + 2))) {
        minlist.put(next, data.get(i + 1));
    } else {
    }
    next++;
}

minlist.put(next, (data.get(data.size() - 1)));

}

//If the central pitch is a maximum, add it to 'hashmax.' (pitch flagged)
//If the central pitch is not a maximum, add it to 'unflagged.'

//Guarantees that last pitch of 'data' will be retained in 'hashmax.'
//***Step 2: Flag all minima in C downwards; call the result min-list
//Reset integer 'next' to zero
//Guarantees that first pitch of 'data' will be retained in 'hashmin.'
//Increment 'next' by 1
//Iterate through 'data'

//Compare each pitch value to the pitch values on either side
//If the central element is a minimum, add it to 'hashmin.' (pitch flagged)
//If the central pitch is not a minimum, add it to 'unflagged.'

//Guarantees that the last pitch from 'data' will be retained in 'hashmin.'
//***Step 3: If all c-pitches are flagged, go to step 6
//Compare 'hashdata' with the combined 'minlist' and 'maxlist.'
//If they are not the same size, then add the extra cps of ...
//...'hashdata' to 'unflagged.'

test.putAll(maxlist);
test.putAll(minlist);

for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) { //Create a keymap for 'hashdata'
    keymap.add(entry.getKey());
}

for (Map.Entry<Integer,Double> entry : test.entrySet()) { //Create a keymap for 'test'
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap.size(); i++) {
    if (keymap2.contains(keymap.get(i))) {
    } else {
        //If a key from 'test' is not in 'hashdata', add it to 'unflagged'
    }
}

```

```

        unflagged.put(keysm2.get(i),hashdata.get(keysm2.get(i)));
    }
}

if (unflagged.isEmpty()) {
    break;
} else {
    //If there are no unflagged pitch, skip to step 6.
    //Break from the current loop and procede to step 6.
    //If there are unflagged pitches, continue to step 4.

    test.clear();

    /**step 4: Delete all non-flagged c-pitches in C
    //Iterate through 'hashdata' and remove any entires from 'unflagged'
    keysm2 = new ArrayList();
    for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) {
        keysm2.add(entry.getKey());
    }

    keysm2 = new ArrayList();
    for (Map.Entry<Integer,Double> entry : unflagged.entrySet()) {
        keysm2.add(entry.getKey());
    }

    for (i = 0; i < keysm2.size(); i++) {
        if (keysm2.contains(keysm2.get(i))) {
            hashdata.remove(keysm2.get(i));
        }
    }

    N++;
    break;
}

//step 5: N is incremented by 1

unflagged.clear();

//Before continuing, clear the contents of 'unflagged.'
//*****
/**Step 6: Flag all the maxima of the max-list.
//For any string of equal and adjacent maxima in the max-list...
//... flag all of them, unless:

```



```

// (1) one c-pitch in the string is the first or last c-pitch of C,
// then flag only it; or (2) both the first and last c-pitches of C ...
// ... are in the string, then flag (only) both the first and last c-pitches of C.

// My understanding of Steps 6 and 7: They are the same as Steps 1 and 2,
// but if there are repeated maxima or minima chains that begin with
// the first cp or end with the last cp, delete every repetition ...
// ... except for the first or last cp.
// The integer 'yes' will tell us if a change occurs during Steps 6 and 7
// Find the maxima of the maxlist and the minima of the minlist
// Create a keymap for the 'maxlist.'

yes = 0;

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) {
    keymap.add(entry.getKey());
}

for (i = 0; i < keymap.size() - 2; i++) {
    // Iterate through 'hashmax' to find max of max values
    if ((maxlist.get(keymap.get(i+1)) >= maxlist.get(keymap.get(i)))
        && (maxlist.get(keymap.get(i+1)) >= maxlist.get(keymap.get(i+2)))) {
    } else {
        unflagged.put(keymap.get(i+1), maxlist.get(keymap.get(i+1)));
        yes++;
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry: maxlist.entrySet()) {
    keymap.add(entry.getKey());
}

keymap2 = new ArrayList();
for (Map.Entry<Integer, Double> entry: unflagged.entrySet()) {
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        maxlist.remove(keymap2.get(i));
    }
}

```

```

    }
}

next = 1;
j=1;
keysmap = new ArrayList();
for (Map.Entry<Integer,Double> entry: maxlist.entrySet()){
    keysmap.add(entry.getKey());
}

while (next ==1) {
    for ( i = 0; i < keysmap.size()-1; ) {
        if (maxlist.get(keysmap.get(i+1)).equals(maxlist.get(keysmap.get(i)))
            && keysmap.get(i).equals(0)
            && keysmap.get(i+1)!=keysmap.get(keysmap.size()-1)){
            unflagged.put(keysmap.get(i+1),maxlist.get(keysmap.get(i+1)));
            keysmap.remove(keysmap.get(i+1));
            next = 1;
            yes++;
        } else {
            next = 0;
            break;
        }
    }
}

keysmap = new ArrayList();
for (Map.Entry<Integer,Double> entry: maxlist.entrySet()){
    keysmap.add(entry.getKey());
}

for ( i = keysmap.size()-1; i >= 1; i-- ) {
    if (maxlist.get(keysmap.get(i-1)).equals(maxlist.get(keysmap.get(i)))
        && keysmap.get(i).equals(keysmap.get(keysmap.size()-1))
        && keysmap.get(i-1)!=keysmap.get(0)){
        unflagged.put(keysmap.get(i-1), maxlist.get(keysmap.get(i-1)));
        keysmap.remove(keysmap.get(i-1));
    }
}

```

//Look for equal and adjacent c-pitches in the maxlist

```

        yes++;
    } else {
    }
}

//***Step 7: Flag all the minima of the min-list.
//For any string of equal and adjacent minima in the min-list...
//... flag all of them, unless:
// (1) one c-pitch in the string is the first or last c-pitch of C,
//... then flag only it; or (2) both the first and last c-pitches of C...
//... are in the string, then flag (only) both the first ...
//... and last c-pitches of C.
//Clear contents of keymap
keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) {
    keymap.add(entry.getKey());
}

for (i = 0; i < keymap.size()-2; i++) {
    if ((minlist.get(keymap.get(i+1)) <= minlist.get(keymap.get(i)))
        && (minlist.get(keymap.get(i+1)) <= minlist.get(keymap.get(i+2)))) {
    } else {
        unflagged.put(keymap.get(i+1), minlist.get(keymap.get(i+1)));
        yes++;
    }
}

keymap = new ArrayList();
//Remove 'unflagged' from 'minlist'
for (Map.Entry<Integer,Double> entry: minlist.entrySet()){
    keymap.add(entry.getKey());
}

keymap2 = new ArrayList();
for (Map.Entry<Integer,Double> entry: unflagged.entrySet()){
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++){
    if (keymap.contains(keymap2.get(i))){

```

```

        minlist.remove(keysm2.get(i));
    }
}

next = 1;
keysm = new ArrayList();
for (Map.Entry<Integer,Double> entry: minlist.entrySet()){
    keysm.add(entry.getKey());
}

while (next == 1) {
    for (i = 0; i < keysm.size() - 1; ) {
        if (minlist.get(keysm.get(i + 1)).equals(minlist.get(keysm.get(i)))
            && keysm.get(i).equals(0)
            && keysm.get(i + 1) != keysm.get(keysm.size() - 1)) {
            unflagged.put(keysm.get(i + 1), minlist.get(keysm.get(i + 1)));
            keysm.remove(keysm.get(i + 1));
            next = 1;
            yes++;
        } else {
            next = 0;
            break;
        }
    }
}

keysm = new ArrayList();
for (Map.Entry<Integer,Double> entry: minlist.entrySet()){
    keysm.add(entry.getKey());
}

for (i = keysm.size() - 1; i >= 1; i--) {
    if (minlist.get(keysm.get(i - 1)).equals(minlist.get(keysm.get(i)))
        && keysm.get(i).equals(keysm.get(keysm.size() - 1))
        && keysm.get(i - 1) != keysm.get(0)) {
        unflagged.put(keysm.get(i - 1), minlist.get(keysm.get(i - 1)));
        keysm.remove(keysm.get(i - 1));
    }
}

```

//Look for equal and adjacent c-pitches in the minlist

```

        yes++;
    } else {
    }
}

}

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : hashdata.entrySet()) {
    keymap.add(entry.getKey());
}

for (i = 1; i < keymap.size() - 2; i++) {
    if (hashdata.get(keymap.get(i + 1)).equals(hashdata.get(keymap.get(i)))) {
        unflagged.put(keymap.get(i+1), hashdata.get(keymap.get(i+1)));
        yes++;
    }
}

}

/**Step 8: For any string of equal and adjacent ...
// ... maxima in the max-list in which no minima intervene,
// ... remove the flag from all but (any) one c-pitch in the string.
/**Step 9: for any string of equal and adjacent
// ... minima in the min-list in which no maxima intervene,
// ... remove the flag from all but (any) one c-pitch in the string.
//Create a keymap for 'hashdata,' which was updated in Step 4.
//Clear the keymap

//Determine if there are duplicate values remaining ...
// ... other than the second or penultimate pitch in 'hashdata'

/**Step 10: If all c-pitches are flagged, and no more than one c-pitch...
// ... repetition in the max-list and min-list (combined)...
// ... exists, not including the first and last c-pitches of C...
// ... proceed directly to step 17.
//My understanding of Step 10: If there are repetitions in either list,
// ... then continue to Step 11.

//If the above process in Steps 6-9 added any values to 'unflagged' ...
// ... it means that not every pitch is flagged ...
// ... or that duplicates exist. Remove the members of 'unflagged'...
// ... from 'maxlist' and 'minlist.'
//Now we can examine those sets without the unflagged pitches...
// ... (the max of the 'maxlist' and the min of the 'minlist')

if (unflagged.isEmpty()) {

```

```

    } else {
    }

    //Look for duplicates in the maxlist.
    //To do that we remove the unflagged cps from the maxlist
    //Clear keysmap
    keysmap = new ArrayList();
    for (Map.Entry<Integer, Double> entry : unflagged.entrySet()) { //Add the keys of 'unflagged' to the keysmap
        keysmap.add(entry.getKey());
    }

    keysmap2.clear();
    for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Add the keys of hashmap 'maxlist' to keysmap2
        keysmap2.add(entry.getKey());
    }

    if (yes > 0) {
        for (i = 0; i < keysmap.size(); i++) {
            if (keysmap2.contains(keysmap.get(i))) {
                maxlist.remove(keysmap.get(i));
            }
        }
    }

    keysmap2.clear();
    for (Map.Entry<Integer, Double> entry : minlist.entrySet()) { //Add the keys of hashmap 'minlist' to keysmap2
        keysmap2.add(entry.getKey());
    }

    if (yes > 0) {
        for (i = 0; i < keysmap.size(); i++) {
            if (keysmap2.contains(keysmap.get(i))) {
                minlist.remove(keysmap.get(i));
            }
        }
    }
    yes = 0;

    //Reset the yes counter to zero

```

```

//Look for repetitions in the maxlist
keysmap = new ArrayList();
for (Map.Entry<Integer,Double> entry : maxlist.entrySet()) {
    keysmap.add(entry.getKey());
}

for (i = 1; i < keysmap.size()-2; i++) {
    if (maxlist.get(keysmap.get(i)).equals(maxlist.get(keysmap.get(i+1)))) {
        yes++;
    }
}

keysmap = new ArrayList();
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) {
    keysmap.add(entry.getKey());
}

//Clear the keysmap
//Create a keysmap for 'minlist'

for (i = 1; i < keysmap.size()-2; i++) {
    if (minlist.get(keysmap.get(i)).equals(minlist.get(keysmap.get(i+1)))) {
        yes++;
    }
}

if (unflagged.isEmpty() && yes <= 1) {
    hashmapfinal.clear();
    valuesmap = new ArrayList();
    hashmapfinal.putAll(maxlist);
    hashmapfinal.putAll(minlist);
    keysmap = new ArrayList();

    for (Map.Entry<Integer,Double> entry : hashmapfinal.entrySet()) {
        keysmap.add(entry.getKey());
    }

