The *Echoi* of Modern Greek Church Chant in Written and Oral Transmission: A Computational Model and Its Cognitive Implications

by

Panayotis Mavromatis

Submitted in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy

Supervised by

Professor Matthew G. Brown
Music Theory Department
Eastman School of Music

University of Rochester
Rochester, New York

2005
UMI Number: 3169609

Copyright 2005 by
Mavromatis, Panayotis

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3169609
Copyright 2005 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Curriculum Vitae

Panayotis Mavromatis was born in Athens, Greece on July 23, 1965. He attended Cambridge University, England (BA in Mathematics, 1987; Diploma of Advanced Study in Mathematics, 1988; MA in Mathematics, 1992) and Boston University (MA in Physics, 1997). He began graduate studies in Music Theory at the Eastman School of Music, University of Rochester in the Fall of 1995. He pursued his research under the direction of Professor Matthew Brown.
Acknowledgements

I would like to thank the members of my committee Matthew Brown, Davy Temperley, and Gabriela Ilnitchi for their help and encouragement during the preparation of this dissertation.
Abstract

This dissertation presents a mathematical technique for exploring the melodic structure of the eight *Echoi* of Greek church chant, as understood and employed by chanters in present-day liturgy. Being able to recall or improvise a melody within the framework of an *Echos* forms an essential part of a chanter’s training; this skill, however, relies to a great extent on implicit, internalized knowledge that is passed on from teacher to student by example, without explicit appeal to rules. Such knowledge can be inferred from consistent patterns in both written and oral versions of chant melodies. This dissertation proposes a formal grammar that tries to make explicit the knowledge underlying the execution of the *Echoi*. This grammar can be obtained by analyzing quantitatively a representative written corpus of Greek chant. Given a chant text and an *Echos*, the grammar enables one to calculate algorithmically all the possible settings of that text within the *Echos*, rating the generated melodies according to how typical they are. This dissertation focuses on two main issues: on the one hand, it explores the process of model-building, presenting the mathematical framework and addressing the methodological issues that arise; on the other hand, it examines the structure of the resulting grammar, with a view to understanding its possible cognitive implications. In particular, the dissertation shows that the structure of our grammar resonates with modern psychological theories of sequence representation, and thus promises to shed light on the cognition of melody.
Contents

1 Introduction
   1.1 The Problem ........................................... 2
   1.2 The Solution ........................................... 6

2 The Echoi in Theory and Practice .......... 15
   2.1 The Heirmologikoi Echoi in Chrysanthine Theory . . . . . 18
   2.2 The Heirmologikoi Echoi in Present Practice ............. 25

3 Finite State Models of the Echoi ........... 31
   3.1 Fundamentals of Finite State Machines .................. 32
   3.2 Choosing among Competing Models ..................... 41

4 Bayesian Model Selection .................... 49
   4.1 Fundamentals of Probability Theory ................... 50
   4.2 Bayes’s Law and Model Selection ..................... 54
   4.3 Minimum Description Length in Model Selection .......... 61

5 Analysis of the Model ....................... 70
CONTENTS

5.1 Training the Hidden Markov Model .......................... 71
5.2 The Structure of the Hidden Markov Model ................. 74
5.3 Melodic Chunks and Decision Points ......................... 77
5.4 Evaluating the Model ........................................ 82

6 Implications and Future Directions .......................... 86
6.1 Oral Transmission and Western Chant Studies ............... 86
6.2 Towards a Cognitive Model of Melody ....................... 88
6.3 The Question of Model Selection ............................ 91
6.4 The Value of Computational Modeling ....................... 94

A The HMM Implementation ...................................... 101

B Complete Code Listings ....................................... 107

C Complete Sample Model of Echos 1 .......................... 152
List of Tables

2.1  The eight-week cycle of the Octoechos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

C.1  Sample chant used to obtain the model of Appendix C  . . . . . . . 161
# List of Figures

1.1 Effect of Word Stress on Shaping a Melody .................... 8
1.2 Effect of Word Stress on Melodic Adaptation ................. 9
2.1 Scales, Principal, and Cadential Tones for the *Heirmologikoi Echoi* ........ 20
2.2 Intonation Formulas for the *Heirmologikoi Echoi* ............ 22
2.3 Examples of *Stichoi* in the *Heirmologikoi Echoi* .......... 24
2.4 Representative Example of *Heirmologikos Echos* 1 ............ 25
2.5 Representative Example of *Heirmologikos Echos* 2 ............ 25
2.6 Representative Example of *Heirmologikos Echos* 3 ............ 26
2.7 Representative Example of *Heirmologikos Echos* 4 ............ 26
2.8 Representative Example of *Heirmologikos Echos* 5 ............ 26
2.9 Representative Example of *Heirmologikos Echos* 6 ............ 27
2.10 Representative Example of *Heirmologikos Echos* 7 .......... 27
2.11 Representative Example of *Heirmologikos Echos* 8 .......... 27
3.1 A sample representing a family of phrases in *Echos* 1 .......... 33
3.2 A FSM model of the opening formula of Figure 3.1 ............ 34
# List of Figures

3.3 A FSM model of the cadential formula of Figure 3.1  
3.4 The FSM model of Figure 3.3 in interval-based representation  
3.5 A Finite State Transducer for the cadential formula of Figure 3.1  
3.6 A poor FSM model of the opening formula of Figure 3.1  
3.7 The FSM of Figure 3.5 reproduced.  
3.8 An alternative to the model of Figure 3.7 that fails to generalize.  
3.9 Adding abstraction by appropriate state merging.  
3.10 A set of state merges leading to the model of Figure 3.7.  
3.11 An alternative to the model of Figure 3.7 that over-generalizes.  
4.1 Data points on a plane with no identifiable pattern.  
4.2 Data points on a plane that fall on a straight line.  
5.1 Structure of an HMM that models *Echos* 1  
5.2 Subgraph corresponding to the opening formula OF.01  
5.3 Subgraph corresponding to the cadential formula CF.D01  
5.4 Family of all possible realizations of a given text phrase, illustrating chunks and decision points  
5.5 Another family showing a similar organization  
C.1 Structure of an HMM modeling *Echos* 1  
C.2 Subgraph corresponding to opening formula OF.01.  
C.3 Subgraph corresponding to cadential formula CF.G01.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.4</td>
<td>Subgraph corresponding to cadential formula CF_G02.</td>
<td>156</td>
</tr>
<tr>
<td>C.5</td>
<td>Subgraph corresponding to cadential formula CF_G03.</td>
<td>157</td>
</tr>
<tr>
<td>C.6</td>
<td>Subgraph corresponding to cadential formula CF_G04.</td>
<td>158</td>
</tr>
<tr>
<td>C.7</td>
<td>Subgraph corresponding to cadential formula CF_D01.</td>
<td>159</td>
</tr>
<tr>
<td>C.8</td>
<td>Subgraph corresponding to phrase body.</td>
<td>160</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This dissertation presents a mathematical technique for exploring the melodic structure of the eight Echoi of Greek church chant, as understood and employed by chanter's in present-day liturgy. The Echoi of Eastern chant corresponds roughly to the mode of Western chant, but is realized in more structured terms than just a scale and final. What gives the Echoi special significance is the role it plays in the chanter’s knowledge of their art. Being able to recall or improvise a melody within the framework of an Echoi forms an essential part of a chanter's training; this skill, however, relies to a great extent on implicit, internalized knowledge that is passed on from teacher to student by example, without explicit appeal to rules. Such knowledge can be inferred from consistent patterns in both written and oral versions of chant melodies. This dissertation proposes a formal grammar that tries to make explicit the knowledge underlying the execution of the Echoi. This grammar can be obtained by analyzing quantitatively a representative written corpus of Greek chant. Given a chant text and an Echoi,
CHAPTER 1. INTRODUCTION

the grammar enables one to calculate algorithmically all the possible settings of that text within the *Echos*, rating the generated melodies according to how typical they are. As it turns out, the structure of our grammar resonates with modern psychological theories of sequence representation, and thus promises to shed light on the cognition of melody.

To put these issues in context, we will describe the problem in more detail in Section 1.1, offering the necessary background in the history and performance practice of Greek chant. In Section 1.2, we will outline the steps towards our solution.

1.1 The Problem

Modern Greek chant draws on a body of liturgical texts that were compiled in the period of the Byzantine Empire (AD 330–1453) and have remained largely fixed for more than a millennium.\(^1\) The music, however, went through several stages of stylistic changes, not only during the Byzantine period, but also in the intervening years of Ottoman rule (1453–c. 1830).\(^2\) The modern repertory took shape in the years 1770–1820, largely under the influence of the distinguished chanter Petros Peloponnesios (c. 1730–1778) and his students.\(^3\) At that time, received

\(^1\)For an overview of the Greek liturgy, see (Ware, 1969). On the history of Byzantine liturgy, see (Taft, 1992).

\(^2\)For a survey of Byzantine music, see (Levy and Troelsgård, 2001; Vellimirovic, 1994). The best short account of Greek chant during the Ottoman years is (Conomos, 1988); see also (Patrnelis, 1973; Dragoumis, 1969).

\(^3\)See the *New Grove* entries on Petros Peloponnesios (Conomos, 2001) and Petros Byzantios (Lingas, 2001); see also (Patrnelis, 1973).
versions of the *automela* (chants that were fixed and served as models for other chants) were compiled alongside with realized versions of the *idiomela* (chants that were not fixed and could therefore be improvised) into the chant book known as the *Anastasimatarion*, the musical realization of the *Octoechos*. Among other things, this compilation encouraged standardization; many melodies that were previously improvised were now replaced by fixed musical texts.

The dissemination of the new repertory was facilitated by the Chrysanthine reform that followed shortly thereafter. In a twenty year period between 1812 and 1832, Archbishop Chrysanthos of Madytos established a new simplified music notation that made music printing practical for the first time. In his classic treatise, *Theoretikon Mega tes Mousikes* (1832), he presented a new theoretical system that was both broader in scope and more pedagogically oriented than its predecessors. This system is still used in Greece with minor modifications. In 1815, Chrysanthos and his collaborators founded the Third Patriarchal School in Constantinople. For many decades, the best teachers of the Greek chant tradi-

---

4 According to Wolfram (2001), “*Stichëra idiomela* have their own individual melodies, whereas the *automela* function as melodic and metrical patterns for generating *stichëra prosomoia* (contrafacta).” In his definition, Chrysanthos of Madytos (1832, pp. 183–4) provides some insight into the compositional process. It is in this latter sense that the terms are used here.

5 On the possible role of Petros in the standardization of melodies and the motivation behind it, see the introduction to (Vallindras, 1998). On the performance of the *idiomela*, see (Chrysanthos of Madytos, 1832, Book 5, Chapter 3).

6 The *Theoretikon* was reprinted in facsimile in recent years, and is still available in Greece as (Chrysanthos of Madytos, 1832). Chrysanthos’s theory of scales received minor revisions by the Patriarchal Commission of 1883 (Patriarchal Commission, 1888); see also (Morgan, 1971). Modern textbooks on the theory of Greek chant include (Margaziotis, 1958; Kakoulidis, 1988; Iliopoulos, 2000).

7 For a history of Chrysanthos’s reform and its reception, see (Morgan, 1971).
tion received their training in that school, spreading Chrysanthos's new method, as well as the style of Petros and his students, to the rest of the Greek church. Since 1820, a continuous series of printed editions of the Anastasimatarion and the Heirmologia (the collection of Heirmoi, an important genre of fixed model melodies) has brought the two books to a wider audience, establishing them as standard references. These books have been fundamental to the training of chanters down to the present day.

The appearance of printed sources, however, did not eliminate the oral tradition. The liturgical service in the modern-day Greek tradition involves complex and ever-changing combinations of proper and ordinary material, making it impractical for chanters to perform exclusively from music notation during service; the melodies are often recalled with the aid of the words alone. This imposes considerable demands on the chanter's memory. Except for the most commonly used automela, the received melodies can vary considerably in performance, and even free improvised renderings of the idiomela are sung to the present day. Besides, even the printed sources that supposedly transmit Petros's original versions differ from each other, sometimes considerably so. These differences indicate that a process of oral transmission had already taken place before individual chant teachers had committed their versions to print.\(^8\)

\(^8\)A comparison of the written sources (Valliandras, 1998; Kyriakidis, 1970; Kallinikos, 1971; Georgiadis, 1974) illustrates this point. The original versions from Petros's school are recorded in pre-Chrysantine notation in a number of sources; see (Hatzigiakoumis, 1980). The earliest transcriptions in Chrysanthine notation are printed in (Ephesios, 1820).
CHAPTER 1. INTRODUCTION

Given this complex interplay between the written and the oral mode, the importance of the Echoi for the current practice of Greek chanting cannot be overestimated. The Echos provides the primary framework for learning and recalling chants; it supplies a context for what is essential and what can be altered in the performance of an existing melody, such as an automelon; it is a necessary background for the composition of new chants; and it offers schemes for improvisation during church service when this is allowed, as in the rendering of the idiomela.