    Collections.sort(keysmap);

    for (i = 0; i < keysmap.size(); i++) {

```

```

        valuesmap.add(hashmapfinal.get(keysmmap.get(i)));
    }

    endloop = 1;
    break;

} else {
}

    /**Step 11: If more than one c-pitch repetition in the max-list
    //and/or min-list(combined) exists, not including the first and last
    //c-pitches of C, remove flags on all repeated c-pitches except those
    //closest to the first and last c-pitches of C

    //Locate duplicates in the 'maxlist'
    //Clear the contents of the keysmmap
    for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Create a keysmmap for 'maxlist'
        keysmmap.add(entry.getKey());
    }

    keysmmap2 = new ArrayList();
    //Clear the contents of keysmmap2
    for (Map.Entry<Integer, Double> entry : hashdata.entrySet()) { //Create a keysmmap for 'hashdata'
        keysmmap2.add(entry.getKey());
    }

    for (i = 1; i < keysmmap.size() - 2; i++) {
        yes = 0;
        no = 0;
        first = (keysmmap2.get(1));
        last = (keysmmap2.get(keysmmap2.size()-2));

        if (maxlist.get(keysmmap.get(i)).equals(maxlist.get(keysmmap.get(i + 1)))) { //If adjacent pitch values are the same...
            if ((keysmmap.get(i).equals( first ))
                && (keysmmap.get(i+1) != last)){
                unflagged.put(keysmmap.get(i+1), maxlist.get(keysmmap.get(i+1)));
                yes++;
            }
        }
    }
}

```



```

    }
    if ((keymap.get(i+1).equals(last))
        && (keymap.get(i) != first)){
        unflagged.put(keymap.get(i),maxlist.get(keymap.get(i)));
        yes++;
    }
    if (keymap.get(i).equals( first )
        && keymap.get(i+1).equals(last)){
        no++;
        // ... do nothing
    }
    if ((yes == 0) && (no == 0)) {
        unflagged.put(keymap.get(i), maxlist.get(keymap.get(i)));
        unflagged.put(keymap.get(i + 1), maxlist.get(keymap.get(i + 1)));
        yes++;
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) { //Create a keymap for the 'minlist'
    keymap.add(entry.getKey());
}

// ... then, determine if there are duplicates in the 'minlist '
//Clear the contents of the keymap
//Iterate through the keymap for 'minlist'
for (i = 1; i < keymap.size() - 2; i++) {
    yes = 0;
    no = 0;
    first = (keymap2.get(1));
    last = (keymap2.get(keymap2.size()-2));

    if (minlist.get(keymap.get(i)).equals(minlist.get(keymap.get(i + 1)))) { //If adjacent pitch values are the same...
        if ((keymap.get(i).equals( first ))
            && (keymap.get(i+1) != last)){
            unflagged.put(keymap.get(i+1), minlist.get(keymap.get(i+1)));
            yes++;
        }
    }
}

```

```

    if ((keymap.get(i+1).equals(last))
        && (keymap.get(i) != first)) {
        unflagged.put(keymap.get(i).minlist.get(keymap.get(i)));
        yes++;
    }
    if (keymap.get(i).equals( first )
        && keymap.get(i+1).equals(last)) {
        no++;
    }
    // ...do nothing
}
if ((yes == 0) && (no == 0)) {
    unflagged.put(keymap.get(i).minlist.get(keymap.get(i)));
    unflagged.put(keymap.get(i + 1).minlist.get(keymap.get(i + 1)));
    yes++;
}
}
}

//If hashdata is 4 pcs long, proceed to step 12...
// ... Else, proceed to Step 13.
//*****
//First, we need to learn what the new hashdata will be if we removed...
// ... all the unflagged cps right now add the contents...
// ... of 'hashdata' to 'test'
//Clear the contents of keymap
//Clear the contents of keymap2

test = new LinkedHashMap(hashdata);
keymap = new ArrayList();
keymap2 = new ArrayList();

for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) { //Create a keymap for 'hashdata'
    keymap.add(entry.getKey());
}

for (Map.Entry<Integer,Double> entry : unflagged.entrySet()) { //Create a keymap for 'unflagged'
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        test.remove(keymap2.get(i));
    }
}
//If an unflagged cp is a member of 'hashdata'...
// ... remove it from our test hash 'test'

```

```

    }
}

if (test.size()==4) {

    //If the test hash 'test' is 4 cps long, then we need to goto step 12.
    //***step 12: If both flagged c-pitches remaining from step 11...
    // ... are members of the max-list,
    // ... flag any one (and only one) former member of the...
    // ... min-list whose flag was removed in step 11; if both c-pitches...
    // ... are members of the min-list,
    // ... flag any one (and only one) former member...
    // ... of the max-list whose flag was removed in step 11
    //Clear the contents of keymap

    keymap = new ArrayList();
    List<Integer> maxkey = new ArrayList();
    List<Integer> minkey = new ArrayList();

    for (Map.Entry<Integer, Double> entry : test.entrySet()) { //Create a keymap for 'test'
        keymap.add(entry.getKey());
    }

    keymap.remove(keymap.get(0)); //Remove the first pitch from the keymap
    keymap.remove(keymap.get(keymap.size()-1)); //Remove the last pitch from the keymap

    for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Create a keymap for the maxlist
        maxkey.add(entry.getKey());
    }

    for (Map.Entry<Integer, Double> entry : minlist.entrySet()) { //Create a keymap for the minlist
        minkey.add(entry.getKey());
    }

    yes = 0;
    for (i = 0; i < maxkey.size(); i++) {
        if (keymap.contains(maxkey.get(i))) {
            yes++;
        }
    }

    //If any of the items in 'maxlist' are a member of test...
    // ... (disregarding the first and last cps) ...
    // ... then increase the 'yes' counter

```

```

if (yes == 0) {
    maxValue = (Collections.max(maxlist.values()));
    maxKey = 0;
    for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) {
        if (maxValue == entry.getValue()) {
            maxKey = (entry.getKey());
        }
    }
    test.put(maxKey, maxValue);
}

// ... add it back to 'test'

// Now run the same process for the minlist
// Reset the yes counter
// If none of the items in min are a members of 'test' ...
// ... then increase the 'yes' counter

yes = 0;
for (i = 0; i < minkey.size(); i++) {
    if (keysmat.contains(minkey.get(i))) {
        yes++;
    }
}

if (yes == 0) {
    minValue = (Collections.min(minlist.values()));
    minKey = 0;
    for (Map.Entry<Integer, Double> entry : minlist.entrySet()) {
        if (minValue == entry.getValue()) {
            minKey = (entry.getKey());
        }
    }
    test.put(minKey, minValue);
}

// ... add it back to 'test'

// End Step 12

}
else {
}

keysmat = new ArrayList();
keysmat2 = new ArrayList();

for (Map.Entry<Integer, Double> entry : hashdata.entrySet()) { // Create a keysmat for 'hashdata'

```

```

        keymap.add(entry.getKey());
    }

    for (Map.Entry<Integer,Double> entry : unflagged.entrySet()) { //Create a keymap for 'unflagged'
        keymap2.add(entry.getKey());
    }

    //If an entry from 'unflagged' exists in 'hashdata,' remove it.

    for (i = 0; i < keymap2.size(); i++) {
        if (keymap.containsKey(keymap2.get(i))) {
            hashdata.remove(keymap2.get(i));
        }
    }

    //For the loop to repeat properly, all 'unflagged' cps ...
    // ...should also be removed from 'maxlist' and 'minlist'

    keymap = new ArrayList();
    for (Map.Entry<Integer,Double> entry : maxlist.entrySet()) { //Clear the contents of keymap and create a keymap for maxlist
        keymap.add(entry.getKey());
    }

    //If an entry from 'unflagged' exists in 'maxlist,' remove it.

    for (i = 0; i < keymap2.size(); i++) {
        if (keymap.containsKey(keymap2.get(i))) {
            maxlist.remove(keymap2.get(i));
        }
    }

    keymap = new ArrayList();
    for (Map.Entry<Integer,Double> entry : minlist.entrySet()) { //Clear the contents of keymap
        keymap.add(entry.getKey()); //Create a keymap for minlist
    }

    //If an entry from 'unflagged' exists in 'maxlist,' remove it.

    for (i = 0; i < keymap2.size(); i++) {
        if (keymap.containsKey(keymap2.get(i))) {
            minlist.remove(keymap2.get(i));
        }
    }

    //***Step 14: If N != 0, N is incremented by 1
    //***Step 15: if N = 0, N is incremented by 2

    if (N == 0) {

```

```

        N += 2;
    } else {
        N++;
    }

    //Step 16: Goto Step 6

    yes = 0;
    endloop = 0;
    unflagged.clear();
    if (endloop > 0) {
        break;
    }
}

//End forloop for steps 6–16
System.out.println("Method getSchultzCont completed successfully");
return valuesmap;
}

//END getSchultzCont

public static Integer getSchultzDepth (ArrayList list) {
    ArrayList<Double> data = new ArrayList(list);
    ArrayList<Double> valuesmap = new ArrayList();
    List<Integer> keymap = new ArrayList();
    ArrayList<Integer> keymap2 = new ArrayList();
    LinkedHashMap<Integer, Double> hashdata = new LinkedHashMap();
    LinkedHashMap<Integer, Double> maxlist = new LinkedHashMap();
    LinkedHashMap<Integer, Double> minlist = new LinkedHashMap();
    LinkedHashMap<Integer, Double> unflagged = new LinkedHashMap();
    LinkedHashMap<Integer, Double> test = new LinkedHashMap();
    HashMap<Integer, Double> hashmapfinal = new HashMap();
    int N = 0;
    int i;
    int j = 0;
    int next = 0;
    int yes;
    int no;
    int first;
    int last;
    int endloop;
    int maxKey;
    //N is the variable for measuring contour depth
    //i is a variable for looping
    //Variable for generating keys for hashmap of the data input

```

```

int minKey;
double maxVal;
double minVal;

for (i = 0; i < data.size(); i++) {
    hashdata.put(next, data.get(i));
    next++;
}

for (i = 0; i < data.size(); ) {

    next = 0;
    maxlist.put(next, data.get(0));
    next++;
    for (i = 0; i < data.size() - 2; i++) {
        if ((data.get(i + 1) >= data.get(i)
            && data.get(i + 1) >= data.get(i + 2))) {
            maxlist.put(next, data.get(i + 1));
        } else {
        }
        next++;
    }
    maxlist.put(next, (data.get(data.size() - 1)));

    next = 0;
    minlist.put(next, data.get(0));
    next++;
    for (i = 0; i < data.size() - 2; i++) {
        if ((data.get(i + 1) <= data.get(i)
            && data.get(i + 1) <= data.get(i + 2))) {
            minlist.put(next, data.get(i + 1));
        } else {
        }
        next++;
    }

    //Iterate through 'data' and place values into hashmap 'datahash'

    //FORLOOP for steps 1-5
    //***Step 1: Flag all maxima in C upwards; call the result 'max-list'
    //Iterate through 'hashdata.' Find maxima.
    //Place all non-maxima into the hashmap 'unflagged.'
    //Place all maxima into the hashmap 'maxlist.'
    //Every time we loop, reset integer 'next' to zero
    //Guarantees that the first pitch from 'data' will be retained in 'hashmax.'
    //Increment 'next' by 1
    //Iterate through the data list
    //Compare each pitch value to the pitch values on either side

    //If the central pitch is a maximum, add it to 'hashmax.' (like flagging)
    //If the central pitch is not a maximum, add it to 'unflagged.'