Expert chanters use their knowledge of an Echos to shape a melody, down to a level of considerable detail. This level of detail, however, is not explicitly addressed in the Chrysanthine theory of the Echoi, which characterizes each one in terms of its scale(s), principal tones, cadences, and intonation formulas. Nor can the chanter themselves always explain exactly what they do, except perhaps by illustration. In light of present theories of cognitive psychology, this phenomenon is characteristic of implicit, or internalized knowledge, and is indicative of expert behavior. In this mode, people perform tasks quickly and efficiently, without the need to attend to them in every detail; as a result, they are also unable to analyze their actions and verbalize about them. Within the Greek chant tradition,

---

9Chapter 2 presents in some detail all the relevant portions of existing Greek theory. For more detail, see (Chrysanthos of Madytos, 1832, Book 4), the authoritative source for the theory of the Echoi. See also (Rebours, 1906; Desby, 1974; Giannelos, 1996; Karas, 1985; Mavroeidis, 1999) and the Greek textbooks listed in Footnote 6, p. 3.
10For a discussion and references, see (Anderson, 2000), especially chapters 3, 8, and 9.
this implicit knowledge has been transmitted for decades: students acquire it by carefully imitating the example of their teacher, absorbing the melodic detail “by ear,” and eventually internalizing it themselves. The fact that Greek chant theorists have not sought to further refine their characterizations of the Echoi suggests that this method can be fairly effective.

It is precisely this implicit knowledge that our model intends to capture. Quite apart from its usefulness to Greek chant studies, including pedagogical applications, a cognitive model of the Echoi may shed light on the cognition of melody in general, and may indeed further clarify the possible contribution of music psychology to music theoretic modeling. In this sense, the present project is offered as a case study in model building. The changing nature of the written text among different editions, the significance of the mental processes involved in oral transmission, the lack of preexisting analytical tools to tackle the problem, and the limitations evident in existing theoretical characterizations of the Echoi, combine to make this problem an ideal case study of building a model out of raw data.

1.2 The Solution

Let us now describe how we will actually build our model. The process of model building and the formal grammar itself are both implemented in computer programs. This implementation allows us to develop our model and calculate its
consequences quickly, accurately, and exhaustively. The computer implementation is a central part of this study and will be covered in more detail in the chapters that follow, as well as in Appendices A and B.

The techniques followed in this dissertation are inspired by research in computational linguistics and machine learning.\textsuperscript{11} They allow us to analyze quantitatively a corpus of chants to extract rules governing the behavior of each specific Echos. Even though this method can be applied in principle to both written and oral corpora, we will build our model using a sample of written chants. This strategy will enable us to develop our formalism without having to worry at this stage about issues of field work, data collection and transcription.\textsuperscript{12} But it should be stressed that in principle, the same techniques could apply to an oral corpus available in some possible transcribed form. The chants will be sampled from a modern edition of the Anastasimatarion (Vallindras, 1998). This chant book was chosen because of the aforementioned central role it plays in teaching and transmitting chant. We will hypothesize that all the information contained in the chanter’s internalized knowledge is already present in this suitably chosen written corpus. In Chapter 6, we will show how the validity of this working hypothesis can be tested and, if necessary, how the grammar can be modified in light of oral data.

\textsuperscript{11}For a discussion and references, see for example (Manning and Schütze, 1999), especially Chapters 2, 6, and 9.

\textsuperscript{12}For conceptual background and a discussion of methodology, see (Foley, 1990; Finneghan, 1992; Ong, 2002).
CHAPTER 1. INTRODUCTION

Figure 1.1: Examples of melodies in Echos 1 illustrating the effect of word stress on shaping a melody. Phrases (a) and (b) open two troparia of the Anavathmoi. Translation: (a) When I am afflicted, listen to my distress; (b) To the Holy Spirit honor and glory [belong], as to the Father. Phrase (c) is marked by an asterisk to show that it is ill-formed, a convention used in linguistics.

To give a preliminary and intuitive illustration of the kind of information that may be contained in the written corpus, consider the melodies in Figures 1.1 and 1.2, all of which are examples of the first Echos. As an illustration of how Chrysanthine theory characterizes the Echoi, Echos 1 is built on a diatonic scale with a final of D; it employs G as the principal tone around which the melody moves; G is also the goal of the secondary cadences of the Echos. In addition, Echos 1 is characterized by melodic features not spelled out by Chrysanthine theory, such as the opening and cadential formulae marked in square brackets. In Figure 1.1, phrases (a) and (b) open two troparia (hymn verses) of the Anavathmoi (musical commentaries to the Gradual Psalms, 119–133). Both phrases agree with the theoretical descriptions of Echos 1. However, even though they have the same number of syllables (15), the melody of one fits the other very
Figure 1.2: Examples of melodies in Echos 1 illustrating the effect of word stress on melodic adaptation. Phrases (a) and (b) open two troparia of the Resurrectional Apolytikion. Translation: (a) When the stone was secured by the Jews; (b) When Gabriel greeted you, o Virgin. As in Figure 1.1, the ill-formed phrase (c) is marked by an asterisk.

uncomfortably, as line (c) illustrates. To someone accustomed to the idiom, line (c) goes against the natural stress of the words; the differences between melodies (a) and (b) are the results of a process of accommodation.\textsuperscript{13}

The situation is most dramatically illustrated with Figure 1.2. Phrases (a) and (b) each open one of the two verses of the Resurrectional Apolytikion (dismissal hymn of the vespers) for Echos 1. In strophic forms such as this one, the verses normally follow the same stress pattern and are thus set to practically identical melodies. In this case, however, the opening lines of each verse differ substantially in their stress pattern; in view of this fact, the melody of line (b) needs to alter

\textsuperscript{13}Such pitch-stress relations are not of course limited to Greek chant. The reader can experience the effect of mismatched stress by trying the words “The little star that twinkles” on the tune of “Twinkle twinkle little star” without inserting or deleting notes.
the model of line (a) quite drastically, even though the corresponding lines of
text only differ by one syllable in length (14 and 15 respectively). And while the
hypothetical solution of line (c) follows model (a) more closely than (b) (the only
difference is an inserted note G, marked +), it goes against word stress to such
an extent as to render the text hard to understand.

Our model captures text-melody relations such as the above in the form of
a function that connects selected text-related input variables (linguistic word
stress, syntactic boundaries) to desired output variables (pitch, duration). The
main challenge is that the rules of chant melody are too complex to write down
directly. Instead, we must use an algorithmic method, within a framework com-
puter scientists often refer to as grammatical inference. In addition, we will argue
that our rule system must be probabilistic; this means that, given an input, the
system will make several different predictions as to what the output might be,
and these predictions will be weighed according to their likelihood. This will
in fact be necessary during the intermediate stages of our modeling: while we
attempt to pin down all the relevant variables, the effect of others will remain
uncertain. Probabilistic systems quantify this uncertainty and allow us to ma-
nipulate our rules quantitatively in the process of evaluating and refining our
model. But we will also argue that some residual probability is inevitable, even
for our most refined models: no model is ever final, and there are always factors
influencing choices of text setting that are left unaccounted for (expressive fac-
tors, the chanter’s short-term memory, etc.) The point is that, if our system is to reflect practice, it should allow for several possible grammatical solutions to a given text; moreover, these should be rated according to how likely or typical they may be.

The focus of this dissertation will therefore be two-fold: on the one hand, we will explore the process of model-building, presenting the mathematical framework and addressing the methodological issues that arise; on the other hand, we will examine the structure of the resulting model, with a view to understanding its cognitive implications. Although a comprehensive study of the Greek Echoi is beyond the present scope, Chapter 2 will introduce the eight Echoi of modern Greek chant, as necessary background for the analysis that follows. Section 2.1 will give a brief theoretical exposition based on the work of Chrysanthos of Myd Dentos (1832); we will see how Chrysanthine theory defines the tonal framework of an Echos as a scale structured by a final, and other principal tones. Section 2.2 will then illustrate the theoretical ideas at work through representative examples of chants in each Echos. We also get a first view of those factors not addressed by Chrysanthine theory, such as the formulaic character of the chant melody.

Chapters 3 and 4 will give a systematic and self-contained exposition of our mathematical methods. Chapter 3 will introduce the formalism of Finite State Machines (FSM’s) as an efficient way to capture the rules that govern pitch-stress relations. A FSM represents a grammar graphically as paths in a melodic space,
CHAPTER 1. INTRODUCTION

capturing all grammatical realizations in a compact and visually intuitive way. After examining the relevant factors that contribute to formulaic variation, we will build FSM models of melodic formulas in Echos 1. Section 3.2 will address the question of choosing among competing models, identifying several properties that a successful model must have. These properties sometimes overlap, and sometimes conflict with each other.

Chapter 4 will reformulate the problem of model selection in the quantitative framework of Bayesian inference. This will allow us to implement our model-building in a computer program. Bayesian inference is defined within the framework of probability theory, and Section 4.1 will give a brief and self-contained introduction to the latter. Section 4.2 will introduce Bayesian inference as a way to quantify our belief in a model given the observed corpus of chants. This will motivate us to consider probabilistic versions of FSM’s known as Hidden Markov Models (HMM’s). Section 4.3 will present the Minimum Description Length principle, which allows us to implement the desired model properties discussed in Chapter 3 within the Bayesian framework.

Having spelled out our method, Chapter 5 will present and discuss our results, based on the analysis of a small sample collection of chants in Echos 1. Section 5.1 will present a general description of HMM’s; it will also identify some efficiency improvements in the model selection algorithm, based on the chant’s phrase structure. Next, Section 5.2 will examine the form of the resulting model, showing
how its graph breaks down into subgraphs that correspond to formulaic material and generate the chant's phrase components. Finally, Section 5.3 will investigate the properties of our model's predictions. In particular, it will show that, given an *Echos* and a word stress pattern, the melody can be completely determined by a relatively small number of note choices made at key decision points. Each melodic phrase typically contains two to three such points, which divide it into formulaic chunks. Each chunk is about six to nine notes long, with a melodic contour that highlights word stress.

Chapter 6 will discuss the relation of our work to previous research, as well as implications and future directions. Section 6.1 will show how our research responds to ideas advanced in Western chant studies, particularly in relation to the question of oral transmission. Section 6.2 will then argue that the chunk decomposition of the melody outlined in Section 5.3 is consistent with psychological theories of sequence representation (Miller, 1956; Cowan, 2001); this idea suggests in turn that the purpose of this decomposition may be to reduce processing load in real-time tasks such as recall and improvisation. Our work therefore not only makes explicit the formulaic structure of the Greek *Echoi*, but also indicates how this structure may originate in properties of orally transmitted skill.

We have delegated some implementation details to appendices, in order to preserve the main flow of ideas in Chapters 3–5. Appendix A will provide a brief tour of the code, complete listings of which may be found in Appendix B.
Appendix C will produce the complete model of *Echos* 1 obtained from the chant sample of Section 5.2.
Chapter 2

The System of the Greek *Echoi* in Chrysanthine Theory and in Present Practice

The book of the *Octoechos* contains texts for the ordinary of the principal Greek offices, and has been in use since around the eighth century. Each *Echos* of the *Octoechos* is assigned to a week in a cycle of eight. As shown in Table 2.1, the *Echoi* are organized according to the familiar arrangement of four *kyrioi* (main) and four corresponding *plagioi* (plagal). According to the Byzantine nomenclature, which is still in use, each *kyrios* receives an ordinal number, while each *plagios* is named after its corresponding *kyrios*, with the exception of the third which bears the individual designation *Barys*, or “low.” The cycle of the *Echoi* begins on Easter Sunday of each year, and repeats until Holy Week of the following year.¹

¹Many terms familiar from Western modal theory have never been used in the East and are therefore consistently avoided here. Thus ‘authentic’ (‘authenticus’) is a Carolingian coinage and the Greek *kyrios* is used in its place. Nevertheless, the term *plagios* is part of the Byzantine vocabulary. Names like ‘protus’ and ‘tetrardus’ are Latin adaptations, and sometimes
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Echos 1</th>
<th>Protos</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 2</td>
<td>Echos 2</td>
<td>Deuteros</td>
<td>Second</td>
</tr>
<tr>
<td>Week 3</td>
<td>Echos 3</td>
<td>Tritos</td>
<td>Third</td>
</tr>
<tr>
<td>Week 4</td>
<td>Echos 4</td>
<td>Tetartos</td>
<td>Fourth</td>
</tr>
<tr>
<td>Week 5</td>
<td>Echos 5</td>
<td>Plagios Protos</td>
<td>Plagal First</td>
</tr>
<tr>
<td>Week 6</td>
<td>Echos 6</td>
<td>Plagios Deuterus</td>
<td>Plagal Second</td>
</tr>
<tr>
<td>Week 7</td>
<td>Echos 7</td>
<td>Barys</td>
<td>Low</td>
</tr>
<tr>
<td>Week 8</td>
<td>Echos 8</td>
<td>Plagios Tetartos</td>
<td>Plagal Fourth</td>
</tr>
</tbody>
</table>

Table 2.1: The eight-week cycle of the Octoechos.

A simple regular pattern thus underlies the cycle of the Octoechos. But when it comes to characterizing the Echoi musically, things are not so simple. In this respect, both the Byzantine repertory, throughout its long and evolving history, and its modern Greek descendent, present challenges familiar to the student of Western chant. To the extent that we can tell from the notated sources, each Echos was initially recognized in terms of its melodic behavior, embodied in a set of prototypical melodic families; as in Western music theory, the earliest attempts to understand the musical features of these families associate them with a regular arrangement of finals. And just as in the West, this symmetry does not fit practice very well: the melodic families within each Echos can differ significantly among themselves and can rarely be brought together under a single set of shared tonal
corruptions, of the Greek originals, in this case protos and tetartos respectively. Finally, tribal names like ‘Dorian’ (Gk. ‘Dorios’), though regularly alluded to in Byzantine sources, do not correspond to those established in the West; in any case have completely dropped from modern Greek practice. For a detailed comparative account of Octoechal naming and numbering in East and West, see (Jeffrey, 2001), esp. p. 153. For an outline of the Byzantine modes, see (Strunk, 1977), esp. pp. 3–36. In translating the Greek terms, Strunk follows naming conventions different from mine.
features. At present, the repertory has perhaps reached its most diversified state, reflecting several centuries of changing styles and cross-fertilizations with other Eastern musical cultures. No simple and symmetric theoretical model can do justice to this diversity.

Chrysanthine chant theory was motivated by practical and pedagogical concerns. Earlier we mentioned how its founder, Archbishop Chrysanthos of Madytos, made a lasting impact on Greek church music by revolutionizing the way chan ters were taught and by articulating a new theoretical system for his pedagogical purposes. For that reason, Chrysanthine theory does not try to suppress the irregularities of the repertory and does not care to force any symmetry that doesn’t fit comfortably. Chrysanthine theory goes a long way toward characterizing the Echoi, perhaps more so than any earlier such attempt. It is therefore an appropriate starting point for our modeling. In Section 2.1, we will present those aspects of Chrysanthine theory that are relevant to the present study. In Section 2.2, we will use representative examples for each Echoi to illustrate the theory’s descriptive power; we will also show that the theory does not address certain important factors that determine the melody, like contour, formula, and the role played by word stress in shaping these. As we explained in the Introduction, our goal will be to capture all those determining factors in a precise quantitative model. In Chapter 3, we will take the first steps in that direction.
 CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

2.1 The Heirmologikoi Echoi in Chrysanthine Theory

Within each Echos, Chrysanthos described individual melodic families in terms of a set of tonal features; several of these sometimes widely differing families are needed to cover a single Echos. Four tonal features are assigned to each family: a scale (klimax); a set of principal tones (despozontes phthongoi), which include the Echos’s final (basis) and other tonally stable tones; a set of cadences (katalexeis) of varying strength; and a set of intonations (apechemata), melodic patterns that can be used to introduce each Echos in performance.\(^2\)

Chrysanthos’s theory of the Echoi is also interested in melodic style and process. Three distinct melodic styles are recognized, each with its own characteristic ways of composition and transmission; through these different processes, each style encompasses its own melodic families, often representing the same Echos with different tonal characteristics.\(^3\) The three styles are: the heirmologikon (syllabic, fast to moderate tempo), the sticherarikon (neumatic, moderate to slow tempo), and the papadikon (melismatic, moderate to slow tempo). The heirmologikon style covers a great portion of the chants sung in the present-day

\(^2\)See (Chrysanthos of Madytos, 1832, pp. 132–3). In fact, Chrysanthos does not explicitly evoke the concept of “melodic family,” which I have adopted here for convenience. Instead, for each Echos that contains more than one melodic family, Chrysanthos simply lists all the alternative sets of tonal features. See (Chrysanthos of Madytos, 1832, pp. 142–168).

\(^3\)In addition to melodic process and performance practice, historical factors may be partly responsible for the tonal differences among the three styles; the reasons for those differences falls beyond the scope of the present study.
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

service and represents all aspects of the interplay between written and oral process outlined in the Introduction.⁴ Since the Heirmologikoi Echoi offer a perfectly adequate illustration of our methods given the limited space of the present dissertation, we will focus on them in the rest of this study. Let us now turn to the Chrysanthine characterization of the Heirmologikoi Echoi in terms of scales, principal tones, cadences, and intonations.

The first characteristic, the scale, forms the set of pitches available to each melodic family. The scales for the principal families of each Heirmologikos Echos are given in Figure 2.1, according to the standard way of transcription⁵ into Western notation that is followed nowadays in Greece.⁶ The transcription has its limitations, especially with regard to tuning, since the intervals employed in Greek chant differ from their Western counterparts. The differences are summarized in the caption of Figure 2.1. In addition to its basic scale structure, each Echos employs characteristic chromatic notes that depend on the melodic context and are often understood without being explicitly notated. In Echos 1, for

⁴See p. 4.
⁵This is perhaps the place to say that the Chrysanthine chant notation used in Greece to the present day is neumatic, and descends from the medieval Byzantine notation. In this dissertation, we will present all examples transcribed into Western notation.
⁶One exception is Echos 6, which is normally transcribed to have a final of D instead of G, following Chrysanthous (mostly “fixed-do”) solmization system. The latter assigns the syllables PA VOU GA DI KE ZO NI to notes corresponding to D through C of the Western diatonic scale; each Echos final is then assigned a fixed syllable in that system. My choice of G instead of D as a final for Echos 6 follows the standard convention of transcribing manuscripts from the Byzantine and Ottoman periods, and is meant to make more explicit the relation between present-day Echos 6 and its historical predecessors. The question of transposition will not affect the results of the present study.
Figure 2.1: Scales, principal tones, and cadential tones for the main melodic family of each *Heirmologikos Echos*. Echo 1, 4, 5, 8 follow the basic diatonic Greek tuning, in which the notes E and B are roughly 1/3 of a semitone lower than their Western equal-tempered counterparts. *Echoi* 3 and 7 use Pythagorean tuning. For *Echo* 2, the slashed flats in the key signature represent lowering by approximately 2/3 of a semitone; the notes E, B are lower by 1/3 of a semitone, as in *Echoi* 1, 4, 5, 8. White noteheads represent each *Echos's* principal tones, with a breve representing the final. Cadential tones are labeled as A (*imperfect*), E (*internal perfect*), and T (*final perfect*).
example, the note B is lowered when descending and the note E is typically also lowered in the cadential approach to the final D. In Figure 2.1, such contextual chromaticism is indicated by small accidentals above the affected scale steps; the reader can find examples in the sample chants of Figures 2.4 through 2.11.

The second defining characteristic of a melodic family is its set of principal tones; Figure 2.1 represents these by white noteheads; among them is the final, represented by a breve. The principal tones determine melodic behavior in two related ways: on the one hand, they are the salient pitches around which the melody revolves; on the other, most of them are goals of cadential motion.

The third defining characteristic of a melodic family is its set of cadences; these cadences come in three different types, each carrying a different degree of closure, which Chrysanthine theory explains by analogy with language. Thus, *imperfect* (*ateleis*) cadences correspond to commas or colons; *internal perfect* (*enteleis*) cadences correspond to full stops in the middle of the text; *final perfect* (*telikai*) cadences correspond to the full stop at the end of the verse, or *troparion*. These linguistic analogies, as Chrysanthos notes, are also good guidelines for composing or improvising an *idiomelon*, though they need not be followed rigidly in practice. In Figure 2.1, the goal tones of the three cadence types are marked by the letters A, E, and T respectively. Note that in Chrysanthine theory, cadences are explicitly identified only by their cadential pitch, and not by melodic formulas;

---

7Chrysanthos of Madytos (1832, pp. 183–4)
as we will see, the latter are actually important for a fuller characterization of the cadences. Typical such formulas can be found in the examples of Figures 2.4 through 2.11. This is a characteristic case of the theory stopping short of explicit description whenever transmission by example is deemed adequate.

The fourth and final defining characteristic of a melodic family is the intonation. In its simplest form, an intonation can be a one- or two-pitch pattern that establishes the Echos’s final; in its longest form, it can be a short melodic pattern that outlines the intervallic environment around that final. Examples of the latter type are given for each Echos in Figure 2.2. An intonation is typically sung
by the leader of the choir whenever a new Echos is introduced in performance. As can be seen from these examples, modern intonations do not attempt to convey any characteristic melodic material of the melodic family, such as cadential formulas. They are only intended to help the singer recall the interval pattern contained in the underlying scale.\(^8\)

One of the simplest contexts in which we can see the defining characteristics of the Echoi at work is the recitation of the stichoi, or psalm verses, in each Echos. In the Greek/Byzantine tradition, the stichoi occur as prefixes to certain troparia known as stichera; they are somewhat analogous to the psalm tones of Western chant, but differ from them in both function and melodic form. A typical stichos involves recitation on one of the Echos’s principal tones, followed by an often extensive formula that brings the stichos to a close on an imperfect or internal perfect cadence. In Figure 2.3, some of the simplest possibilities of stichoi are given, one for each Echos. In fact, according to Chrysanthos, a stichos can also function as an intonation, in the sense that it introduces a troparion, while defining the Echos’s underlying tonal space.\(^9\) An Echos’s set of possible stichoi

\(^8\)The intonations (Gk apechemata, echemata, enechemata) are perhaps as old as the Echoi themselves. It is in one of their earliest forms that they became known to the West as they were imported and adapted by the Carolingian theory of the modes. The tunes of the intonations have changed over the centuries, the only constant factor in their practice being the nonsense syllables attached to them (ananéanes for Echos 1, néanes for Echos 2, etc.); even the latter were largely supplanted by Chrysanthos’s solmization syllables since the 19th century. (See Footnote 6, p. 19.) Only the modern intonation tunes will be considered here. For more information on the Byzantine intonations, see (Raasted, 1966; Hugo, 1991). For intonations in the post-Byzantine tradition, see (Chrysanthos of Madytos, 1832, pp. 135–142).

\(^9\)Chrysanthos of Madytos (1832), pp. 133, 136
Figure 2.3: Representative stichoi for each Heirmologikos Echos. Cadence types are marked by E or T, as defined in Figure 2.1. Translation: Now and always, and unto the ages of ages. Amen.
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

Figure 2.4: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 1. In this and the following Figures, cadence types are marked by A, E or T, as defined in Figure 2.1.

Figure 2.5: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 2.

can therefore be viewed as an important complement to the Echos’s theoretical characterization.

2.2 The Heirmologikoi Echoi in Present Practice

Figures 2.4 through 2.11 provide examples representative of each Heirmologikos Echos, taken from the modern Anastasimatiarion (Vallindras, 1998). The examples are transcribed in Western notation, with a quarter note representing the
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

Figure 2.6: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 3.

Figure 2.7: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 4.

Figure 2.8: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 5.
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

Figure 2.9: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 6.

Figure 2.10: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 7.

Figure 2.11: A troparion from the Anavathmoi of the morning office (Orthros), representative of Heirmologikos Echos 8.
basic pulse. The pattern of x's under the text, known by linguists as the metric grid, indicates word stress as follows: two x's for a stressed syllable, one x for an unstressed syllable, and no x under a note that does not carry a change of syllable).

As the reader can verify, the melodies clearly bring out the defining characteristics of each Echos as specified by Chrysanthine theory: the tunes employ the scales of Fig. 2.1, with the prescribed emphasis on the principal tones, and they use one of the many prescribed cadential tones at the ends of phrases. The fact that these features are also brought out in recalled or improvised performances with remarkable consistency, shows that they are part of the chanter's internalized knowledge of the Echos.

However, as can only be glimpsed from this small sample, there are additional distinguishing features for each Echos that are not addressed in Chrysanthine theory. These are as much a part of the chanter's internalized knowledge of the

---

10 We will not provide here the details on how rhythm is conceptualized in the Chrysanthine system (Chrysanthos of Madytos, 1832, pp. 51–93). In summary, an underlying steady pulse is assumed, which we represent by quarter notes in our transcriptions. Chrysanthine notation can express simple multiples and subdivisions of this pulse, here transcribed in their corresponding Western counterparts. In the typical syllabic melody, quarter notes predominate. Longer notes are typically reserved for articulating pauses at the ends of phrases. Shorter durations are also infrequent, and are typically part of written-out ornamental figures.