    //Guarantees that last pitch of 'data' will be retained in 'hashmax.'
    //***Step 2: Flag all minima in C downwards; call resulting set min-list
    //Reset integer 'next' to zero
    //Gaurantees that first pitch of 'data' will be retained in 'hashmin.'
    //Increment 'next' by 1
    //Iterate through 'data'

    //Compare each pitch value to the pitch values on either side
    //If the central element is a minimum, add it to 'hashmin.' (like flagging)
    //If the central pitch is not a minimum, add it to 'unflagged.'

```

```

    }
    minlist.put(next, (data.get(data.size() - 1)));
    //Guarantees that the last pitch from 'data' will be retained in 'hashmin.'
    //***Step 3: If all c-pitches are flagged, go to step 6
    //Compare 'hashdata' with the combined 'minlist' and 'maxlist.'
    //If they are not the same size, then add the extra cps of ...
    //...'hashdata' to 'unflagged.'

    test.putAll(maxlist);
    test.putAll(minlist);

    for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) { //Create a keymap for 'hashdata'
        keymap.add(entry.getKey());
    }

    for (Map.Entry<Integer,Double> entry : test.entrySet()) { //Create a keymap for 'test'
        keymap2.add(entry.getKey());
    }

    for (i = 0; i < keymap.size(); i++) {
        if (keymap2.contains(keymap.get(i))) {
        } else {
            unflagged.put(keymap.get(i), hashdata.get(keymap.get(i)));
        }
    }

    if (unflagged.isEmpty()) {
        break;
    } else {
    }
    test.clear();

    keymap = new ArrayList();
    for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) {
        keymap.add(entry.getKey());
    }

    keymap2 = new ArrayList();
    for (Map.Entry<Integer,Double> entry : unflagged.entrySet()) {

```



```

        keymap2.add(entry.getKey());
    }

    for (i = 0; i < keymap2.size(); i++) {
        if (keymap.contains(keymap2.get(i))) {
            hashdata.remove(keymap2.get(i));
        }
    }

    N++;
    break;
}

unflagged.clear();

//*****
//Before continuing, clear the contents of 'unflagged.'
//*****

//**Step 6: Flag all the maxima of the max-list.
//For any string of equal and adjacent maxima in the max-list,
//.. flag all of them,unless:
//(1) one c-pitch in the string is the first or last c-pitch of C,
//then flag only it;or (2) both the first and last c-pitches of C...
//... are in the string,then flag (only) both the first ...
//... and last c-pitches of C.

//My understanding of Steps 6 and 7: They are the same as Steps 1 and 2,
//but if there are repeated maxima or minima chains that begin with
//the first cp or end with the last cp, delete every repetition ...
//... except for the first or last cp.
//The integer 'yes' will tell us if a change occurs during Steps 6 and 7
//Find the maxima of the maxlist an the minima of the minlist
//Create a keymap for the 'maxlist.'

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry : maxlist.entrySet()) {
    keymap.add(entry.getKey());
}

yes = 0;

for (i = 0; i < keymap.size()-2; i++) {
    //Iterate through 'hashmax' to find max of max values

```

```

if ((maxlist.get(keysmat.get(i+1)) >= maxlist.get(keysmat.get(i)))
    && (maxlist.get(keysmat.get(i+1)) >= maxlist.get(keysmat.get(i+2)))) {
    } else {
        unflagged.put(keysmat.get(i+1), maxlist.get(keysmat.get(i+1)));
        yes++;
    }
}

keysmat = new ArrayList();
//Remove 'unflagged' from 'maxlist'
for (Map.Entry<Integer,Double> entry: maxlist.entrySet()) {
    keysmat.add(entry.getKey());
}

keysmat2 = new ArrayList();
for (Map.Entry<Integer,Double> entry: unflagged.entrySet()) {
    keysmat2.add(entry.getKey());
}

for (i = 0; i < keysmat2.size(); i++) {
    if (keysmat.contains(keysmat2.get(i))) {
        maxlist.remove(keysmat2.get(i));
    }
}

next = 1;
j=1;
keysmat = new ArrayList();
for (Map.Entry<Integer,Double> entry: maxlist.entrySet()) {
    keysmat.add(entry.getKey());
}

while (next ==1) {
    for (i = 0; i < keysmat.size()-1; ) {
        if (maxlist.get(keysmat.get(i+1)).equals(maxlist.get(keysmat.get(i)))
            && keysmat.get(i).equals(0)
            && keysmat.get(i+1)!=keysmat.get(keysmat.size()-1)) {
            unflagged.put(keysmat.get(i+1), maxlist.get(keysmat.get(i+1)));
        }
    }
}

```

```

        keymap.remove(keymap.get(i+1));
        next = 1;
        yes++;
    } else {
        next = 0;
        break;
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry: maxlist.entrySet()){
    keymap.add(entry.getKey());
}

for (i = keymap.size()-1; i >= 1; i--) {
    if (maxlist.get(keymap.get(i-1)).equals(maxlist.get(keymap.get(i)))
        && keymap.get(i).equals(keymap.get(keymap.size()-1))
        && keymap.get(i-1)!=keymap.get(0)){
        unflagged.put(keymap.get(i-1), maxlist.get(keymap.get(i-1)));
        keymap.remove(keymap.get(i-1));
        yes++;
    } else {
    }
}

/**Step 7: Flag all the minima of the min-list.
//For any string of equal and adjacent minima in the min-list,
// ... flag all of them,unless:
//(1) one c-pitch in the string is the first or last c-pitch of C,
// ... then flag only it ;or (2) both the first and last c-pitches of C ...
// ... are in the string,then flag (only) both the first ...
// .. and last c-pitches of C.
//Clear contents of keymap

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) {
    keymap.add(entry.getKey());
}

```

```

for (i = 0; i < keysm2.size() - 2; i++) {
    //Iterate through the minlist to find min of min values
    if ((minlist.get(keysm2.get(i+1)) <= minlist.get(keysm2.get(i)))
        && (minlist.get(keysm2.get(i+1)) <= minlist.get(keysm2.get(i+2)))) {
    } else {
        unflagged.put(keysm2.get(i+1), minlist.get(keysm2.get(i+1)));
        yes++;
    }
}

keysm2 = new ArrayList();
for (Map.Entry<Integer,Double> entry: minlist.entrySet()) {
    keysm2.add(entry.getKey());
}

keysm2 = new ArrayList();
for (Map.Entry<Integer,Double> entry: unflagged.entrySet()) {
    keysm2.add(entry.getKey());
}

for (i = 0; i < keysm2.size(); i++) {
    if (keysm2.contains(keysm2.get(i))) {
        minlist.remove(keysm2.get(i));
    }
}

next = 1;
keysm2 = new ArrayList();
for (Map.Entry<Integer,Double> entry: minlist.entrySet()) {
    keysm2.add(entry.getKey());
}

while (next == 1) {
    for (i = 0; i < keysm2.size() - 1; i) {
        if (minlist.get(keysm2.get(i + 1)).equals(minlist.get(keysm2.get(i)))
            && keysm2.get(i).equals(0)
            && keysm2.get(i + 1) != keysm2.get(keysm2.size() - 1)) {
            unflagged.put(keysm2.get(i + 1), minlist.get(keysm2.get(i + 1)));
        }
    }
}

```

```

        keymap.remove(keymap.get(i + 1));
        next = 1;
        yes++;
    } else {
        next = 0;
        break;
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry: minlist.entrySet()){
    keymap.add(entry.getKey());
}

for (i = keymap.size()-1; i >= 1; i--) {
    if (minlist.get(keymap.get(i-1)).equals(minlist.get(keymap.get(i)))
        && keymap.get(i).equals(keymap.get(keymap.size()-1))
        && keymap.get(i-1)!=keymap.get(0)) {
        unflagged.put(keymap.get(i-1), minlist.get(keymap.get(i-1)));
        keymap.remove(keymap.get(i-1));
        yes++;
    } else {
    }
}

/**Step 8: For any string of equal and adjacent ...
// ... maxima in the max-list in which no minima intervene,
// ... remove the flag from all but (any) one c-pitch in the string.
/**Step 9: for any string of equal and adjacent
// ... minima in the min-list in which no maxima intervene,
// ... remove the flag from all but (any) one c-pitch in the string.
//Create a keymap for 'hashdata,' which was updated in Step 4.
//Clear the keymap

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : hashdata.entrySet()) {
    keymap.add(entry.getKey());
}

//Determine if there are duplicate values remaining ...

```

```

for (i = 1; i < keymap.size() - 2; i++) {
    if (hashdata.get(keymap.get(i + 1)).equals(hashdata.get(keymap.get(i)))) {
        unflagged.put(keymap.get(i+1), hashdata.get(keymap.get(i+1)));
        yes++;
    }
}

//***Step 10: If all c-pitches are flagged, and no more than one c-pitch...
//... repetition in the max-list and min-list (combined)...
//... exists, not including the first and last c-pitches of C,...
//... proceed directly to step 17.
//My understanding of Step 10: If there are repetitions in either list,
//... then continue to Step 11.

//If the above process in Steps 6-9 added any values to 'unflagged'...
//... it means that not every pitch is flagged ...
//... or that duplicates exist. Remove the members of 'unflagged'...
//... from 'maxlist' and 'minlist'.
//Now we can examine those sets without the unflagged pitches...
//...(the max of the 'maxlist' and the min of the 'minlist')

//Look for duplicates in maxlist. Remove unflagged cps from maxlist
//Clear keymap

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : unflagged.entrySet()) { //Add the keys of 'unflagged' to the keymap
    keymap.add(entry.getKey());
}

keymap2.clear();
for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Add the keys of hashmap 'maxlist' to keymap2
    keymap2.add(entry.getKey());
}

if (yes > 0) {
    for (i = 0; i < keymap.size(); i++) {
        if (keymap2.contains(keymap.get(i))) {
            //If (yes > 0) and if 'maxlist' contains any of the keys of ...
            //...'unflagged,' remove them.
        }
    }
}

```

```

        maxlist.remove(keymap.get(i));
    }
}

keymap2.clear();
//Remove any unflagged cps from the minlist
//Clear keymap2
for (Map.Entry<Integer, Double> entry : minlist.entrySet()) { //Add the keys of hashmap 'minlist' to keymap2
    keymap2.add(entry.getKey());
}

if (yes > 0) {
    for (i = 0; i < keymap.size(); i++) {
        if (keymap2.contains(keymap.get(i))) {
            minlist.remove(keymap.get(i));
        }
    }
}
yes = 0;
//Reset the yes counter to zero

keymap = new ArrayList();
//Look for repetitions in the maxlist
//Clear the keymap
for (Map.Entry<Integer,Double> entry : maxlist.entrySet()) { //Create a keymap for 'maxlist'
    keymap.add(entry.getKey());
}

for (i = 1; i < keymap.size()-2; i++) {
    if (maxlist.get(keymap.get(i)).equals(maxlist.get(keymap.get(i+1)))) {
        yes++;
    }
}

keymap = new ArrayList();
//Clear the keymap
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) { //Create a keymap for 'minlist'
    keymap.add(entry.getKey());
}

```

```

for (i = 1; i < keymap.size()-2; i++) {
    //Look for repetitions (not including first and last cp) in 'minlist'
    if (minlist.get(keymap.get(i)).equals(minlist.get(keymap.get(i+1)))) {
        yes++;
    }
}

if (unflagged.isEmpty() && yes <= 1) {
    hashmapfinal.clear();
    valuesmap = new ArrayList();
    hashmapfinal.putAll(maxlist);
    hashmapfinal.putAll(minlist);
    keymap = new ArrayList();

    for (Map.Entry<Integer, Double> entry : hashmapfinal.entrySet()) {
        keymap.add(entry.getKey());
    }

    Collections.sort(keymap);

    for (i = 0; i < keymap.size(); i++) {
        valuesmap.add(hashmapfinal.get(keymap.get(i)));
    }

    endloop = 1;
    break;
} else {
}

yes = 0;

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Create a keymap for 'maxlist'

```

```

//**Step 11: If more than one c-pitch repetition in the max-list
//and/or min-list(combined) exists, not including the first and last
//c-pitches of C, remove flags on all repeated c-pitches except those
//closest to the first and last c-pitches of C

```

```

//Locate duplicates in the 'maxlist'
//Clear the contents of the keymap

```



```

        keymap.add(entry.getKey());
    }

    keymap2 = new ArrayList();
    for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) { //Clear the contents of keymap2
                                                                    //Create a keymap for 'hashdata'

        keymap2.add(entry.getKey());
    }

    for (i = 1; i < keymap.size() - 2; i++) {
        yes = 0;
        no = 0;
        first = (keymap2.get(1));
        last = (keymap2.get(keymap2.size() - 2));