11 Stress in modern Greek is dynamic. It is defined at the lexical level, each word typically carrying one stressed syllable. Subordinate words such as articles, possessives and prepositions that are pre-pended or appended to a main word typically do not carry their own stress. Sometimes, an appended word may transfer stress to the last syllable of the main word immediately preceding it. For the literature on modern phonological studies of Greek stress, see (Arvaniti, 1992, 2000).
features in turn.

First, each Echos involves characteristic melodic formulas that are especially prominent at phrase endings, and sometimes also at phrase beginnings. Comparison with Figure 2.3 will reinforce this point: all the stichoi endings given there involve cadential formulas that can be used more generally in phrase endings within the Echos. In the selected examples, this is only explicitly seen in the troparia for Echoi 1, 3, 6, 7, 8, but it is true in general. It is also worth noting that there may be more than one cadential formula for each cadential pitch; in Figure 2.4, for example, two different formulas are used for each of the two cadences on the pitch G, both of them marked A (imperfect). So the Chrysanthine classification of cadences through goal pitch alone does not characterize cadence types completely.

Second, there appears to be a clear effect of word stress on the choice of melodic intervals. In all of the above examples, a stressed syllable of text (marked by two x’s in the metrical grid) is almost always preceded by an ascending interval, or followed by a descending one, or both. There are very few exceptions to this pattern, and these mostly happen in the context of formulaic phrase endings or beginnings. This pitch-stress relation often results in intervallic patterns that recur in different diatonic positions in more than one Echos. For instance, the openings of the example troparia for Echoi 1, 4, 5, 6, and with some adjustment.

\[\text{\footnotesize\emph{In the Introduction, we already hinted at the role of word stress in shaping the chant’s melody: cf. Figures 1.1 and 1.2, pp. 8, 9, and the surrounding discussion.}}\]
CHAPTER 2. THE ECHOI IN THEORY AND PRACTICE

8, illustrate a typical phrase opening formula, the four-note stepwise ascent to a principal tone above the final. This opening pattern is shared among several Echoi, even though the exact interval sizes involved are different in each case. Another, somewhat more elaborate pattern is shared by the first six notes of the example troparia in Echoi 2 and 3; this pattern also occurs as the last six notes of the third cadential formula in the example troparion for Echos 1.

Understanding such pitch-stress relations and showing how they interact with the tonal framework of each Echos to determine its formulaic, as well as its non-formulaic material, will be the main goal of our study. After having outlined all the basic ingredients as codified by existing theory, and as observed in current practice, we are ready to attempt a synthesis in our proposed new theoretical framework. The next two chapters will offer a systematic exposition of this framework, before we discuss our results in Chapter 5.
Chapter 3

Modeling the *Echoi* with Finite State Machines

In this chapter we will begin building our model of an *Echo* using the formalism of *Finite State Machines* (FSM's), a framework that is widely used in computer science and linguistics.\(^1\) Section 3.1 will introduce the fundamentals of the formalism, and will illustrate its usefulness by building models of melodic formulas in *Echo* 1. The models built in this section will be based on a rather small test sample; although it will have no claim to completeness, it will be sufficient for the purpose of illustration. Moreover, it is to be understood that a complete model of an *Echo* will involve a combination of several models of individual formulas, as well as of non-formulaic material. A more comprehensive model will be presented and analyzed in Chapter 5.

One of the main issues facing us in this chapter is to decide how we might choose the best model(s) out of several competing ones. Section 3.2 will provide

\(^1\text{See (Hopcroft and Ullman, 1979; Gazdar and Mellish, 1989).}\)
an informal introduction to this important and rather subtle problem that will receive fuller mathematical treatment in Chapter 4.

3.1 Fundamentals of Finite State Machines

In this section, we will build simple models of formulas, based on the test sample shown in Figure 3.1. This sample represents the same phrase family, and belongs to Echos 1. The family as a whole appears to be characterized by the following features:

(i) The body of the phrase, enclosed by the first and third vertical dashed lines, revolves around G, one of the principal tones of the Echos. The motions away from G and back are presumably caused by different levels of word stress, as outlined in Section 2.2.\footnote{See p. 29}

(ii) An (optional) opening formula, marked by the first square bracked above the staff, outlines the pitch sequence D E F G, and serves as an initial ascent to G, starting from the basis D of the Echos.

(iii) A cadential formula, marked by the second square bracked above the staff, outlines the pitch sequence G F E F G, and leads the phrase to an imperfect cadence on G.

\footnote{See p. 29}
Figure 3.1: A sample representing a family of phrases in Echos 1. The phrases are aligned so as to show correspondences in pitch structure, and no rests are implied by the uneven spacing. The cadential and (optional) opening formulas of the family are marked with square brackets above the staff, the letter names indicating the main structural pitches. The x’s under each melody represent the stress pattern of the underlying text.
Both the opening and cadential formulas alter their basic pitch sequence through deletions or duplications, which are presumably also linked to the different stress levels of the underlying text.

A Finite State Machine (FSM) offers a concise representation of insertions and deletions such as the above. In addition, a FSM can be represented graphically, making it possible to visualize the model. A FSM is defined by states, graphically represented by circles; and transitions, graphically represented by arrows. Figure 3.2 shows a FSM with five states, labeled 0 through 4. Each transition in the FSM is labeled by an output symbol—in our case one of the letters D, E, F, G that represent pitches. A FSM is also characterized by an initial state (here State 0) and one or more final states (here State 4). To generate a string of symbols, we simply choose a path connecting the initial state to a final state, following the direction of the arrows; the generated string is the sequence of output symbols in the order they are encountered. In our example, following the path through the states 0, 2, 3, 4 produces the sequence E F G, corresponding
Figure 3.3: A FSM model of the cadential formula of Figure 3.1.

to the opening of Figure 3.1, Example 5. The reader can verify that all instances of the opening formula in Figure 3.1 (Examples 1–7) can be generated from the FSM in Figure 3.2.

The advantage of using a FSM may not be immediately obvious. After all, the melodic pattern just examined can be described in words fairly easily, if somewhat awkwardly, perhaps along the following lines: “one or more D’s, optionally followed by E, F, followed by a G; when the E, F is present, the initial D may be omitted.” But the simplicity and elegance of the FSM representation becomes clear once we consider more complicated melodic patterns, such as the cadential formula of Figure 3.1. The reader may verify that the FSM of Figure 3.3 generates all instances of that formula. Any attempt at an equivalent verbal description of the model will be too awkward to work with.

So far we have introduced FSM’s as merely a convenient shorthand representation for networks of melodic possibilities. However, since our goal is to model
human behavior, we might also be interested in possible interpretations of FSM’s in terms of the process of creating a melody, perhaps in real time. The ingredients of a FSM are states and transitions, and these names suggest an obvious possibility: a FSM transition could represent an event triggering some change; a state could likewise represent a period in time in which no change occurs (at the level of detail assumed by our model). In other words, FSM transitions characterize instants in time, or time points, whereas FSM states characterize durations, or time spans. This interpretation has implications for our choice of output symbols: since the latter are associated with transitions, and hence with time points, it will be more natural to choose the output symbols to be intervals, rather than pitches. Pitches characterize time spans, and are therefore more naturally associated with states. This implies an interval-based representation of the chant melody. Such a representation carries the same amount of information as a pitch-based one, given the context of a scale, and we can always convert one representation to the other if necessary.

Figure 3.4 shows a straightforward modification of the FSM of Figure 3.3 that implements the interval-based representation. In this model, the output symbols are represented by ordered pairs: the first entry in the pair represents the size and direction of the diatonic interval, and the second entry represents the target pitch. Thus, we write (1,F) for ‘step up to F’, and (-2,E) for ‘move down by minor third to E’. The second entry thus keeps track of the interval’s position
Figure 3.4: The FSM model of Figure 3.3 in interval-based representation. The choice of pitch for the initial state is arbitrary.

in the scale. In addition, each state carries a definite pitch, as reflected in the new state labels. Naturally, there is a consistency constraint on the position of an output symbol: its target pitch must match that of its transition's target state. This seemingly redundant (interval, target pitch) representation ensures, e.g., that a generated melody can reach closure on a desired goal pitch, which would be very hard to enforce in an interval-only representation. This may seem complicated at first, but as we will see, the resulting model is much easier to interpret.\footnote{In fact, an interval-based representation turns out to be more consistent with Chrysanthine notation, which involves neumes representing diatonic intervals, interspersed with absolute pitch signs \textit{(martyria)} that provide consistency checks. Our FSM output symbols correspond to Chrysanthine neumes, and each FSM state corresponds to a potential \textit{martyria}.}

The FSM's we have built so far model pitch patterns alone. How can we also model patterns of pitch-stress relations, which are so crucial in shaping the melody? A particular type of FSM, known as a \textit{Finite State Transducer} (FST), offers an elegant solution to this problem. Instead of assigning simple outputs
CHAPTER 3. FINITE STATE MODELS OF THE ECHOI

Figure 3.5: A Finite State Transducer for the cadential formula of Figure 3.1, modeling pitch-stress relations.

to its transitions, a FST assigns input-output pairs. When traversed, a FST can thus generate a mapping from its input to its output variables. We will use the notation

\[(\text{input} \rightarrow \text{output})\]

to represent an input/output pair, where ‘input’ and ‘output’ may stand for several variables, separated by commas. In our case, the ‘output’ part will include all the variables that characterize the melody, namely the (interval, target pitch) pair. The ‘input’ part will include all the text-related variables that contribute to the melody’s shape. Under our assumptions, this will be the stress level alone, with values S2, S1, S0 representing ‘stressed syllable’, ‘unstressed syllable’, and ‘no syllable change’ respectively. We write

\[(S1 \rightarrow -1, F)\]
Figure 3.6: A poor FSM model of the opening formula of Figure 3.1; cf Figure 3.2. for ‘an unstressed syllable that causes the melody to step down to an F’.\(^4\) In traversing the FST, one follows a path that matches the stress pattern of the given input text. The corresponding outputs collected will generate the associated melody. Figure 3.5 shows a FST model of the pitch-stress relations that shape the cadential formula of Figure 3.1. The reader is invited to check that this model can generate all the variants of the cadential formula contained in the Figure.

In this section we considered a series of progressively more elaborate models for melody and pitch-stress relations. At each step, we minimally required that the model generates all instances in our given sample. But is this requirement sufficient by itself to determine the model? And in any case, is it enough to guarantee that the model will be useful? A simple example will suffice to show that the answer is ‘no’ in both cases. Consider for simplicity the model of Figure 3.2 that generates the opening formula for the phrase family. Clearly the model of Figure 3.6 generates all opening formulas in our sample. But it also generates

\(^4\)Note that, since stress is a property of a time point rather than a time span, an interval-based representation allows a more natural input/output pairing than does a pitch-based one.
examples such as D F E D E G, that we perhaps do not want to include, given the sample that we have seen so far. Moreover, its underlying rule ‘D followed by any number of D, E, or F’s, ending on a G’, does not appear particularly insightful, in that it does not seem to point to a relevant underlying mechanism. We would not expect that the free scrambling of the three notes D, E, F could be part of a mental model of a melody in this style.\footnote{5}

The question is not as simple as it may at first seem: how can we choose among rival models, if all we can see is the given sample and not the underlying mechanism that generated it? Chapter 4 will develop a quantitative framework to address this problem, based on probability theory. Simply stated the answer will be: we cannot know for sure, but under certain assumptions, we can calculate a probability for each candidate model that tells us how trustworthy the model is, given the observed sample. But before we lay out a more formal treatment, it will be useful to develop some intuition for some of the issues that come into play when choosing among competing models.
3.2 Choosing among Competing Models

Figure 3.7 reproduces the FSM of Figure 3.5, for the sake of comparison with the other models of this section. In Section 3.1, we presented this FSM as a model for the cadential formula of Figure 3.1. We did not address how good or useful that model may be, but simply noted that it succeeds in generating all instances of the formula recorded in the sample. Clearly this is a minimal condition on our model, which we will refer to as the completeness requirement. How can we articulate other factors that may be influencing our choice of model? To answer this question, we must consider models that meet the completeness requirement, but that can be shown to be problematic on other grounds.

\footnote{Strictly speaking, the model of Figure 3.2 can also generate some unlikely solutions—such as a melody that begins with 37 D's in a row—due to the transition loop attached to state 1. This problem can be addressed by assigning probabilities to the FSM transitions, which will be done in Chapter 4. For the model of Figure 3.2, probabilities can be chosen in the process of fitting the model to the data, so that the “bad” solutions become very unlikely. Due to its simplified structure, the model of Figure 3.6 will make the bad solutions as likely as the good ones, since a single transition loop generates both.}
Figure 3.8: An alternative to the model of Figure 3.7 that meets the completeness requirement but that fails to generalize. The numbers emanating from the initial state refer to the numbering of the phrases in Figure 3.1.