        //Iterate through the keymap for 'minlist'

        if (maxlist.get(keymap.get(i)).equals(maxlist.get(keymap.get(i + 1)))) { //If adjacent pitch values are the same...
            if ((keymap.get(i).equals( first ))
                && (keymap.get(i+1) != last)){
                unflagged.put(keymap.get(i+1), maxlist.get(keymap.get(i+1)));
                yes++;
            }
            if ((keymap.get(i+1).equals(last))
                && (keymap.get(i) != first)){
                unflagged.put(keymap.get(i),maxlist.get(keymap.get(i)));
                yes++;
            }
            if (keymap.get(i).equals( first )
                && keymap.get(i+1).equals(last)){
                no++;
            }
            // ...do nothing
        }
        if ((yes == 0) && (no == 0)) {
            unflagged.put(keymap.get(i), maxlist.get(keymap.get(i)));
            unflagged.put(keymap.get(i + 1), maxlist.get(keymap.get(i + 1)));
            yes++;
        }
    }
}

```

```

    }

    keymap = new ArrayList();
    // ... then, determine if there are duplicates in the 'minlist'
    // Clear the contents of the keymap
    for (Map.Entry<Integer,Double> entry : minlist.entrySet()) {
        keymap.add(entry.getKey());
    }

    for (i = 1; i < keymap.size() - 2; i++) {
        yes = 0;
        no = 0;
        first = (keymap2.get(1));
        last = (keymap2.get(keymap2.size() - 2));

        if (minlist.get(keymap.get(i)).equals(minlist.get(keymap.get(i + 1)))) { // If adjacent pitch values are the same...
            if ((keymap.get(i).equals( first ))
                && (keymap.get(i+1) != last)) {
                unflagged.put(keymap.get(i+1), minlist.get(keymap.get(i+1)));
                yes++;
            }
            if ((keymap.get(i+1).equals(last))
                && (keymap.get(i) != first)) {
                unflagged.put(keymap.get(i), minlist.get(keymap.get(i)));
                yes++;
            }
            if (keymap.get(i).equals( first )
                && keymap.get(i+1).equals(last)) {
                no++;
            }
            // ... do nothing
        }
        if ((yes == 0) && (no == 0)) {
            unflagged.put(keymap.get(i), minlist.get(keymap.get(i)));
            unflagged.put(keymap.get(i + 1), minlist.get(keymap.get(i + 1)));
            yes++;
        }
    }
}

```

```

//If hashdata is 4 pcs long, proceed to step 12...
//... Else, proceed to Step 13.
//*****
//First, we need to learn what the new hashdata will be if we removed...
//... all the unflagged cps right now
//Add the contents of 'hashdata' to 'test'

test = new LinkedHashMap(hashdata);
keymap = new ArrayList();
keymap2 = new ArrayList();

//Clear the contents of keymap
//Clear the contents of keymap2

for (Map.Entry<Integer,Double> entry : hashdata.entrySet()) { //Create a keymap for 'hashdata'
    keymap.add(entry.getKey());
}

for (Map.Entry<Integer,Double> entry : unflagged.entrySet()) { //Create a keymap for 'unflagged'
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        test.remove(keymap2.get(i));
    }
}

if (test.size() == 4) {

    //If the test hash 'test' is 4 pcs long, then we need to goto step 12.
    //***step 12: If both flagged c-pitches remaining from step 11...
    //...are members of the max-list,
    //... flag any one (and only one) former member of the...
    //... min-list whose flag was removed in step 11; if both c-pitches are..
    //... members of the min-list,
    //... flag any one (and only one) former member...
    //... of the max-list whose flag was removed in step 11
    //Clear the contents of keymap

    keymap = new ArrayList();
    List<Integer> maxkey = new ArrayList();
    List<Integer> minkey = new ArrayList();

    for (Map.Entry<Integer, Double> entry : test.entrySet()) { //Create a keymap for 'test'

```

```

        keymap.add(entry.getKey());
    }

    keymap.remove(keymap.get(0));
    keymap.remove(keymap.get(keymap.size()-1));
    //Remove the first pitch from the keymap
    //Remove the last pitch from the keymap

    for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Create a keymap for the maxlist
        maxkey.add(entry.getKey());
    }

    for (Map.Entry<Integer, Double> entry : minlist.entrySet()) { //Create a keymap for the minlist
        minkey.add(entry.getKey());
    }

    yes = 0;
    for (i = 0; i < maxkey.size(); i++) {
        if (keymap.contains(maxkey.get(i))) {
            yes++;
        }
    }

    //If any of the items in 'maxlist' are a member of test...
    //...(disregarding the first and last cps) ...
    //...then increase the 'yes' counter

    if (yes == 0) {
        maxvalue = (Collections.max(maxlist.values()));
        maxkey = 0;
        for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //...then get the Key of that value from 'maxlist' ...
            if (maxvalue == entry.getValue()) {
                maxkey = (entry.getKey());
            }
        }
        test.put(maxkey, maxvalue);
    }

    // ... add it back to 'test'

    //Now run the same process for the minlist
    //Reset the yes counter
    //If none of the items in min are a members of 'test ...
    //...then increase the 'yes' counter

    yes = 0;
    for (i = 0; i < minkey.size(); i++) {
        if (keymap.contains(minkey.get(i))) {
            yes++;
        }
    }

```

```

    }

    if (yes == 0) {
        minVal = (Collections.min(minlist.values()));
        minKey = 0;
        for (Map.Entry<Integer, Double> entry : minlist.entrySet()) { //...then get the Key of that value from 'minlist '...
            if (minVal == entry.getValue()) {
                minKey = (entry.getKey());
            }
        }
        test.put(minKey, minVal);
    }

    // ... add it back to 'test'

    //End Step 12

}

else {

}

}

}

//***Step 13: Delete all non-flagged c-pitches in C
//Delete cps in 'unflagged' from 'hashdata.'
//Clear contents of keymap
//Clear contents of keymap2

for (Map.Entry<Integer, Double> entry : hashdata.entrySet()) { //Create a keymap for 'hashdata'
    keymap.add(entry.getKey());
}

for (Map.Entry<Integer, Double> entry : unflagged.entrySet()) { //Create a keymap for 'unflagged'
    keymap2.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        hashdata.remove(keymap2.get(i));
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer, Double> entry : maxlist.entrySet()) { //Clear the contents of keymap and create a keymap for maxlist
    keymap.add(entry.getKey());
}

```

```

}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        maxlist.remove(keymap2.get(i));
    }
}

keymap = new ArrayList();
for (Map.Entry<Integer,Double> entry : minlist.entrySet()) {
    keymap.add(entry.getKey());
}

for (i = 0; i < keymap2.size(); i++) {
    if (keymap.contains(keymap2.get(i))) {
        minlist.remove(keymap2.get(i));
    }
}

//If an entry from 'unflagged' exists in 'maxlist,' remove it.

//***Step 14: If N != 0, N is incremented by 1
//***Step 15: if N = 0, N is incremented by 2

//Step 16: Goto Step 6

yes = 0;
endloop = 0;
unflagged.clear();
if (endloop > 0) {
    break;
}

}

System.out.println("Method getSchultzDepth completed successfully");
return N;
}

```

# Appendix H

## Source Code: servlets

```
package net.cooleysekula.CAT;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

/**
```

```

*
* @author ksekula
*/

public class GetMidi extends HttpServlet {

    ArrayList<Integer> list = new ArrayList();
    ArrayList<Integer> listy = new ArrayList();

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        try {
            List<FileItem> items = new ServletFileUpload(new DiskFileItemFactory()).parseRequest(request);
            for (FileItem item : items) {
                if (item.isFormField()) {
                } else {
                    InputStream filecontent = item.getInputStream();

                    listy.clear();
                    list.clear();
                    try {
                        int b = 0;
                        while ((b = filecontent.read()) != -1) {
                            list.add(b);
                            char c = (char) b;
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    } finally {
                        if (filecontent != null) {
                            filecontent.close();
                        }
                    }
                }
            }

            for (int i = 0; i < list.size(); i++) {
                if (list.get(i) == 144) {
                    listy.add(list.get(i + 1));
                }
            }
        }
    }
}

```



```

    }

    RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
    String todo2 = listy.toString().replace(" ", "&nbsp;").replaceAll("&nbsp;", "&nbsp;");
    request.setAttribute("todo2", todo2);
    rd.forward(request, response);
}

} catch (FileUploadException e) {
    throw new ServletException("Cannot parse multipart request.", e);
}

}
}

```

```

package net.cooleysekula.CAT;

//author: Kate Sekula
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Scanner;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Results extends HttpServlet {

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

        String str = request.getParameter("pitches").toString();
        if (str == null) {
            String todo = "You did not enter any pitches";
            RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
            request.setAttribute("todo", todo);
            rd.forward(request, response);
        } else {
            String[] str2 = str.split(" ");
            String str3 = Arrays.toString(str2).replace("[", "").replace("]", "").replace(" ", "");
            ArrayList<Double> data = new ArrayList();

            Scanner scanner = new Scanner(str3);
            while (scanner.hasNext()) {
                data.add(Double.valueOf(scanner.next()));
            }

            ArrayList<Double> schultz = new ArrayList(net.cooleysekula.CAT.Schultz.getSchultzCont(data));

```



# Appendix I

## Source Code: Microsoft Excel TG Segmentation Spreadsheet

```
Sub weighted_pitch_value()
Dim b As Range
Set b = Range("B4", Range("B4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("I4").Activate

For Each i In b
    answer = i * Range("I2").Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next i
Range("I4").Activate
End Sub

Sub adjusted_metronome()
Dim C As Range
Set C = Range("C4", Range("C4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("J4").Activate

For Each i In C
    ActiveCell.Formula = (i.Value * (60 / i.Offset(0, 1).Value))
    ' * 10
    ActiveCell.Offset(1, 0).Select
Next i
Range("J4").Activate
End Sub

Sub weighted_time_value()
Dim j As Range
```

```

Set j = Range("J4", Range("J4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("K4").Activate

For Each i In j
    answer = i * Range("K2").Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next i
Range("K4").Activate
End Sub

Sub weighted_initial_intensity ()
Dim E As Range
Set E = Range("E4", Range("E4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("L4").Activate

For Each i In E
    answer = i * Range("L2").Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next i
Range("L4").Activate
End Sub

Sub weighted_final_intensity ()
Dim F As Range
Set F = Range("F4", Range("F4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("M4").Activate

For Each i In F
    answer = i * Range("L2").Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next i
Range("M4").Activate
End Sub

Sub weighted_timbre_value()
Dim G As Range
Set G = Range("G4", Range("G4").End(xlDown))
Dim i As Variant
Dim answer As Double

Range("N4").Activate

For Each i In G
    answer = i * Range("N2").Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next i

```

```

Range("N4").Activate
End Sub
Sub pitch_interval ()
Dim i As Range
Set i = Range("I4", Range("I4").End(xlDown))
Dim answer As Double

Range("P4").Activate

For Each Cell In i
    answer = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("P4").Activate
End Sub
Sub start_time_interval ()
Dim K As Range
Set K = Range("K4", Range("K4").End(xlDown))
Dim answer As Double

Range("Q4").Activate

For Each Cell In K
    answer = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("Q4").Activate
End Sub
Sub mean_intensity()
Dim L As Range
Set L = Range("L4", Range("L4").End(xlDown))
Dim answer As Double

Range("R4").Activate

For Each Cell In L
    answer = (Cell.Value + Cell.Offset(0, 1).Value) / 2
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("R4").Activate
End Sub
Sub intensity_interval ()
Dim r As Range
Set r = Range("R4", Range("R4").End(xlDown))
Dim answer As Double

Range("S4").Activate

For Each Cell In r
    answer = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell

```

```

Range("S4").Activate
End Sub
Sub timbre_interval()
Dim N As Range
Set N = Range("N4", Range("N4").End(xlDown))
Dim answer As Double

Range("T4").Activate

For Each Cell In N
    answer = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("T4").Activate
End Sub
Sub clang_mean_distance()
Dim P As Range
Set P = Range("P4", Range("P4").End(xlDown))
Dim answer As Double

Range("U4").Activate

For Each Cell In P
    answer = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 3).Value + Cell.Offset(0, 4).Value
    answer = Round(answer, 3)
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("U4").Activate
End Sub
Sub clang_pitch_BI()
Dim P As Range
Set P = Range("P4", Range("P4").End(xlDown))

Range("W4").Activate

For Each Cell In P
    ActiveCell.Formula = Cell.Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("W4").Activate
End Sub
Sub clang_time_BI()
Dim Q As Range
Set Q = Range("Q4", Range("Q4").End(xlDown))

Range("X4").Activate

For Each Cell In Q
    ActiveCell.Formula = Cell.Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("X4").Activate
End Sub
Sub clang_intensity_BI()

```

```

Dim M As Range
Set M = Range("M4", Range("M4").End(xlDown))
Dim answer As Double

Range("Y4").Activate

For Each Cell In M
    answer = Abs(Cell.Value - Cell.Offset(1, -1))
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("Y4").Activate
End Sub

Sub clang_timbre_BI()
Dim t As Range
Set t = Range("T4", Range("T4").End(xlDown))

Range("Z4").Activate

For Each Cell In t
    ActiveCell.Formula = Cell.Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("Z4").Activate
End Sub

Sub clang_boundary_distance()
Dim w As Range
Set w = Range("W4", Range("W4").End(xlDown))

Range("AA4").Activate

For Each Cell In w
    answer = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
    ActiveCell.Formula = answer
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("AA4").Activate
End Sub

Sub clang_termination()
Sheets("Clang_Determination").Select
Dim U As Range
Set U = Range("U5", Range("U5").End(xlDown))
Dim MyCount2 As Integer
MyCount2 = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Range("V4:" & Cells(MyCount2, "V").Address).ClearContents
Range("W4:" & Cells(MyCount2, "W").Address).ClearContents
Dim r As Double
Dim s As Double
Dim t As Double

Range("U5").Activate

For Each Cell In U
    r = Cell.Value

```



```

s = Cell.Offset(1, 0).Value
t = Cell.Offset(-1, 0).Value
    If (r > s) = True And (r > t) = True Then
        ActiveCell.Offset(0, 1).Formula = "X"
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim i As Integer
i = 1

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then
        ActiveCell.Offset(0, 1).Formula = i
        i = i + 1
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(0, 1).Formula = i
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

    If IsEmpty(ActiveCell.Value) Then
        i = i - 1
        Sheets("Sequence_Determination").Select
        Dim A As Range
        Set A = Range("A2:" & Cells(i, "A").Address)
        Dim j As Integer
        j = 1
        Range("A2").Activate
        For Each Cell In A
            ActiveCell.Formula = j
            j = j + 1
            ActiveCell.Offset(1, 0).Select
        Next Cell
    End If
Sheets("Clang_Determination").Select
Range("W4").Activate
End Sub
Sub sequence_mean_pitch()
Sheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim i As Integer
i = 1
Dim x As Integer

```

```

x = 0
Dim answer As Double
Dim sum As Double
sum = 0
Dim j As Integer
j = 2

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -13).Value
        x = x + 1
        answer = (sum / x)
        Worksheets("Sequence_Determination").Range(Cells(j, "B").Address) = answer

        i = i + 1
        j = j + 1
        sum = 0
        x = 0
        ActiveCell.Offset(1, 0).Select
    Else
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -13).Value
        x = x + 1
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

'answer = (sum / x)
' Worksheets("Sequence_Determination").Range(Cells(j, "B").Address) = answer

Sheets("Sequence_Determination").Select
Range("B2").Activate

End Sub

Sub sequence_start_time()
Sheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim i As Integer
i = 1
Dim j As Integer
j = 2

Range("V4").Activate
Worksheets("Sequence_Determination").Range(Cells(j, "C").Address) = ActiveCell.Offset(0, -11).Value
j = j + 1

For Each Cell In v
    If ActiveCell.Value = "X" Then
        Worksheets("Sequence_Determination").Range(Cells(j, "C").Address) = ActiveCell.Offset(1, -11).Value
        ActiveCell.Offset(1, 0).Select
        j = j + 1
    Else

```

```

        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
Sheets("Sequence_Determination").Select
Range("C2").Activate
End Sub
Sub sequence_mean_intensity_value()
Sheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim i As Integer
i = 1
Dim x As Integer
x = 0
Dim answer As Double
Dim sum As Double
sum = 0
Dim j As Integer
j = 2

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -4).Value
        x = x + 1
        answer = (sum / x)
        Worksheets("Sequence_Determination").Range(Cells(j, "D").Address) = answer
        i = i + 1
        j = j + 1
        sum = 0
        x = 0
        ActiveCell.Offset(1, 0).Select
    Else
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -4).Value
        x = x + 1
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
'answer = (sum / x)
'Worksheets("Sequence_Determination").Range(Cells(j, "D").Address) = answer
Sheets("Sequence_Determination").Select
Range("D2").Activate
End Sub
Sub sequence_mean_timbre_value()
Sheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim i As Integer
i = 1

```

```

Dim x As Integer
x = 0
Dim answer As Double
Dim sum As Double
sum = 0
Dim j As Integer
j = 2

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -8).Value
        x = x + 1
        answer = (sum / x)
        Worksheets("Sequence_Determination").Range(Cells(j, "E").Address) = answer
        i = i + 1
        j = j + 1
        sum = 0
        x = 0
        ActiveCell.Offset(1, 0).Select
    Else
        'ActiveCell.Offset(0, 1).Formula = i
        sum = sum + Cell.Offset(0, -8).Value
        x = x + 1
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
' answer = (sum / x)
'Worksheets("Sequence_Determination").Range(Cells(j, "E").Address) = answer
Sheets("Sequence_Determination").Select
Range("E2").Select

End Sub

Sub sequence_mean_pitch_interval()
Sheets("Sequence_Determination").Select
Dim b As Range
Set b = Range("B2", Range("B2").End(xlDown))

Range("G2").Activate

For Each Cell In b
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("G2").Activate

End Sub

Sub sequence_time_mean_interval()
Sheets("Sequence_Determination").Select
Dim C As Range
Set C = Range("C2", Range("C2").End(xlDown))

Range("H2").Activate

For Each Cell In C

```

```

        ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
        ActiveCell.Offset(1, 0).Select
    Next Cell
    Range("H2").Activate
End Sub

Sub sequence_intensity_mean_interval()
    Sheets("Sequence.Determination").Select
    Dim D As Range
    Set D = Range("D2", Range("D2").End(xlDown))

    Range("I2").Activate

    For Each Cell In D
        ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
        ActiveCell.Offset(1, 0).Select
    Next Cell
    Range("I2").Activate
End Sub

Sub sequence_timbre_mean_interval()
    Sheets("Sequence.Determination").Select
    Dim E As Range
    Set E = Range("E2", Range("E2").End(xlDown))

    Range("J2").Activate

    For Each Cell In E
        ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
        ActiveCell.Offset(1, 0).Select
    Next Cell
    Range("J2").Activate
End Sub

Sub sequence_mean_interval()
    Sheets("Sequence.Determination").Select
    Dim G As Range
    Set G = Range("G2", Range("G2").End(xlDown))

    Range("K2").Activate

    For Each Cell In G
        ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
        ActiveCell.Offset(1, 0).Select
    Next Cell
    Range("K2").Activate
End Sub

Sub sequence_pitch_BI()
    Worksheets("Clang.Determination").Select
    Dim v As Range
    Dim MyCount As Integer
    MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
    Set v = Range("V4:" & Cells(MyCount, "V").Address)
    Dim j As Integer
    j = 2
    Dim answer As Double
    Dim answer2 As Double

    Range("V4").Activate

```

```

For Each Cell In v
    If ActiveCell.Value = "X" Then
        Worksheets("Sequence_Determination").Range(Cells(j, "M").Address) = Abs(Cell.Offset(1, -13).Value - Cell.Offset(0,
-13).Value)
        ActiveCell.Offset(1, 0).Select
        j = j + 1
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
Worksheets("Sequence_Determination").Range(Cells(j, "M").Address) = ActiveCell.Offset(-1, -13).Value
Worksheets("Sequence_Determination").Select
Range("M2").Activate
End Sub
Sub sequence_time_BI()
Worksheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim j As Integer
j = 2
Dim answer As Double

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then
        Worksheets("Sequence_Determination").Range(Cells(j, "N").Address) = Abs(Cell.Offset(1, -11).Value - Cell.Offset(0,
-11).Value)
        ActiveCell.Offset(1, 0).Select
        j = j + 1
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
Worksheets("Sequence_Determination").Range(Cells(j, "N").Address) = Abs(ActiveCell.Offset(1, -11).Value - ActiveCell.Offset
(0, -11).Value)
Worksheets("Sequence_Determination").Select
Range("N2").Activate
End Sub
Sub sequence_intensity_BI()
Worksheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim j As Integer
j = 2
Dim answer As Double

Range("V4").Activate

For Each Cell In v
    If ActiveCell.Value = "X" Then

```

```

Worksheets("Sequence_Determination").Range(Cells(j, "O").Address) = Abs(Cell.Offset(0, -9).Value - Cell.Offset(1,
-10).Value)
ActiveCell.Offset(1, 0).Select
j = j + 1
Else
ActiveCell.Offset(1, 0).Select
End If
Next Cell
Worksheets("Sequence_Determination").Range(Cells(j, "O").Address) = ActiveCell.Offset(-1, -13).Value
Worksheets("Sequence_Determination").Select
Range("O2").Activate
End Sub
Sub sequence_timbre_BI()
Worksheets("Clang_Determination").Select
Dim v As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("U4", Range("U4").End(xlDown))) + 3
Set v = Range("V4:" & Cells(MyCount, "V").Address)
Dim j As Integer
j = 2
Dim answer As Double

Range("V4").Activate

For Each Cell In v
If ActiveCell.Value = "X" Then
Worksheets("Sequence_Determination").Range(Cells(j, "P").Address) = Abs(Cell.Offset(1, -8).Value - Cell.Offset(0,
-8).Value)
ActiveCell.Offset(1, 0).Select
j = j + 1
Else
ActiveCell.Offset(1, 0).Select
End If
Next Cell
Worksheets("Sequence_Determination").Range(Cells(j, "P").Address) = ActiveCell.Offset(-1, -13).Value
Worksheets("Sequence_Determination").Select
Range("P2").Activate
End Sub
Sub sequence_BI_sum()
Sheets("Sequence_Determination").Select
Dim M As Range
Set M = Range("M2", Range("M2").End(xlDown))

Range("Q2").Activate

For Each Cell In M
ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
ActiveCell.Offset(1, 0).Select
Next Cell
Range("Q2").Activate
End Sub
Sub sequence_BI_sum_divided()
Sheets("Sequence_Determination").Select
Dim Q As Range
Set Q = Range("Q2", Range("Q2").End(xlDown))

```

```

Range("R2").Activate

For Each Cell In Q
    ActiveCell.Formula = Cell.Value / 2
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("R2").Activate
End Sub
Sub sequence_disjunction()
Sheets("Sequence_Determination").Select
Dim r As Range
Set r = Range("R2", Range("R2").End(xlDown))

Range("S2").Activate

For Each Cell In r
    ActiveCell.Formula = Round((Cell.Value + Cell.Offset(0, -7).Value), 3)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("S2").Activate
End Sub
Sub sequence_termination()
Sheets("Sequence_Determination").Select
Dim s As Range
Set s = Range("S3", Range("S3").End(xlDown))
MyCountStart = Application.CountA(Range("S2", Range("S2").End(xlDown)))
Range("T2:" & Cells(MyCountStart, "T").Address).ClearContents

Dim MyCount2 As Integer
MyCount2 = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Range("T2:" & Cells(MyCount2, "T").Address).ClearContents
Range("U2:" & Cells(MyCount2, "U").Address).ClearContents

Range("T2").Activate
ActiveCell.Formula = ""
ActiveCell.Offset(1, 0).Select

For Each Cell In s
    If Cell.Value > Cell.Offset(1, 0).Value And Cell.Value > Cell.Offset(-1, 0).Value Then
        ActiveCell.Formula = "X"
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim i As Integer
i = 1

Range("T2").Activate

For Each Cell In t

```



```

        If ActiveCell.Value = "X" Then
            ActiveCell.Offset(0, 1).Formula = i
            i = i + 1
            ActiveCell.Offset(1, 0).Select
        Else
            ActiveCell.Offset(0, 1).Formula = i
            ActiveCell.Offset(1, 0).Select
        End If
    Next Cell