Let us first note that the sample of Figure 3.1 includes eight phrases, but only five different instances of the cadential formula, since Examples 1, 4, 5, and 8 all share the same instance. We can construct an FSM that generates these five instances and only these, if we proceed in the following manner: Pick one initial and one final state. For each distinct instance of the cadential formula, create a path from the initial to the final state that generates that instance, and that does not share any states with any other path. The resulting FSM, consisting of as many disjoint paths as there are instances, is shown in Figure 3.8.

By construction, the FSM of Figure 3.8 satisfies the completeness requirement. But the fact that it does not generate any instances other than already observed ones may seem odd, to say the least. After all, we expect a musical model to capture a style, not just a few concrete instances in that style. In other words, the model fails to meet what we will refer to as the generalization requirement.
CHAPTER 3. FINITE STATE MODELS OF THE ECHOI

But the problem goes deeper than that; even if the five instances in question were the only possible ones—which is not untypical in this style—a mere listing of the possibilities can hardly constitute an adequate model. Such a model lacks abstraction, and thus fails to provide relations among observations that may help simplify our understanding of them. We will refer to this missing property as the abstraction requirement. Could we begin to address these problems through some suitable modification of the model?

We may begin by observing that the lower three paths in the model all end with the melodic pattern F, G, G, represented by distinct states in each path, except for the final state, which is shared. The pattern appears each time in a similar context of word stress and pitch (see Figure 3.1), which may invite us to consider the three F states as functionally equivalent to each other, and similarly for the three G states not already shared by the paths. We can represent this equivalence by creating a new FSM in which separate but equivalent states merge into a single one, with their attached transitions appropriately redirected in and out of that merged state. Performing the merges suggested by the shared pattern F, G, G in Figure 3.8 leads to the FSM shown in Figure 3.9.

Identifying and merging states that appear in the same or similar contexts is a simple mechanism for introducing abstraction: the merged states correspond to abstract situations, and through them the model generates several concrete instances. But note that the model of Figure 3.9 hasn’t achieved generalization yet,
since it still produces just the five observed instances of the modeled formula. In
other words, even though abstraction and generalization are clearly related, they
could still be considered as separate requirements. This is because a FSM's ca-
pacity for abstraction is reflected in the choice and meaning of its states, whereas
its capacity for generalization depends on how it is interconnected. Nevertheless,
if we continue the state merging process that we started here, we will eventually
reach a model that does achieve generalization.

In fact, the process of state merging can turn the model of Figure 3.8 into
that of Figure 3.7; the required merges are shown in Figure 3.10, and the reader
is invited to verify that they produce the intended result. In Figure 3.10, the
states to be merged are connected by thick lines, and carry the same labels as
the resulting merged states in Figure 3.7. The reader is also invited to verify that
the model of Figure 3.7 does achieve generalization, in that it generates at least
one previously unseen instance of the cadential formula that is being modeled.
Figure 3.10: A set of state merges that can be applied to the model of Figure 3.8 to produce that of Figure 3.7. States to be merged are connected by thick lines, and carry the same label as the resulting states in Figure 3.7.

In light of their origin in Figure 3.10, the states in Figure 3.7 have a natural musical interpretation: each merged state could be considered to represent a melodic function that is characterized by its pitch as well as its melodic context. Thus, state F:2 represents an optional note F, subordinate to its more stable surrounding G and E, very much like the passing tone of Western theory. By contrast, F:5 and F:6 represent notes necessary to the formula—in the sense that either one or the other must be present—and act as a preparation to the cadential G:8. On a finer level of detail, F:5 and F:6 represent different functions, since the former is unaccented, whereas the latter is accented and requires the “anticipation” note G:7. A representation of an Echos’s pitch material in terms of a FSM contains therefore more information than the static representation in terms of a scale, even when the latter is structured by principal and secondary

---

6Here accented and unaccented simply refer to the stress of the underlying text, and not to musical meter.
tones.

The state merging technique pursued here originates in the work of Stolcke and Omohundro (1993, 1994) on Hidden Markov Models, an idea that we will discuss in Chapter 5. We have adopted this technique as the basis for constructing our model from data. Given any set of chants, we can always start with a FSM of separate paths, each path representing a different chant, like the model of Figure 3.8. Such a model satisfies the completeness requirement, and so does any model obtained from it by merging states. We can therefore perform appropriate state merges on our starting model, in the spirit of Figure 3.10, until we have achieved the desired level of abstraction and generalization.

We have not completely solved the problem of choosing a model. Rather, we have translated it into a new framework; it now amounts to answering the following two questions:

1. How do we choose which states to merge?

2. How do we know when we need to stop merging?

In answer to the first question, we have programmed a computer to systematically consider all the possibilities.\(^7\) The second question cannot be fully addressed without the quantitative framework of Chapter 4. But let us first discuss qualitatively the issues involved.

\(^7\)For more details on the choice of state merging, see Section 5.1.
Consider the FSM of Figure 3.11. It is not hard to see that we can obtain this model from that of Figure 3.7 by performing additional state merges, namely merging E:3 with E:4, and F:2 with F:5 and F:6. The model therefore still generates the desired instances of the formula, plus additional ones. But we may rightly suspect that at this point we have taken the process too far: the model now generates practically any sequence of E’s and F’s between the initial and final G’s, a situation which we already encountered in Figure 3.6, p. 39. Moreover, in merging together states F:2, F:5, and F:6, for example, we lose all the nuance contained in their functional differentiation as discussed earlier. In other words, in addition to over-generalization, the model suffers from excessive and inappropriate abstraction. Clearly generalization and abstraction are properties that must be enjoyed in moderation.

In fact generalization both below and above the desired level causes the model to lose in predictive power: in the former case, the model may fail to predict future observations; in the latter it runs the risk of creating instances that will never be observed. Likewise, abstraction that is either below or above the desired
CHAPTER 3. FINITE STATE MODELS OF THE ECHOI

level may lead to models that are not easily interpretable. For example, we would not have been able to interpret states F:2, F:5, and F:6 in terms of melodic function, if these states had not previously merged together all the appropriate states from Figure 3.8. Neither would we have achieved our interpretation if we had gone on to merge the three states together, as in Figure 3.11.

In this section we have considered three requirements that a good model must meet, namely completeness, generalization, and abstraction. We have seen how the last two are related, and how they lead to predictive power and interpretability. We have carried out this discussion in a qualitative framework. Moreover, we have introduced state merging as a search strategy for achieving the optimal model. The next chapter will put all these ideas in a quantitative framework, allowing us to implement the search in a computer algorithm. In the process, we will introduce two additional model qualities, namely model simplicity and goodness of fit. We have refrained from introducing these two properties earlier because, as we will see, they are best understood quantitatively. In fact, it is precisely these properties that will provide a termination criterion for our search.
Chapter 4

A Bayesian Framework for Model Selection

In the previous chapter we saw how our intuition about what constitutes a good model can be neatly articulated with the help of model selection criteria. The application of these criteria, however, involves some degree of uncertainty. Probability theory is a mathematical formalism for quantifying uncertainty; it allows us to draw conclusions from uncertain facts, and to calculate the certainty of these conclusions given the certainty of the premises. For this reason, probability theory has become a standard tool in the natural and social sciences. Even the humanities have occasionally reaped benefits from it, and some musical applications in particular have appeared in the last few decades.\footnote{See (Meyer, 1956, 1957; Cohen, 1962; Temperley, 2004).} In this chapter, we will develop a probabilistic framework that will allow us to address the questions left unanswered in the previous chapter.
in probability theory, and can be skimmed if the reader is already familiar with the topic.\footnote{For a tutorial that covers similar ground and is also geared toward musical applications, see (Temperley, 2004). For a more general introduction, see (Pearl, 2000, pp. 2–6).} Section 4.2 outlines the general framework of Bayesian inference in which we will approach our problem of model selection. Section 4.3 explains the Minimum Description Length (MDL) principle, which is the basis of our particular implementation of Bayesian model selection.

### 4.1 Fundamentals of Probability Theory

Out of all candidate models which can generate the given corpus, model selection seeks to determine the most likely one. But the corpus is only a finite sample; whichever rule we might infer from it, we can never be sure that the rule will not be broken by some future instance not included in the original sample. Any regular patterns that may appear in the corpus affect our belief in a candidate model. For example, given the consistency of the cadential formula in Figure 3.1, we would be inclined to prefer a model that predicts that formula most of the time to a model that doesn’t. What we need is a way to quantify our belief in a given model, one that moreover takes into consideration the criteria presented in the previous chapter.

In the Bayesian interpretation, “probabilities encode degrees of belief about events in the world and data are used to strengthen, update, or weaken those degrees of belief.” (Pearl, 2000, p. 2) Our degree of belief in statement $A$ is
CHAPTER 4. BAYESIAN MODEL SELECTION

represented by a number $P(A)$—read “the probability of $A$”—that satisfies

$$0 \leq P(A) \leq 1$$

with $P(A) = 1$ if $A$ is certainly true, and $P(A) = 0$ if it is certainly false. The closer the value to 1 or 0, the higher the certainty that the statement is true or false respectively; the value 0.5 represents maximum uncertainty.

Our belief in $A$ will in general depend on any relevant knowledge that we may have, including observations and prior assumptions. We acknowledge this dependence by writing $P(A|K)$—read “the probability of $A$ given (knowledge) $K$”—which is known as a conditional probability. Any probability is in principle conditional, insofar as any belief is sensitive to background knowledge, some of it perhaps unstated. For practical reasons, however, we typically choose to suppress those parts of the background knowledge that are easily understood and that are assumed to be constant over the course of the study; we often simply write $P(A)$ if no conditions need to be explicitly stated.

In our study, we will write $P(M|C)$ to represent our degree of belief in model $M$ given the observed corpus $C$, the latter being the main source of knowledge that affects our belief in $M$. Any other background assumptions will be silently understood, but will not be included in our notation.

A probability system is such a prescription for calculating the value of $P(A|K)$ for all $A$ and $K$ in a domain of interest. The consistency of this system is ensured
with the help of two basic rules that relate the probabilities of complex statements to those of simpler ones. We will first state these rules, and then we will illustrate them through a simple example. Rule 1 (equation 4.1) asserts that, if $A$ and $B$ are mutually exclusive statements, the probability that either of them holds is equal to the sum of their individual probabilities:

$$P(A \text{ or } B|K) = P(A|K) + P(B|K)$$ (4.1)

given any common background knowledge $K$. $K$ of course may be empty, in which case we simply write

$$P(A \text{ or } B) = P(A) + P(B)$$ (4.2)

Rule 2 states that, for any $A$ and $B$, the probability that both statements hold is equal to the probability of $B$ times the probability of $A$ given $B$:

$$P(A \text{ and } B|K) = P(A|B \text{ and } K)P(B|K)$$ (4.3)

given any common background knowledge $K$. As before, if $K$ is empty, we may simply write

$$P(A \text{ and } B) = P(A|B)P(B)$$ (4.4)

If $P(A|B) = P(A)$, this means that our belief in $A$ does not depend on our knowledge of $B$; we then say that $A$ and $B$ are independent. In this case, Rule 2 implies that $P(A \text{ and } B) = P(A)P(B)$.\footnote{From this relation we may also deduce that $P(B|A) = P(B)$, since $P(A)P(B) = P(A \text{ and } B) = P(B \text{ and } A) = P(B|A)P(A)$. Independence is thus symmetric in $A$ and $B$.} Likewise, if $P(A|B \text{ and } K) = P(A|K)$, we say
that $A$ and $B$ are independent given $K$, which implies that $P(A \text{ and } B|K) = P(A|K)P(B|K)$. In what follows, we will write "$A$, $B$" instead of "$A$ and $B$", for the sake of simplicity.

To use one of the most common examples, the rules listed above allow one to calculate the probabilities associated with the outcome of a fair die. This outcome is one of six mutually exclusive possibilities, 1, 2, ... 6. By Rule 1 (equation 4.2), the probabilities for each outcome must add up to 1. If in addition the die is assumed to be fair, the probabilities for each outcome are the same, hence equal to $1/6$. Using Rule 1 again, we can calculate the probability for an even outcome (2, 4, or 6) to be $3 \times 1/6 = 1/2$. Using Rule 2 (equation 4.4), we can also calculate the probability of two successive 6's to be $P(\text{first 6 and second 6}) = P(\text{second 6}|\text{first 6})P(\text{first 6}) = 1/6 \times 1/6 = 1/36$; we have used the fact that $P(\text{second 6}|\text{first 6}) = P(\text{second 6})$, since successive throws of a die are assumed to be independent. More complex probabilities can also be calculated in this way.