    If IsEmpty(ActiveCell.Value) Then
        'i = i + 1
        Sheets("Segment_Determination").Select
        Dim A As Range
        Set A = Range("A2:" & Cells(i, "A").Address)
        Dim j As Integer
        j = 1
        Range("A2").Activate
        For Each Cell In A
            ActiveCell.Formula = j
            j = j + 1
            ActiveCell.Offset(1, 0).Select
        Next Cell
    End If
    Sheets("Sequence_Determination").Select
    Range("U2").Select
End Sub

Sub segment_mean_pitch()
    Sheets("Sequence_Determination").Select

    Dim w As Range
    Dim MyCount As Integer
    MyCount = Application.CountA(Worksheets("Clang_Determination").Range("W:W"))
    MyCount = MyCount + 3
    Set w = Worksheets("Clang_Determination").Range("W4:" & Cells(MyCount, "W").Address)
    Dim E As Variant
    Dim F As Variant

    Dim A As Range
    Dim MyCount1 As Integer
    MyCount1 = Application.CountA(Worksheets("Segment_Determination").Range("A:A"))
    MyCount1 = MyCount1 + 2
    Set A = Worksheets("Segment_Determination").Range("A2:" & Cells(MyCount1, "A").Address)

    Dim U As Range
    Dim MyCount2 As Integer
    MyCount2 = Application.CountA(Worksheets("Sequence_Determination").Range("U:U"))
    Set U = Range("U2:" & Cells(MyCount2, "U").Address)
    Dim G As Variant

    Dim i As Integer
    i = 1
    Dim b As Integer
    Dim sum As Double
    Dim x As Double
    Dim z As Integer
    Dim y As Integer

```

```

y = 1
z = 2

Range("U2").Select

For Each Cell In U
    If Cell.Value = i Then
        b = Cell.Offset(0, -20).Value
        For Each E In w
            If E.Value = b Then
                sum = sum + E.Offset(0, -14).Value
                x = x + 1

            End If
        Next E
    Else
        answer = (sum / x)
        Worksheets("Segment.Determination").Range("B" & z).Value = answer

        z = z + 1
        x = 0
        sum = 0
        y = b
        y = y + 1

        For Each F In w
            If F.Value = y Then
                sum = sum + F.Offset(0, -14).Value
                x = x + 1
            End If
        Next F
        i = i + 1
    End If
    ActiveCell.Offset(1, 0).Select
Next Cell
answer = (sum / x)
Worksheets("Segment.Determination").Range("B" & z).Value = answer
Sheets("Segment.Determination").Select
End Sub

Sub segment_mean_intensity()
Sheets("Sequence.Determination").Select

Dim w As Range
Dim MyCount As Integer
MyCount = Application.CountA(Worksheets("Clang.Determination").Range("W:W"))
Set w = Worksheets("Clang.Determination").Range("W4:" & Cells(MyCount, "W").Address)
Dim E As Variant
Dim F As Variant

Dim A As Range
Dim MyCount1 As Integer
MyCount1 = Application.CountA(Worksheets("Segment.Determination").Range("A:A"))
MyCount1 = MyCount1 + 2
Set A = Worksheets("Segment.Determination").Range("A2:" & Cells(MyCount1, "A").Address)

```

```

Dim U As Range
Dim MyCount2 As Integer
MyCount2 = Application.CountA(Worksheets("Sequence_Determination").Range("U:U"))
Set U = Range("U2:" & Cells(MyCount2, "U").Address)
Dim G As Variant

Dim i As Integer
i = 1
Dim b As Integer
Dim sum As Double
Dim x As Double
Dim z As Integer
Dim y As Integer
y = 1
z = 2

Range("U2").Select

For Each Cell In U
    If Cell.Value = i Then
        b = Cell.Offset(0, -20).Value
        For Each E In w
            If E.Value = b Then
                sum = sum + E.Offset(0, -5).Value
                x = x + 1

                End If
            Next E
        Else
            answer = (sum / x)
            Worksheets("Segment_Determination").Range("D" & z).Value = answer
            z = z + 1
            x = 0
            sum = 0
            y = b
            y = y + 1

            For Each F In w
                If F.Value = y Then
                    sum = sum + F.Offset(0, -5).Value
                    x = x + 1

                    End If
                Next F
            i = i + 1
        End If
        ActiveCell.Offset(1, 0).Select
    Next Cell
    answer = (sum / x)
    Worksheets("Segment_Determination").Range("D" & z).Value = answer
    Sheets("Segment_Determination").Select
End Sub
Sub segment_mean_timbre()
    Sheets("Sequence_Determination").Select

Dim w As Range

```

```

Dim MyCount As Integer
MyCount = Application.CountA(Worksheets("Clang_Determination").Range("W:W"))
Set w = Worksheets("Clang_Determination").Range("W4:" & Cells(MyCount, "W").Address)
Dim E As Variant
Dim F As Variant

Dim A As Range
Dim MyCount1 As Integer
MyCount1 = Application.CountA(Worksheets("Segment_Determination").Range("A:A"))
MyCount1 = MyCount1 + 2
Set A = Worksheets("Segment_Determination").Range("A2:" & Cells(MyCount1, "A").Address)

Dim U As Range
Dim MyCount2 As Integer
MyCount2 = Application.CountA(Worksheets("Sequence_Determination").Range("U:U"))
Set U = Range("U2:" & Cells(MyCount2, "U").Address)
Dim G As Variant

Dim i As Integer
i = 1
Dim b As Integer
Dim sum As Double
Dim x As Double
Dim z As Integer
Dim y As Integer
y = 1
z = 2

Range("U2").Select

For Each Cell In U
    If Cell.Value = i Then
        b = Cell.Offset(0, -20).Value
        For Each E In w
            If E.Value = b Then
                sum = sum + E.Offset(0, -9).Value
                x = x + 1

                End If
            Next E
        Else
            answer = (sum / x)
            Worksheets("Segment_Determination").Range("E" & z).Value = answer
            z = z + 1
            x = 0
            sum = 0
            y = b
            y = y + 1

            For Each F In w
                If F.Value = y Then
                    sum = sum + F.Offset(0, -9).Value
                    x = x + 1

                    End If
                Next F
            i = i + 1

```

```

End If

ActiveCell.Offset(1, 0).Select
Next Cell
answer = (sum / x)
Worksheets("Segment.Determination").Range("E" & z).Value = answer
Sheets("Segment.Determination").Select
End Sub

Sub segment_start_time()
Sheets("Sequence.Determination").Select
Dim U As Range
Set U = Range("U2", Range("U2").End(xlDown))
Dim j As Integer
j = 2
Dim i As Integer
i = 1
Range("U2").Activate

For Each Cell In U
    If ActiveCell.Value = i Then
        Worksheets("Segment.Determination").Range("C" & j).Value = ActiveCell.Offset(0, -18).Value
        ActiveCell.Offset(1, 0).Select
        j = j + 1
        i = i + 1
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
Worksheets("Segment.Determination").Select
Range("C2").Activate

End Sub

Sub segment_mean_pitch_interval()
Sheets("Segment.Determination").Select
Dim b As Range
Set b = Range("B2", Range("B2").End(xlDown))

Range("G2").Activate

For Each Cell In b
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("G2").Activate
End Sub

Sub segment_time_mean_interval()
Sheets("Segment.Determination").Select
Dim C As Range
Set C = Range("C2", Range("C2").End(xlDown))

Range("H2").Activate

For Each Cell In C
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("H2").Activate

```

```

End Sub

Sub segment_intensity_mean_interval()
Sheets("Segment_Determination").Select
Dim D As Range
Set D = Range("D2", Range("D2").End(xlDown))

Range("I2").Activate

For Each Cell In D
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("I2").Activate
End Sub

Sub segment_timbre_mean_interval()
Sheets("Segment_Determination").Select
Dim E As Range
Set E = Range("E2", Range("E2").End(xlDown))

Range("J2").Activate

For Each Cell In E
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("J2").Activate
End Sub

Sub segment_mean_interval()
Sheets("Segment_Determination").Select
Dim G As Range
Set G = Range("G2", Range("G2").End(xlDown))

Range("K2").Activate

For Each Cell In G
    ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("K2").Activate
End Sub

Sub segment_pitch_BI()
Sheets("Sequence_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 2
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4

```

```

Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Sequence.Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                U = Worksheets("Clang.Determination").Cells(r, "I").Value
                v = Worksheets("Clang.Determination").Cells(s, "I").Value
                answer = Abs(U - v)
                Worksheets("Segment.Determination").Range("M" & z).Value = answer
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
            r = Cell.Row
            s = r - 1
            U = Worksheets("Clang.Determination").Cells(r, "I").Value
            v = Worksheets("Clang.Determination").Cells(s, "I").Value
            answer = Abs(U - v)
            Worksheets("Segment.Determination").Range("M" & z).Value = answer
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Segment.Determination").Select
Range("M2").Activate
End Sub
Sub segment_time_BI()
Sheets("Sequence.Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang.Determination").Range("K4", Worksheets("Clang.Determination").Range("K4").End(xlDown))
Dim i As Variant
Dim answer As Double
Dim z As Integer
z = 2
Dim object2 As Double
Worksheets("Sequence.Determination").Range("T2").Activate

For Each Cell In t
    If Cell.Value = "X" Then

```

```

        object = Cell.Offset(1, -17).Value
    For Each i In K
        If i = object Then
            answer = Abs(i.Value - i.Offset(-1, 0).Value)
            Worksheets("Segment_Determination").Range("N" & z).Value = answer
            z = z + 1
            ActiveCell.Offset(1, 0).Select
        Exit For
    End If
Next i
Else
    ActiveCell.Offset(1, 0).Select
End If
object2 = Cell.Offset(1, -17).Value
Next Cell

    For Each i In K
        If i = object2 Then
            answer = Abs(i.Value - i.Offset(-1, 0).Value)
            Worksheets("Segment_Determination").Range("N" & z).Value = answer
        End If
    Next i
Worksheets("Segment_Determination").Select
Range("N2").Activate
End Sub
Sub segment_intensity_BI()
Sheets("Sequence_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 2
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Sequence_Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then

                r = Cell.Row
                s = r - 1

```



```

        U = Worksheets("Clang_Determination").Cells(r, "L").Value
        v = Worksheets("Clang_Determination").Cells(s, "M").Value
        answer = Abs(U - v)
        Worksheets("Segment_Determination").Range("O" & z).Value = answer
        z = z + 1
        ActiveCell.Offset(1, 0).Select
    End If
    r = Cell.Row
    s = r - 1
    U = Worksheets("Clang_Determination").Cells(r, "L").Value
    v = Worksheets("Clang_Determination").Cells(s, "M").Value
    answer = Abs(U - v)
    Worksheets("Segment_Determination").Range("O" & z).Value = answer
Next Cell

Else
    ActiveCell.Offset(1, 0).Select
End If

Next j
Worksheets("Segment_Determination").Select
Range("P2").Activate
End Sub

Sub segment_timbre_BI()
    Sheets("Sequence_Determination").Select
    Dim t As Range
    Dim MyCount As Integer
    MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
    Set t = Range("T2:" & Cells(MyCount, "T").Address)
    Dim object As Double

    Dim K As Range
    Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
    Dim i As Variant
    Dim answer As Double
    Dim z As Integer
    z = 2
    Dim object2 As Double

    Worksheets("Sequence_Determination").Range("T2").Activate

    For Each Cell In t
        If Cell.Value = "X" Then
            object = Cell.Offset(1, -17).Value
            For Each i In K
                If i = object Then
                    answer = Abs(i.Offset(0, 3).Value - i.Offset(-1, 3).Value)
                    Worksheets("Segment_Determination").Range("P" & z).Value = answer
                    z = z + 1
                    ActiveCell.Offset(1, 0).Select
                Exit For
            End If
            Next i
        Else
            ActiveCell.Offset(1, 0).Select
        End If
        object2 = Cell.Offset(1, -17).Value
    Next Cell

```

```

For Each i In K
    If i = object2 Then
        answer = Abs(i.Offset(0, 3).Value - i.Offset(-1, 3).Value)
        Worksheets("Segment_Determination").Range("P" & z).Value = answer
    End If
Next i
Worksheets("Segment_Determination").Select
Range("P2").Activate
End Sub
Sub segment.BLsum()
Sheets("Segment_Determination").Select
Dim M As Range
Set M = Range("M2", Range("M2").End(xlDown))

Range("Q2").Activate

For Each Cell In M
    ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("Q2").Activate
End Sub
Sub segment.BLdivided()
Sheets("Segment_Determination").Select
Dim Q As Range
Set Q = Range("Q2", Range("Q2").End(xlDown))

Range("R2").Activate

For Each Cell In Q
    ActiveCell.Formula = Cell.Value / 4
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("R2").Activate
End Sub
Sub segment.disjunction()
Sheets("Segment_Determination").Select
Dim r As Range
Set r = Range("R2", Range("R2").End(xlDown))

Range("S2").Activate

For Each Cell In r
    ActiveCell.Formula = Round((Cell.Value + Cell.Offset(0, -7).Value), 3)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("S2").Activate
End Sub
Sub segment.termination()
Sheets("Segment_Determination").Select
Dim s As Range
Set s = Range("S3", Range("S3").End(xlDown))
MyCountStart = Application.CountA(Range("S2", Range("S2").End(xlDown)))
Range("T2" & Cells(MyCountStart, "T").Address).ClearContents

Dim MyCount2 As Integer

```

```

MyCount2 = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Range("T2:" & Cells(MyCount2, "T").Address).ClearContents
Range("U2:" & Cells(MyCount2, "U").Address).ClearContents

Range("T2").Activate
ActiveCell.Formula = ""
ActiveCell.Offset(1, 0).Select

For Each Cell In s
    If Cell.Value > Cell.Offset(1, 0).Value And Cell.Value > Cell.Offset(-1, 0).Value Then
        ActiveCell.Formula = "X"
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim i As Integer
i = 1

Range("T2").Activate

For Each Cell In t
    If ActiveCell.Value = "X" Then
        ActiveCell.Offset(0, 1).Formula = i
        i = i + 1
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(0, 1).Formula = i
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

If IsEmpty(ActiveCell.Value) Then
    i = i + 1
    Sheets("Section_Determination").Select
    Dim A As Range
    Set A = Range("A2:" & Cells(i, "A").Address)
    Dim j As Integer
    j = 1
    Range("A2").Activate
    For Each Cell In A
        ActiveCell.Formula = j
        j = j + 1
        ActiveCell.Offset(1, 0).Select
    Next Cell
    End If
    Sheets("Segment_Determination").Select
    Range("U2").Select
End Sub

Sub section_mean_pitch()
    Sheets("Segment_Determination").Select
    Dim t As Range

```

```

Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang.Determination").Range("K4", Worksheets("Clang.Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer

Worksheets("Segment.Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(0, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                answer = Application.WorksheetFunction.Average(Worksheets("Clang.Determination").Range(Worksheets("
Clang.Determination").Cells(x, "I").Address, Worksheets("Clang.Determination").Cells(s, "I").Address))
                Worksheets("Section.Determination").Range("B" & z).Value = answer
                x = r
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
            ' r = Cell.Row
            ' s = r - 1
            ' answer = Application.WorksheetFunction.Average(Worksheets("Clang.Determination").Range(Worksheets("
Clang.Determination").Cells(x, "I").Address, Worksheets("Clang.Determination").Cells(s, "I").Address))
            ' Worksheets("Section.Determination").Range("B" & z).Value = answer
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Section.Determination").Select
Range("B2").Activate

End Sub
Sub section_start.time()
Sheets("Segment.Determination").Select
Dim U As Range
Set U = Range("U2", Range("U2").End(xlDown))
Dim j As Integer
j = 2
Dim i As Integer

```

```

i = 1
Range("U2").Activate

For Each Cell In U
    If ActiveCell.Value = i Then
        Worksheets("Section_Determination").Range("C" & j).Value = ActiveCell.Offset(0, -18).Value
        ActiveCell.Offset(1, 0).Select
        j = j + 1
        i = i + 1
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell
Worksheets("Section_Determination").Select
Range("C2").Activate
End Sub

Sub section_mean_intensity()
Sheets("Segment_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer

Worksheets("Segment_Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(0, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                answer = Application.WorksheetFunction.Average(Worksheets("Clang_Determination").Range(Worksheets("Clang_Determination").Cells(x, "R").Address, Worksheets("Clang_Determination").Cells(s, "R").Address))
                Worksheets("Section_Determination").Range("D" & z).Value = answer
                x = r
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
            ' r = Cell.Row
            ' s = r - 1
            ' answer = Application.WorksheetFunction.Average(Worksheets("Clang_Determination").Range(Worksheets("

```

```

        Clang_Determination").Cells(x, "R").Address, Worksheets("Clang_Determination").Cells(s, "R").Address))
        'Worksheets("Section_Determination").Range("D" & z).Value = answer
    Next Cell
Else
    ActiveCell.Offset(1, 0).Select
End If
Next j
Worksheets("Section_Determination").Select
Range("D2").Activate
End Sub
Sub section_mean_timbre()
Sheets("Segment_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer

Worksheets("Segment_Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(0, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                answer = Application.WorksheetFunction.Average(Worksheets("Clang_Determination").Range(Worksheets("
Clang_Determination").Cells(x, "N").Address, Worksheets("Clang_Determination").Cells(s, "N").Address))
                Worksheets("Section_Determination").Range("E" & z).Value = answer
                x = r
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
            ' r = Cell.Row
            ' s = r - 1
            ' answer = Application.WorksheetFunction.Average(Worksheets("Clang_Determination").Range(Worksheets("
Clang_Determination").Cells(x, "N").Address, Worksheets("Clang_Determination").Cells(s, "N").Address))
            'Worksheets("Section_Determination").Range("E" & z).Value = answer
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If

```

```

Next j
Worksheets("Section_Determination").Select
Range("E2").Activate
End Sub

Sub section_mean_pitch_interval()
Sheets("Section_Determination").Select
Dim b As Range
Set b = Range("B2", Range("B2").End(xlDown))

Range("G2").Activate

For Each Cell In b
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("G2").Activate
End Sub

Sub section_time_mean_interval()
Sheets("Section_Determination").Select
Dim C As Range
Set C = Range("C2", Range("C2").End(xlDown))

Range("H2").Activate

For Each Cell In C
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("H2").Activate
End Sub

Sub section_intensity_mean_interval()
Sheets("Section_Determination").Select
Dim D As Range
Set D = Range("D2", Range("D2").End(xlDown))

Range("I2").Activate

For Each Cell In D
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("I2").Activate
End Sub

Sub section_timbre_mean_interval()
Sheets("Section_Determination").Select
Dim E As Range
Set E = Range("E2", Range("E2").End(xlDown))

Range("J2").Activate

For Each Cell In E
    ActiveCell.Formula = Abs(Cell.Value - Cell.Offset(1, 0).Value)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("J2").Activate
End Sub

```

```

Sub section_mean_interval()
Sheets("Section.Determination").Select
Dim G As Range
Set G = Range("G2", Range("G2").End(xlDown))

Range("K2").Activate

For Each Cell In G
    ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("K2").Activate
End Sub

Sub section_pitch_BI()
Sheets("Segment.Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang.Determination").Range("K4", Worksheets("Clang.Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Segment.Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                U = Worksheets("Clang.Determination").Cells(r, "I").Value
                v = Worksheets("Clang.Determination").Cells(s, "I").Value
                answer = Abs(U - v)
                Worksheets("Section.Determination").Range("M" & z).Value = answer
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Section.Determination").Select

```



```

Range("M2").Activate
End Sub
Sub section_time_BI()
Sheets("Segment_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 4
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang_Determination").Range("K4", Worksheets("Clang_Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Segment_Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then

                r = Cell.Row
                s = r - 1
                U = Worksheets("Clang_Determination").Cells(r, "K").Value
                v = Worksheets("Clang_Determination").Cells(s, "K").Value
                answer = Abs(U - v)
                Worksheets("Section_Determination").Range("N" & z).Value = answer
                z = z + 1
                ActiveCell.Offset(1, 0).Select

            End If
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Section_Determination").Select
Range("N2").Activate
End Sub

Sub section_intensity_BI()
Sheets("Segment_Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)

```

```

Dim object As Double

Dim K As Range
Set K = Worksheets("Clang.Determination").Range("K4", Worksheets("Clang.Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant
Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Segment.Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                U = Worksheets("Clang.Determination").Cells(r, "L").Value
                v = Worksheets("Clang.Determination").Cells(s, "M").Value
                answer = Abs(U - v)
                Worksheets("Section.Determination").Range("O" & z).Value = answer
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Section.Determination").Select
Range("O2").Activate
End Sub
Sub section_timbre_BI()
Sheets("Segment.Determination").Select
Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S4", Range("S4").End(xlDown))) + 3
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim object As Double

Dim K As Range
Set K = Worksheets("Clang.Determination").Range("K4", Worksheets("Clang.Determination").Range("K4").End(xlDown))
Dim j As Variant
Dim z As Integer
z = 2
Dim x As Integer
x = 4
Dim y As Variant

```

```

Dim answer As Double
Dim r As Integer
Dim s As Integer
Dim U As Double
Dim v As Double
Worksheets("Segment.Determination").Range("T2").Activate

For Each j In t
    If j.Value = "X" Then
        y = j.Offset(1, -17).Value
        For Each Cell In K
            If Cell = y Then
                r = Cell.Row
                s = r - 1
                U = Worksheets("Clang.Determination").Cells(r, "N").Value
                v = Worksheets("Clang.Determination").Cells(s, "N").Value
                answer = Abs(U - v)
                Worksheets("Section.Determination").Range("P" & z).Value = answer
                z = z + 1
                ActiveCell.Offset(1, 0).Select
            End If
        Next Cell
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next j
Worksheets("Section.Determination").Select
Range("P2").Activate
End Sub
Sub section_BI_sum()
Sheets("Section.Determination").Select
Dim M As Range
Set M = Range("M2", Range("M2").End(xlDown))

Range("Q2").Activate

For Each Cell In M
    ActiveCell.Formula = Cell.Value + Cell.Offset(0, 1).Value + Cell.Offset(0, 2).Value + Cell.Offset(0, 3).Value
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("Q2").Activate
End Sub
Sub section_BI_sum_divided()
Sheets("Section.Determination").Select
Dim Q As Range
Set Q = Range("Q2", Range("Q2").End(xlDown))

Range("R2").Activate

For Each Cell In Q
    ActiveCell.Formula = Cell.Value / 8
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("R2").Activate
End Sub
Sub section_disjunction()

```

```

Sheets("Section.Determination").Select
Dim r As Range
Set r = Range("R2", Range("R2").End(xlDown))

Range("S2").Activate

For Each Cell In r
    ActiveCell.Formula = Round((Cell.Value + Cell.Offset(0, -7).Value), 3)
    ActiveCell.Offset(1, 0).Select
Next Cell
Range("S2").Activate
End Sub

Sub section_termination()
Sheets("Section.Determination").Select
Dim s As Range
Set s = Range("S3", Range("S3").End(xlDown))
MyCountStart = Application.CountA(Range("S2", Range("S2").End(xlDown)))
Range("T2:" & Cells(MyCountStart, "T").Address).ClearContents

Dim MyCount2 As Integer
MyCount2 = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Range("T2:" & Cells(MyCount2, "T").Address).ClearContents
Range("U2:" & Cells(MyCount2, "U").Address).ClearContents

Range("T2").Activate
ActiveCell.Formula = ""
ActiveCell.Offset(1, 0).Select

For Each Cell In s
    If Cell.Value > Cell.Offset(1, 0).Value And Cell.Value > Cell.Offset(-1, 0).Value Then
        ActiveCell.Formula = "X"
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

Dim t As Range
Dim MyCount As Integer
MyCount = Application.CountA(Range("S2", Range("S2").End(xlDown))) + 1
Set t = Range("T2:" & Cells(MyCount, "T").Address)
Dim i As Integer
i = 1

Range("T2").Activate

For Each Cell In t
    If ActiveCell.Value = "X" Then
        ActiveCell.Offset(0, 1).Formula = i
        i = i + 1
        ActiveCell.Offset(1, 0).Select
    Else
        ActiveCell.Offset(0, 1).Formula = i
        ActiveCell.Offset(1, 0).Select
    End If
Next Cell

```