For example, what is the probability of the sum of two successive outcomes being 10 or greater? Through successive application of Rules 1 and 2, we have

\[
P(\text{sum} \geq 10) = P(\text{sum} = 12) + P(\text{sum} = 11) + P(\text{sum} = 10) \quad \text{(Rule 1)}
\]
\[
= P(6,6) + [P(6,5) + P(5,6)] + [P(6,4) + P(5,5) + P(4,6)] \quad \text{(Rule 1)}
\]
\[
= 1/36 + 2/36 + 3/36 \quad \text{(Rule 2)}
\]
\[
= 1/6
\]
4.2 Bayes’s Law and Model Selection

Using Rule 2 (equation 4.4) with \( A \) and \( B \) switched gives \( P(A, B) = P(B|A)P(A) \), or

\[
P(B|A) = \frac{P(A, B)}{P(A)}
\]

Using Rule 2 once more, we can rewrite \( P(A, B) \) on the right-hand side to obtain

\[
P(B|A) = \frac{P(A|B)P(B)}{P(A)}
\] (4.5)

Equation 4.5, known as Bayes’s Law, or Bayes’s Inversion Formula, is one of the most famous relations in probability theory. Given how easy it is to derive, Bayes’s Law may in fact appear trivial; but, as we will see, it is actually very rich in applications. The reason is that, even though \( P(A|B) \) and \( P(B|A) \) are formally symmetric, in practice it is often a lot easier and more natural to estimate directly one of the two quantities; we can then use Bayes’s Law to calculate the other.

Let us illustrate this idea with the help of an example.

Consider a chanter who is able to improvise a chant melody on a given liturgical text, and assume that this chanter uses one of two cadences, say \( C_1 \) and \( C_2 \), each time he reaches the end of a grammatical phrase in the text.\(^4\) A grammatical phrase may be marked by a semicolon (\( ; \)) or a period (\( . \)) in the chanter’s (text-only) chant book. Suppose the chanter tells us that when he encounters a semicolon, he is equally likely to use either one of the two cadences. Assuming

\(^4\)This example is purely hypothetical, and is used here only for the purposes of illustrating Bayes’s Law; no music theoretic or linguistic conclusions should be drawn from it.
such a report to be truthful, we can write $P(C_1|.) = P(C_2|.) = 1/2$. When reaching a period, however, the chanter claims that it is three times more likely that he will use cadence $C_1$. Again assuming that this estimate is accurate, we can write $P(C_1.|) = 3/4$, $P(C_2.|) = 1/4$. Suppose now that we listen to the chanter performing cadence $C_2$ in service, and that we don’t have access to his chant book. Can we use our knowledge of the chanter’s style to infer which punctuation mark he encountered at the end of the grammatical phrase?

The answer to our question will of course take the form of a probability, since either punctuation is possible given our observation of cadence $C_2$. So we need to calculate $P(.)|C_2)$ and $P(.,C_2)$ based on all the information that we can collect. The probabilities that we obtained from the chanter are all of the general form $P(cadence|punctuation)$, and it is arguably more natural for him to think directly in terms of these probabilities, rather than in terms of the ones we need to know. (Try formulating a question that would solicit $P(punctuation|cadence)$ directly from the chanter.) Assume in addition that a linguist informs us that periods are a lot more likely to occur in Greek than are semicolons, with probabilities given by $P(.) = 7/8$ and $P(;) = 1/8$ respectively. Knowing that semicolons are a priori less likely must surely affect our belief in what the chanter saw, in addition to any information we may have that is specific to the chanter’s style.

As it happens, Bayes’s Law confirms this last intuition, telling us that all the above information is necessary for calculating $P(;|C_2)$ and $P(.,|C_2)$. More-
over, this information is also sufficient: making the necessary substitutions, equation 4.5 yields
\[ P(\cdot | C_2) = \frac{P(C_2 | \cdot) P(\cdot)}{P(C_2)} \] (4.6)
and an analogous expression for \( P(\cdot | C_2) \). We have direct values for \( P(C_2 | \cdot) = 1/2 \) (from the chanter) and \( P(\cdot) = 1/8 \) (from the linguist), and we can also calculate \( P(C_2) \) easily as follows:
\[
P(C_2) &= P(C_2, \cdot) + P(C_2, \cdot) \\
&= P(C_2 | \cdot) P(\cdot) + P(C_2 | \cdot) P(\cdot) \quad \text{(Rule 1)} \\
&= (1/4)(7/8) + (1/2)(1/8) \\
&= 9/32 \] (4.7)
Substituting these values into equation 4.6, we get \( P(\cdot | C_2) = (1/2)(1/8)(32/9) = 2/9 \). And since \( P(\cdot | C_2) + P(\cdot | C_2) = 1 \), we have \( P(\cdot | C_2) = 1 - P(\cdot | C_2) = 7/9 \). So a semicolon is still less likely than a period (2/9 vs 7/9), reflecting the overall relative probability of these punctuation signs prior to our observation of cadence \( C_2 \); but observing \( C_2 \) has made a semicolon more likely than it was before (2/9 > 1/8), since \( C_2 \) is less commonly used with periods \( P(C_2 | \cdot) = 1/4 \).

This particular example may not be momentous in its practical applications, but it exemplifies a general type of problem that is of great importance: \textit{given an observation that we made, can we calculate a probability for the underlying cause that triggered it, even though we may not be able to observe that cause directly?} According to Bayes, it is indeed possible. All we need for our calculation is (i) a probabilistic understanding of the underlying mechanism (the chanter’s choices), and (ii) knowledge of the \textit{a priori} probability of the cause, independent
CHAPTER 4. BAYESIAN MODEL SELECTION

of any specific observation that was made (the linguist’s input). Bayes’s Law then asserts that

$$P(cause|observation) = \frac{P(cause|observation)P(cause)}{P(observation)} \quad (4.8)$$

where $P(seeing)$ is understood as an overall probability for the observation, which takes into account all possible causes, and is calculated in the manner of equation 4.7. Note also that if we are only interested in identifying the most likely cause, without calculating its probability, we don’t even need to know $P(seeing)$, since it is the same for all causes. All we need then is to find the cause that maximizes the numerator in equation 4.8.

In equation 4.8, $P(cause)$ expresses our belief in a specific cause before the observation was made, and is therefore referred to as a prior probability, or simply a prior. $P(cause|observation)$, likewise reflects our updated belief in that cause after the observation was made, and is known a posterior probability, or simply a posterior. The quantity $P(cause|observation)$ encapsulates the mechanism that links an observation to its underlying cause. The interpretation contained in equation 4.8 is fundamental to the Bayesian philosophy, in which Bayes’s Law amounts to a concrete mechanism for updating our beliefs about the underlying state of the world (the “causes” of what we see) in light of new evidence (new “observations”) that we make.

Let us now turn to our original problem, namely to calculate $P(M|C)$ for
any model $M$ given some observed corpus $C$. In fact, this is another instance of
the problem type discussed in the previous paragraphs: viewed as a generator of
chants, $M$ models the source that generated corpus $C$, and thus $M$ represents
the unseen underlying cause of our observations. Using Bayes's Law, we should
be able to reduce our original problem to subproblems that are easier to solve.
This is indeed the case. Substituting $C$ and $M$ for $A$ and $B$ into equation 4.5,
we get

$$P(M|C) = \frac{P(C|M)P(M)}{P(C)} \tag{4.9}$$

Equation 4.9 suggests that in order to calculate $P(M|C)$, it is sufficient to cal-
culate each of the three probabilities on the right-hand side. We are thus led to
the following two requirements:

1. $M$ must be constructed as a probabilistic model, i.e., one that assigns a non-
zero probability $P(c_i|M)$ to each chant $c_i$ that it generates. The probability
$P(C|M)$ of the corpus as a whole will then be given by the product of all
the $P(c_i|M)$ for each $c_i$, since chants are assumed to be independently
generated.

2. We must specify a way of assigning a model prior $P(M)$ to each model $M$.

This prior would reflect an a priori degree of preference for model $M$ before
the corpus is consulted.
As for $P(C)$, it is in practice very difficult to calculate. Fortunately, it is also unnecessary: for our purposes, it is perfectly adequate to quantify the relative ranking of the candidate models, say in the form of probability ratios. Since $P(C)$ is the same for all models, it only contributes a multiplicative constant to equation 4.9; we may therefore simplify the latter to

$$P(M|C) \propto P(C|M)P(M)$$ (4.10)

This last relation is the fundamental equation of Bayesian inference. With the help of it, we can identify the most likely model $M$ as the one that maximizes the right-hand side $P(C|M)P(M)$. We can also calculate the relative probability of models $M_1$ and $M_2$ as

$$\frac{P(M_1|C)}{P(M_2|C)} = \frac{P(C|M_1)P(M_1)}{P(C|M_2)P(M_2)}$$ (4.11)

since the common constant of proportionality $P(C)$ drops out of equation 4.11.

Let us now take a closer look at requirements 1 and 2 above. Regarding requirement 1, we may note that a probabilistic framework can be particularly useful, perhaps even necessary, when modeling a musical style. Often it is not interesting—or even possible—to simply specify whether an instance belongs to the style or not. Instead, one may wish to specify how typical or how exceptional that instance may be with respect to the style. A probability value is a natural

---

$^5$ $P(C)$ is given by $P(C) = \sum P(C|M_i)P(M_i)$, where the summation is over all possible models $M_i$. Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER 4. BAYESIAN MODEL SELECTION

way to quantify this information. Another reason to consider probabilistic models is that musical rules often allow exceptions. A probabilistic framework makes it possible to work with even approximate rules in a formal and quantitative way. Finally, a musical model may have a possible psychological interpretation; our model in particular is intended to represent a chanter’s internalized knowledge of the Greek chant style. Due to natural variability in human behavior, psychological models are commonly formulated in a probabilistic framework. In Chapter 3, we demonstrated the usefulness of FSM’s in modeling the Greek chant melody. In fact, it is straightforward to augment a FSM into a probabilistic model—known as a Hidden Markov Model—by attaching probabilities to the FSM outputs. We will develop this idea further in Chapter 5.

Requirement 2 poses a difficult problem, namely how to determine the model prior \( P(M) \). As we saw in the earlier example involving the chanter’s cadences, prior probabilities do influence the outcome, sometimes quite strongly; they may therefore lead to a biased answer, if not properly chosen. Section 4.3 will address this final and rather important methodological consideration. There we will motivate the choice of the so-called description length prior, which favors simpler models over more complex ones.
4.3 The Minimum Description Length Principle in Model Selection

To address the problem of choosing a model prior, we will follow a route that may seem like a digression at first. We will consider the problem of data compression, which simply put, addresses the question of how to communicate data in the most efficient manner. As we will see, this amounts to finding a model that encodes the data in the most compact way. We will show that the data compression problem formally corresponds to Bayesian model selection, and that moreover, this correspondence yields a concrete expression for the model prior. We will argue that this prior implements the desired model qualities discussed in Chapter 3, offering an elegant answer to our model selection problem.

Consider the problem of communicating a data set, say over a channel such
CHAPTER 4. BAYESIAN MODEL SELECTION

as a computer network. For simplicity, let us assume that the data set consists of points in a two-dimensional plane (Figure 4.1). Each data point can be completely characterized by two real numbers, representing the coordinates of that point on the plane. Assuming there are \( N \) points in the data set, transmitting the \( 2N \) real coordinates is sufficient for reconstructing the data set at the other end of the channel. We refer to \( 2N \) as the *data description length* (DL).

In fact, we have only shown that the number \( 2N \) represents a worst-case scenario DL for a data set of \( N \) points, since the \( 2N \) coordinates are always sufficient to represent the data set. Is it ever possible to transmit the data set using fewer than \( 2N \) real parameters? And if so, under what conditions? This is the fundamental question of data compression. It can be shown within that framework that a shorter description length can be achieved if and only if we can identify regular patterns in the data set.

To see why this is so, consider the situation shown in Figure 4.2, in which all the data points fall on a straight line. In this case, the data set is less random: a linear relation holds between the coordinates \((x, y)\) of each data point, expressed algebraically as \( y = ax + b \), for fixed real numbers \( a \) and \( b \). This relation embodies the regularities identified in the data set, and can be thought of as a *model* for the latter, since it can be used to characterize membership in the set, as well as to generate new data points that could also belong to the set. Now if we know the parameters \( a \) and \( b \) that identify the model, we can reconstruct any

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 4.2: A data set consisting of $N$ points on a two-dimensional plane that fall on the straight line $y = ax + b$. Only $N + 2$ real parameters are now needed to communicate the data. Two of these parameters characterize the model (the straight line), and are thus referred to as the model description length (DL); the remaining $N$ parameters characterize the data given the model, and so represent the data DL.

given data point using a single real parameter, say its coordinate $x$, from which its $y$ coordinate can also be deduced. To characterize the data set completely, we therefore need $N$ real parameters for the data points (the data description length), plus two real parameters for the linear relation (the model description length). This amounts to a total description length of $N + 2$, which is in general much shorter than $2N$. In this way, a model that captures a regularity in the data set makes it possible to describe the model in a more economical way.