```

If IsEmpty(ActiveCell.Value) Then
Sheets("Piece_Determination").Select
Dim A As Range
Set A = Range("A2:" & Cells(i, "A").Address)
Dim j As Integer
j = 1
Range("A2").Activate
For Each Cell In A
    ActiveCell.Formula = j
    j = j + 1
    ActiveCell.Offset(1, 0).Select
Next Cell
End If
Sheets("Section_Determination").Select
Range("U2").Select
End Sub

```

# Bibliography

- Adams, Charles R. 1976. "Melodic Contour Typology." *Ethnomusicology* 20 (2): 179–215.
- Artaud, Pierre-Yves, and Catherine Dale. 1994. "Aspects of the Flute in the Twentieth Century." *Contemporary Music Review* 8 (2): 131–216.
- Babbitt, Milton. 1961. "Set Structure as a Compositional Determinant." *Journal of Music Theory* 5 (1): 72–94.
- Beard, R. Daniel. 2003. "Contour Modeling by Multiple Linear Regression of the Nineteen Piano Sonatas by Mozart." PhD diss., The Florida State University.
- Bor, Mustafa. 2009. "Contour Reduction Algorithms: A Theory of Pitch and Duration Hierarchies for Post-Tonal Music." PhD diss., University of British Columbia.
- Bosi, Marina, and Richard E Goldberg. 2003. *Introduction to Digital Audio Coding and Standards*. Springer.
- Burt, Peter. 2006. *The Music of Toru Takemitsu*. Cambridge University Press.
- Clarke, Eric F. 1986. "Theory, Analysis and the Psychology of Music: A Critical Evaluation of Lerdahl, F. and Jackendoff, R., A Generative Theory of Tonal Music." *Psychology of Music* 14 (1): 3–16.
- Cope, David. 2008. *Hidden Structure: Music Analysis Using Computers*. Vol. 23. Madison: AR Editions.
- Cuthbert, Michael Scott, and Christopher Ariza. 2010. "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." In *International Society for Music Information Retrieval Conference*, 637–42.
- Donnelly, Colleen Elaine. 1994. *Linguistics for Writers*. SUNY Press.

- Farrell, Joyce. 2013. *Java Programming*. Cengage Learning.
- Forte, Allen. 1973. *The Structure of Atonal Music*. Yale University Press.
- Freedman, Alan. accessed July 15 2014. *Computer Desktop Encyclopedia*. Mobile Application Software. Version June 2014. <https://www.computerlanguage.com/>.
- Friedmann, Michael L. 1985. "A Methodology for the Discussion of Contour: Its Application to Schoenberg's Music." *Journal of Music Theory* 29 (2): 223–48.
- Genos. 2013. "Genos@ONLINE." December. <http://genosmus.com/english/>.
- Gosling, James, and Henry McGilton. 1995. *The Java Language Environment: A White Paper*. Sun Microsystems.
- Graybill, Roger, Raphael Atlas, and Michael Cherlin. 1994. "'Prolongation, Gesture and Musical Motion'." Edited by Raphael Atlas and Michael Cherlin. *Musical Transformation and Musical Intuition: Eleven Essays in Honor of David Lewin*: 199–224.
- Handelman, Eliot, and Andie Sigler. 2011. "A New Technique for Melodic Analysis." *McGill University*.
- Hanninen, Dora A. 2001. "Orientations, Criteria, Segments: A General Theory of Segmentation for Music Analysis." *Journal of Music Theory* 45 (2): 345–433.
- . 2012. *A Theory of Music Analysis: On Segmentation and Associative Organization*. Vol. 92. Eastman Studies in Music. University Rochester Press.
- Hasty, Christopher. 1981. "Segmentation and Process in Post-Tonal Music." *Music Theory Spectrum* 3:54–73.
- Hergenhahn, Baldwin Ross. 2013. *An Introduction to the History of Psychology*. Cengage Learning.
- Hoare, Charles Antony Richard. 1969. "An Axiomatic Basis for Computer Programming." *Communications of the ACM* 12 (10): 576–80.
- Horstmann, Cay S., and Gary Cornell. 2013. *Core Java Programming*. 9th. Vol. I - Fundamentals. Kindle Edition. New Jersey: Prentice Hall.

- Jendrock, Eric, Ian Evans, Devika Gollapudi, Kim Haase, Ricardo Cervera-Navarro, Chinmayee Srivathsa, and William Markito. 2014. *Java EE 7 Tutorial*. Vol. 2. Pearson Education.
- Koozin, Timothy. 1989. "The Solo Piano Works of Toru Takemitsu: A Linear/Set-theoretical Analysis." PhD diss., University of Cincinnati.
- . 2002. "Traversing Distances: Pitch Organization, Gesture and Imagery in the Late Works of Toru Takemitsu." *Contemporary Music Review* 21 (4): 17–34.
- Krenek, Ernst. 1940. *Studies in Counterpoint Based on the Twelve-Tone Technique*. G. Schirmer / Co., Inc.
- Lee, Chung-Lin. 2010. "Analysis and Interpretation of Kazuo Fukushima's Solo Flute Music." PhD diss., University of Washington.
- Lefkowitz, David S., and Kristin Taavola. 2000. "Segmentation in Music: Generalizing a Piece Sensitive Approach." *Journal of Music Theory* 44 (1): 171–229.
- Lerdahl, Fred. 2004. *Tonal Pitch Space*. Oxford University Press, USA.
- Lerdahl, Fred, and Ray Jackendoff. 1983. *A Generative Theory of Tonal Music*. The MIT Press.
- Lewin, David. 1982. "A Formal Theory of Generalized Tonal Functions." *Journal of Music Theory* 26 (1): 23–60.
- . 1987. *Generalized Musical Intervals and Transformations*. Oxford University Press.
- . 1993. *Musical Form and Transformation: Four Analytic Essays*. Oxford University Press.
- Lilypond Development Team. Accessed 14 July 2014. *Lilypond: Essay on Automated Music Engraving*. Version 2.18.2. [lilypond.org/doc/v2.18/Documentation/essay.pdf](http://lilypond.org/doc/v2.18/Documentation/essay.pdf).
- Marvin, Elizabeth West, and Paul A Laprade. 1987. "Relating Musical Contours: Extensions of a Theory for Contour." *Journal of Music Theory* 31 (2): 225–67.



- Mazzola, Guerino, Grard Milmeister, Karim Morsy, and Florian Thalmann. 2008. "Functors for Music: The Rubato Composer System." In *Transdisciplinary Digital Art: Sound, Vision and the New Screen*, 238–54. Springer.
- Meredith, David, Kjell Lemstrom, and Geraint A Wiggins. 2002. "Algorithms for Discovering Repeated Patterns in Multidimensional Representations of Polyphonic Music." *Journal of New Music Research* 31 (4): 321–45.
- Messiaen, Olivier. 1956. *The Technique of My Musical Language*. Translated by John Satterfield. Vol. 1 and 2. (John Satterfield, trans.) Leduc.
- Milmeister, Grard. 2009. *The Rubato Composer Music Software*. Springer.
- Morris, Robert. 1987. *Composition with Pitch-Classes: A Theory of Compositional Design*. New Haven: Yale University Press.
- . 1991. *Class Notes for Atonal Music Theory*. Frog Peak Music.
- . 1993. "New Directions in the Theory and Analysis of Musical Contour." *Music Theory Spectrum*: 205–28.
- Nakov, Svetlin. 2013. *Fundamentals of Computer Programming with C#: The Bulgarian C# Book*. Svetlin Nakov.
- Nattiez, Jean-Jacques, and Anna Barry. 1982. "Varse's Density 21.5: A Study in Semiological Analysis." *Music Analysis* 1 (3): 243–340.
- Ohriner, Mitchell S. 2012. "Grouping Hierarchy and Trajectories of Pacing in Performances of Chopin's Mazurkas." *Music Theory Online* 18, no. 1 (April).
- Polansky, Larry. 1979. "A Hierarchical Analysis of Ruggles' *Portals*." In *Proceedings of the 1978 International Computer Music Conference, Vol. 2*, edited by Curtis Roads, 790–852. Evanston: Northwestern University Press.
- Polansky, Larry, and Richard Bassein. 1992. "Possible and Impossible Melody: Some Formal Aspects of Contour." *Journal of Music Theory* 36 (2): 259–84.
- Pople, Anthony. 2002. "Getting Started with the Tonalities Music Analysis Software." Ver. 6.25/03 (March).

- Quinn, Ian. 1997. "Fuzzy Extensions to the Theory of Contour." *Music Theory Spectrum*: 232–63.
- Robinson, Elizabeth A. 2011. "Voice, Itinerant, and Air: A Performance and Analytical Guide to the Solo Flute works of Toru Takemitsu." PhD diss., Ball State University.
- Ruwet, Nicolas, and Mark Everist. 1987. "Methods of Analysis in Musicology." *Music Analysis* 6 (1/2): 3–36.
- Sampaio, Marcos S., and Pedro Krger. 2009. "Goiaba: A Software to Process Musical Contours." In *Proceedings of the 12th Brazilian Symposium on Computer Music*, 203–6.
- Schenker, Heinrich. 1935. *Der Freie Satz. Neue Musikalische Theorien und Phantasien*. N. Mees, Trans.). Lige: Margada.
- Schoenberg, Arnold. 1967. *Fundamentals of Musical Composition*. Faber / Faber.
- Schler, Nico Stephan. 2000. "Methods of Computer-Assisted Music Analysis: History, Classification, and Evaluation." PhD diss., Michigan State University.
- Schultz, Robert D. 2008. "Melodic Contour and Nonretrogradable Structure in the Birdsong of Olivier Messiaen." *Music Theory Spectrum* 30 (1): 89–137.
- . 2009. "A Diachronic-Transformational Theory of Musical Contour Relations." PhD diss., University of Washington.
- Spinellis, Diomidis. 2006. *Code Quality: the Open Source Perspective*. Adobe Press.
- Straus, Joseph N. 2003. "Uniformity, Balance, and Smoothness in Atonal Voice Leading." *Music Theory Spectrum* 25 (2): 305–52.
- Syrstad, Tracy, and Bill Jelen. 2004. *VBA and Macros for Microsoft Excel*. Kindle edition. Pearson Education.
- Takemitsu, Toru, Yoshiko Kakudo, Glenn Glasow, and Seiji Ozawa. 1995. *Confronting Silence: Selected Writings*. Scarecrow Press.
- Taube, Rick, and Andrew Burnson. 2013. "Harmonia@ONLINE." December. <http://camil.music.illinois.edu/software/harmonia/>.

- Temperley, David. 2009. "A Unified Probabilistic Model for Polyphonic Music Analysis." *Journal of New Music Research* 38 (1): 3–18.
- Temperley, David, and Daniel Sleater. 2013. "The Melisma Music Analyzer Version 2.0." December. <http://theory.esm.rochester.edu/temperley/melisma2/>.
- Tenney, James, and Larry Polansky. 1980. "Temporal Gestalt Perception in Music." *Journal of Music Theory* 24 (2): 205–41.
- Toop, Richard. 2004. "Expanding Horizons: the International Avant-Garde, 1962–1975." In *The Cambridge History of Twentieth-Century Music*, 453–77. Cambridge University Press.
- Uno, Y., and R. Huebscher. 1994. "Temporal-Gestalt Segmentation- Extensions for Compound Monophonic and Simple Polyphonic Musical Contexts: Appl. to Works by Cage, Boulez, Babbitt, Xenakis and Ligeti." In *Proceedings of the International Computer Music Conference*, 7–10. International Computer Music Association.
- Unwalla, Mike. 2006. "LaTeX: an Introduction." *Communicator*: 33. [www.techscribe.co.uk/ta/latex-introduction.pdf](http://www.techscribe.co.uk/ta/latex-introduction.pdf).
- Watanabe, Mihoko. 2008. "The Essence of Mei." *Flutist Quarterly* 33, no. 3 (Spring).
- Wu, Yi-Cheng (Daniel). 2012. "Reflection and Representation: A Unitary Theory of Voice-Leading and Musical Contour in 20th-Century Atonal and Serial Contrapuntal Music." PhD diss., State University of New York at Buffalo.
- Young, Robert W. 1939. "Terminology for Logarithmic Frequency Units." *The Journal of the Acoustical Society of America* 11 (1): 134–39.