Let us apply these ideas to our original model selection problem. The corpus $C$ now plays the role of the data set. A complete characterization of this data set can be achieved, for example, by specifying the pitch, duration, and word stress for each note of chant. This gives a maximal description length of $3N$, 

\[
\text{Total DL} = N + 2
\]
CHAPTER 4. BAYESIAN MODEL SELECTION

where \( N \) is here the total number of notes in the corpus. But this description works even for a corpus that consists of randomly chosen notes. We have already observed the regular patterns that characterize the Echoi, including formulas and pitch-stress constraints. By analogy with the earlier data compression problem, we should therefore be able exploit these patterns to achieve a shorter description length for the corpus. For example, if the cadential formula of Figure 3.1 has a unique realization for each word stress pattern of the text, then given such a stress pattern, a single cadence label would enable us to reconstruct all the individual notes of the realization. This is much shorter than having to specify each note individually in the absence of a pattern.

As we showed in Chapter 3, FSM’s can capture the regularities of our corpus quite effectively and elegantly. In light of the previous paragraph, different FSM’s can be evaluated according to whether they can achieve a parsimonious description of the data. We could quantify this criterion by producing

1. an expression for the model description length (Model DL);

2. an expression for the data description length given the model (Data DL).

The most parsimonious model could then be identified as the one which minimizes the expression

\[
Total \ DL = Model \ DL + Data \ DL
\]  

Equation 4.12 expresses the Minimum Description Length principle in model
CHAPTER 4. BAYESIAN MODEL SELECTION

selection, developed by Rissanen (1989). As we will soon see, this principle has far-reaching consequences in that it relates to both Bayesian inference and to the model qualities advanced in section 3.2.

Let us now estimate the quantities Model $DL$ and Data $DL$ in equation 4.12. For a FSM, the model description length is the amount of information needed to specify its states, transitions, and output symbols—or input/output pairs in the case of a Finite State Transducer. If in addition the FSM is probabilistic, we will need to specify also the probabilities associated with each transition, input and output. This quantity can be easily calculated using a computer program. To get an intuitive feel for the model DL, it will suffice to compare the three models of Figures 3.7, 3.8, and 3.11 (pages 41, 42, and 47 respectively). The “coarse” model of Figure 3.11 clearly has the shortest description length of the three, as it involves the fewest states and transitions. For the same reason, the “fine” model of Figure 3.8 has the longest description length, and the optimal model of Figure 3.7 lies somewhere between the two. In fact the Model $DL$ quantifies another requirement that we might expect of a model, namely that of simplicity. Of course the “coarse” model of Figure 3.11 performs best in that respect. But equation 4.12 reminds us that simplicity is only one part of the picture.

The data description length is slightly more difficult to quantify. On intuitive grounds, we would expect the “fine” model of Figure 3.8 to offer the shortest description of the data: given the model, we can completely specify a generated
CHAPTER 4. BAYESIAN MODEL SELECTION

chant using a single parameter, namely the choice of path at the start state. By contrast, the "coarse" model of Figure 3.11 can generate almost any sequence of F's and E's between the initial and final G's; in order to specify a given chant using that model, we need to make note-to-note choices, resulting in a rather long description of the data. Again we expect the optimal model of Figure 3.7 to lie between these two extremes. Quantifying the Data DL in this manner can also be implemented in a computer algorithm, even though this time the calculation is considerably more involved.

Fortunately, there is an even simpler method to estimate the Data DL, and this method makes use of Shannon's coding theorem, one of the important early results in information theory. According to Shannon's theorem, if we wish to encode efficiently an output produced by a probabilistic source, the shortest description length we can ever achieve is given by $-\log p$, where $p$ is the probability assigned to that output by the source.\(^6\) The logarithm is commonly taken to be base 2, in which case DL is measured in digits of binary code, or bits. Using Shannon's result, it is not hard to compute the optimal code length for a corpus $C$ given model $M$: if a chant $c_i$ in the corpus is assigned probability $P(c_i|M)$ by the source, which is modeled by $M$, then the shortest DL for the chant is given

\(^6\)The original proof of the theorem appears in (Shannon and Weaver, 1949); see also (Rissanen, 1989, pp. 23–28). In qualitative terms, the theorem asserts that an efficient encoding must assign to the most frequent outputs (high $p$) the shortest codes (low $-\log p$). For example, the Morse code assigns the shortest pattern (a single dot) to the most frequent letter 'e'; the longest patterns are reserved for the least frequent letters, such as 'q' and 'z'.
CHAPTER 4. BAYESIAN MODEL SELECTION

by \(- \log P(c_i|M)\). The DL corresponding to the entire corpus is then given by

\[
Data\ DL = - \sum \log P(c_i|M) = - \log \prod P(c_i|M) = - \log P(C|M) \quad (4.13)
\]

where we have used the property of the logarithm function

\[
\log a + \log b = \log ab
\]

and Rule 2 of probability (p. 52).

It is useful to regard the quantity \(\log P(C|M)\) as a measure of how well the model \(M\) fits the data \(C\). Informally speaking, a model which fits the data closely will make the data more probable, and so will receive a high value for \(\log P(C|M)\). We will refer to this quality as goodness-of-fit. Thus the “coarse” model of Figure 3.11 achieves the poorest fit: since it generates the most chants, the observed corpus becomes a small fraction of the model’s total output and receives low overall probability. By contrast the “fine” model of Figure 3.8 generates nothing but the corpus, and thus achieves highest fit by assigning the corpus the highest overall probability. Once more, the optimal model (Figure 3.7) lies in the middle.\(^7\)

As we have seen, the two qualities of model simplicity and goodness-of-fit are each achieved at the expense of the other. Neither is desirable in the extreme, and the MDL principle provides a concrete way to achieve a balance between the

\(^7\)It is perhaps worth noting that goodness-of-fit generalizes and quantifies our earlier criterion of completeness: if a model fails to include an observed instance, then \(P(C|M)\) is zero, producing the lowest value for the goodness-of-fit.
two. After having outlined the significance of the MDL principle, we are now
ready to demonstrate its connection to Bayesian model selection.

Substituting Shannon’s expression (4.13) for Data DL in the MDL equa-
tion (4.12) we obtain

\[ \text{Total DL} = \text{Model DL} - \log P(C|M) \]  \hfill (4.14)

Taking the negative logarithm of Bayes’s equation (4.9) we obtain

\[ -\log P(M|C) = -\log P(C|M) - \log P(M) + k \]  \hfill (4.15)

where \( k = \log P(C) \) is a constant. Bayesian model selection requires maxi-
mizing \( P(M|C) \), or equivalently minimizing \(-\log P(M|C)\). Cast in this form,
equations 4.14 and 4.15 now show that the two minimization problems become
identical if we identify \(-\log P(M)\) with Model DL, or equivalently if we set the
model prior \( P(M) \) to be

\[ P(M) \propto 2^{-\text{Model DL}} \]  \hfill (4.16)

Equation 4.16 defines the description length prior. This choice of prior ensures
that Bayesian model selection coincides with the MDL approach, and that more-
over the model chosen in this way embodies the model qualities that we have
advocated in this and the previous chapters.\(^8\)

---

\(^8\)It may be useful to briefly pause and reconsider the full list of model qualities introduced
in the course of our discussion. These qualities are completeness, generalization, abstraction,
predictive power, interpretability, goodness-of-fit, and simplicity. We have already remarked
that some of these qualities are interrelated: abstraction enhances interpretability and gener-
As promised earlier, we have established a model selection framework that is grounded in a rigorous mathematical formalism. This concludes the methodological portion of the dissertation. In the following chapter we will present and discuss our results.
Chapter 5

Analysis of the Resulting Model

In this chapter we will describe our model, which is a probabilistic version of a FSM known as a Hidden Markov Model (HMM). In Section 5.1, we will begin with a general description of HMM's. We will then focus on the state merging algorithm introduced in Section 3.2, showing how its efficiency can be improved based on properties of the chant phrase’s structure. In Sections 5.2 and 5.3, we will examine a sample model representing the tonal/formulaic environment of syllabic Echos 1. In Section 5.2, we will describe the structure of the HMM, showing how its subgraphs correspond to chant phrase subdivisions. In Section 5.3, we will study the structure of the melodies generated by the model of Section 5.2. We will show that, given a fixed stress pattern of underlying text, the resulting melodies appear to be composed of short segments which we call chunks. The chunk decomposition of melody greatly simplifies and illuminates the properties of the Echos as predicted by our model, and points to a cognitive interpretation. In Section 5.4, we will discuss some of our model’s shortcomings and how we
might try to address these in the future. We will also discuss possible ways to validate our model against empirical data.

5.1 Training the Hidden Markov Model

A Hidden Markov Model (HMM) (Rabiner, 1989; Manning and Schütze, 1999, pp. 317–340) can be defined as a FSM with probabilities attached to its transitions and output symbols. Transition probabilities emanating from a given state add up to 1, and so do output probabilities on a given transition. The generation of an output string through a specific HMM path has probability equal to the product of all transition and output probabilities encountered in the process. The “hidden” in HMM comes from the fact that when we observe an output string, we generally do not know the path that generated it. More than one path can produce the same result, and the output string’s overall probability is obtained by summing up the probabilities of all its possible derivations. In a HMM, the states are not postulated explicitly; rather, they are inferred from a corpus through a training algorithm. As such, the states’ meaning may not be obvious, and will have to be retrieved by appropriate analysis. In fact, in many applications the meaning of the HMM states need not be uncovered at all. The model can be used as a “black box,” i.e., a device that produces the correct result, without us understanding its internal structure.

In our work, the Stolcke-Omohundro (SO) HMM training algorithm was cho-
CHAPTER 5. ANALYSIS OF THE MODEL

sen over its standard alternative (Baum-Welch). The reason is that we are particularly interested in determining the model’s connectivity in a reliable and interpretable way; a “black box” model, however compact and efficient, is not adequate for our purposes. The SO algorithm offers great flexibility in determining the model’s connectivity; moreover, as we will see shortly, the algorithm also allows us to incorporate constraints on the model’s geometry that may arise from the specific domain under investigation. This will not only improve the efficiency of the search for the best model, but will also lead to models that are easier to interpret.

The SO training algorithm starts from a maximal model that is overfitted to the corpus, and that is gradually improved by appropriate state merges. This process was described in some detail in Section 3.2. In addition, if the FSM is probabilistic, we need to update the probabilities on the model’s transitions and outputs after each merge. More specifically, merging together two states involves combining the sets of transitions in and out of these states, along with their attached outputs. The associated transition and output probabilities are then recalculated as weighted sums of the corresponding probabilities before merging. This calculation is local, in that it only affects the merged states and their direct neighbors. More on updating the probabilities can be found in (Stolcke and Omohundro, 1994, pp. 7–27).

A complete model of an Echos typically involves hundreds of states. In such
a graph, searching for the best sequence of state merges can be computationally very costly. The interval-based representation of melody introduced in Section 3.1 limits the possibilities somewhat, since only states with the same pitch label need be considered for merging. But we must still try to identify the best merges early, so that the algorithm does not waste time considering poor choices while the graph is large.

One efficient search strategy is to select at each step the state merge that results in the lowest description length (DL), as defined in Section 4.3. This procedure works well most of the time; however, the best immediate choice need not lead to the lowest DL overall. We have found it useful to occasionally “veto” a lowest-DL merge, choosing the second-lowest one instead, if the rejected merge involves members of different opening or cadential formulas. In our experience, such an intervention eventually leads to a set of lower-DL choices, and the resulting model is always easier to interpret.

Another way to make the merging more efficient is to identify all states corresponding to invariant pitches of phrase families, and to merge these before the start of the search. Such states correspond to:

- the last note of a cadential formula,

- the goal pitch of an (optional) opening formula, and

- the next-to-last stressed syllable of a cadential formula.
For the melodic family of Figure 3.1 (p. 33), these invariant pitches occur at the end of the phrase, and after the first and fourth dashed lines respectively.

5.2 The Structure of the Hidden Markov Model

The model presented in this section was obtained from a sample of fourteen syllabic chants in *Echos* 1, totaling 960 notes. The chants appear in the Sunday morning office (Gk *Orthros*), and are recorded in the modern *Anastasimatarion* (Vallindras, 1998). This chant sample will suffice for the present illustration. Analysis of larger samples has been performed, and is still under way. While a larger sample generally reveals more melodic possibilities, the findings reported in this section still apply.

The HMM that results from the training process of Section 5.1 is so large that it cannot be usefully displayed as a graph, unless we identify some meaningful
Figure 5.2: Subgraph of Figure 5.1 corresponding to the opening formula OF.01. States and transitions are labeled in a manner similar to that of Figure 3.5 (p. 38), with pitch labels suppressed from the transitions. Probabilities attached to transitions and outputs have also been omitted.

substructure. The simplest way to achieve this is to isolate subgraphs that correspond to a phrase’s structure as outlined in Section 3.1. One opening formula and five cadential formulas were identified and separated from the rest of the graph. The opening formula, labeled OF.01, is the one identified in Figure 3.1, p. 33 as the portion of melody from the beginning to the first dashed line. Of the cadential formulas, four correspond to imperfect cadences (ateleis) on the pitch G, and are labeled CF.G01 through CF.G04; the first of these has also been identified in Figure 3.1 as the portion of melody from the next-to-last dashed line to the end. The last of the five cadential formulas corresponds to internal or final perfect cadences (enteleis, telikai) on the pitch D, and is labeled CF.D01.
Figure 5.3: Subgraph of Figure 5.1 corresponding to the cadential formula CF.D01. States and transitions are labeled as in Figure 5.2.
CHAPTER 5. ANALYSIS OF THE MODEL

The remainder of the graph corresponds to the phrase’s “body.”

Figure 5.1 shows the overall structure of the model in terms of the connectivity of its subgraphs. Figures 5.2 and 5.3 show the internal structure of two representative subgraphs, corresponding to formulas OF_01 and CF_D01 respectively, as well as their connection to the rest of the graph. Graphs for the remaining formulas and for the phrase body are found in Appendix C.

5.3 Melodic Chunks and Decision Points

Despite the subgraph structure, the melodic properties of the Echos may not be entirely transparent by simply looking at the HMM graph. Moreover, we have set out to model a chanter’s internalized knowledge, and one may question the psychological relevance of a complex representation such as that of Section 5.2. However, we must remember that in any given situation, a chanter need not contemplate all the melodic possibilities of the Echos at once. Rather, he must be able to select among those possibilities compatible with the words he has to render, typically planning and executing one phrase at a time. When we put together all possible settings of a given phrase of text as predicted by our model, a simpler picture emerges.

Let us illustrate the process with the help of an example. Consider the phrase of Greek text shown below the staff in Figure 5.4, excluding the portion in square brackets. The phrase is twelve syllables long and carries the stress pattern [S1
Figure 5.4: The family of all predicted realizations of a given text phrase, as generated by the HMM of Figure 5.1. The family is organized by a decision tree, with decision points marked DP1 and DP2. The melodies are thus divided up into chunks S1 through S4. Chunks S3 and S4 contain instances of cadential formulas CF.G01 and CF.D01 respectively. The number on the right of each staff is the probability assigned to the corresponding melody. Translation: [Now and always] and unto the ages of ages. Amen.

S1 S1 S2 S1 S1 S1 S2 S1 S1 S2] as shown by the metric grid above the words.

The possible settings of this stress pattern can be obtained by following different paths in the HMM of Figure 5.1 and its subgraphs. To obtain a solution, we must first fix the goal state of the phrase to be an end state of a cadential formula. Such end states are shown as squares in graphs such as that of Figure 5.3. (In that figure, the end state is labeled D:53.) Fixing the goal state of the phrase in this way will prevent the melody from cadencing prematurely starting a new musical phrase in the middle of the text phrase.\footnote{Phrase boundary information is a necessary part of the text input, since it cannot be inferred from the stress pattern alone. This is also part of the knowledge needed to realize the melody properly within the Echos. In the present version of the model, this information is added “by hand” as discussed here. Another way to do this would be to annotate the corpus with text phrase boundaries, which of course are not present in the score. The end-of-phrase alignment would then be learned as part of the HMM inference process. A future version of the code might implement this.} For example, let us align the end of the text phrase in Figure 5.4 with the end state D:53 of Figure 5.3; the
latter represents the cadential formula CF.D01. The resulting melodic line is shown on the third staff of Figure 5.4. It is a worthwhile exercise to trace the path of the melody in the graph of Figure 5.3. After traversing the subgraph labelled “Body” in Figure 5.1, we will have obtained the first eight notes of that melody, reaching pitch E and state E:851. The stress pattern [S2 S1 S1 S2] of the remaining text can then be set to music following a path through the states E:851–G:108–F:870–E:871–D:959–D:53 of Figure 5.3. This generates the pitch sequence G F E D D which completes the melody. Note that the transition from G:108 to F:870 makes use of a S0 input. S0’s can be freely inserted into the original stress pattern, since they correspond to no syllable change. This makes it possible to introduce occasional short melismas, which are ornamental in the mostly syllabic style studied here.

In fact, this process can potentially generate a very large number of melodies, including unwanted ones, even for a fixed phrase of text such as the one considered above. It will be necessary to limit our attention to the most significant of these melodies based on the probability assigned to each melody by the model. As it turns out, only a small number of the generated melodies receive significant probability. For the text phrase in question there are only three, and these are listed in Figure 5.4 along with their corresponding probabilities. All other generated solutions are rated less likely by a factor of at least 100; we will choose
CHAPTER 5. ANALYSIS OF THE MODEL

... to treat these as “noise” rather than as meaningful solutions. Such “noise” behavior is common in probabilistically trained models such as the present one.

Let us now take a closer look at the solution Family of Figure 5.4. We first observe that all the melodic material present in the figure derives from only four segments—which we will refer to as chunks—labeled S1 through S4. Of these, S3 and S4 contain realizations of cadential formulas CF_.G01 and CF_.D01 respectively. The remaining two chunks fall inside the phrase body. In fact, S1 and S2 are diatonic transpositions of each other; because of their common intervallic pattern, they both fit the stress pattern of the words.

Next, let us observe that once we know the chunk structure shared by this solution family, we can completely identify any of its members by specifying the melodic choices at only two locations, one in the beginning, and one near the middle. We will refer to these locations as decision points. As shown in Figure 5.4, the set of melodic choices at these points can be concisely represented by a decision tree. This behavior is not unique to this particular family of solutions. Figure 5.5 shows another network of melodic possibilities corresponding to a different text phrase.

The situation revealed in Figures 5.4 and 5.5 is quite typical. A chant phrase normally contains two to three decision points, and these points divide the phrase

---

2 Of course the choice of cutoff probability for discarding a solution is inevitably somewhat subjective. Our present choice is perhaps justified by the two orders of magnitude separating the probabilities of the discarded solutions from those of the retained ones.
CHAPTER 5. ANALYSIS OF THE MODEL

Figure 5.5: Another family of realizations corresponding to a different text phrase. The family shows an organization similar to that of Figure 5.4. Chunks S8 and S9 are variants of S4 and S3 respectively. The probability assigned to each melody is here suppressed for simplicity.

into (usually formulaic) chunks. Each chunk is about six to nine notes long. Minor variations within a chunk are sometimes observed, such as a pattern of A A G substituting for A G G. These variations are not shown in our examples; here we simply present a chunk in its prototypical form.

As we have just shown, given an Echos and word stress pattern, the chunk decomposition allows us to represent a melodic phrase by only two to three parameters, namely the melodic choices at the decision points. In fact, what we observe here is an instance of compact coding of the data through an appropriate model that captures their regularities. In Section 4.3, we discussed this idea in a general setting.\textsuperscript{3} Here we can see a specific realization of it. The psychological implications of this compact coding of the melody will be discussed in Section 6.2.

Comparison of chunks occurring in other Echoi is currently in progress. Our

\textsuperscript{3}See p. 64.
results so far indicate that, even though cadential formulas are generally Echos-specific and fixed in pitch, opening formulas and phrase body chunks can be diatonically transposed, and can even appear in several Echoi. This suggests stress-pitch constraints shared by all the Echoi that determine a chant melody’s interval patterns. When combined with an Echos’s specific pitch system, such as the one shown in Figure 2.1, p. 20, these constraints give rise to concrete melodic formulas. The sharing of transposable pitch patterns by different Echoi was informally observed in our discussion of Figures 2.4–2.11.\textsuperscript{4} Chunks and decision points offer a framework for studying this phenomenon precisely.

\section*{5.4 Evaluating the Model}

Having presented our main results, we will now briefly comment on our model’s main weaknesses and how we might address them in the future. We will also discuss possible ways to validate our model empirically.

Perhaps the most important problem in the current training process is the occasional human intervention required to prevent bad merges, i.e. the “vetoing” described in Section 5.1. Ideally, the entire training process should take place algorithmically. This is not only a practical consideration, but also a reflection of the desire to remove any possible sources of arbitrariness, deriving all choices from clearly stated principles. It should be noted, however, that at the heart of

\textsuperscript{4}See p. 30.
the problem lies the question of how to avoid local minima, which plagues all minimization algorithms, not just the present one. In other words, evaluating merges one at a time can never guarantee that the Minimum Description Length overall will eventually be reached. The early identification of good candidate merges, also outlined in Section 5.1, at least helps alleviate the problem. Since it is based on clearly stated phrase structure criteria, one should be able to implement it algorithmically in the future.

Another important question concerns choosing the appropriate corpus size. We have found that if the corpus is too small, the algorithm may never converge to a minimum, and thus over-merging will inevitably occur. Presumably a corpus that is too small does not have enough redundancy to establish the recurring of patterns as significant, i.e. formulaic. This problem does not occur if the corpus is made sufficiently large. But increasing the corpus size more than necessary causes excessively long model training times. Besides, the digital encoding of a score is time consuming for the user. It is therefore important to be able to establish when the corpus size is adequate. To our knowledge, no theoretical treatment of the problem exists in the literature on Bayesian inference. In this project, we select our corpus size by trial and error.

Finally, it should be noted that the present programming framework is a patchwork of different environments, including a Common Lisp and a Prolog system, and lacks a graphical user interface. In this framework, monitoring the training
CHAPTER 5. ANALYSIS OF THE MODEL

process is extremely awkward, as is the calculation of predicted solutions such as those of Section 5.3. Until this practical problem is addressed, serious theoretical advances will be very difficult. We are currently working on translating the code into a better environment. (See comment at the end of Appendix A.)

Finally, it should be stressed that a quantitative predictive model such as ours opens the possibility of empirical testing. This is an invaluable new opportunity for model development and should be fully exploited. More specifically, the model’s performance can be tested against a variety of empirical data. The simplest type of such data could be a new corpus of written chants that was not part of the model’s training. One would hope that the model consistently assigns high probabilities to actual chants—even ones it hasn’t “seen” yet—compared to, say, random sequences of pitches. Another type of data that could be used to validate the model in a similar way would be a corpus of transcribed expert improvisations. Conversely, the model’s output melodies could be rated for musical quality, or even for correctness, by expert carriers of the chant idiom. These last two possibilities, as well as some difficulties they entail, are outlined further in Section 6.2. Empirical testing of the type suggested above complements the Bayesian inference framework and can indeed serve as a validation of the MDL principle itself; it will no doubt uncover new inadequacies of the model and thus offer further opportunities to develop and fine-tune it.

This chapter concludes the main presentation of our method and results. Ad-
ditional details can be found in the Appendices. In the next and last chapter, we will discuss the relation of our research to previous work and consider its implications and some future directions.
Chapter 6

Implications and Future Directions

6.1 Oral Transmission and Western Chant Studies

Alignment of chants similar to that in Figure 3.1 goes back to historical comparative studies of Western idioms, most notably in the work of Helmut Hucke (Hucke, 1955; Nowacki, 1998). The method is often referred to as paradigmatic analysis. We should stress, however, that the FSM technique presented here is much more general than paradigmatic analysis, since it need not limit itself to comparisons of melodies that follow parallel paths in melodic space, but permits more complex relations, such as the one presented in Figure 5.5, which contains several partial alignments.

The question of oral transmission and how it affects melodic style has been discussed extensively in the context of Western chant scholarship. In a seminal work, Leo Treitler (1974) proposed that in order to understand the melodic
idiom of Gregorian chant, one must take into account the cognitive constraints involved in oral production. In the same article, Treitler outlined specific models of melodic families that loosely characterize their fixed and variable parts.\(^1\) Edward Nowacki (1986) applied the method of melodic alignment to second mode Old Roman tracts. He demonstrated that within these families, melodic variation involves insertions and deletions that are motivated by word stress, a result that is strikingly similar to ours. Matthew Chen (1983) studied the cadential formulas of Gregorian psalm tones. He showed how the stress pattern near the end of the verse determines the tone’s cadential melody through a set of formal rules. Finally, Peter Jeffrey (1992) proposed that the question of orality in Western chant will be better addressed by studying oral transmission in living chant cultures, in an inter-disciplinary effort informed by historical musicology, ethnomusicology, and music psychology.

This work was partly inspired by all of the above studies. Following Jeffrey’s program, we present an exploration of orally transmitted knowledge in the living chant tradition of the Greek Orthodox church. We agree with Treitler’s premise that the cognitive properties of melodic production will be reflected in the chant’s style, and set out to capture these properties in a precise quantitative model. Following the example of Hucke, Treitler, Nowacki, and others, we begin with a systematic comparison of chant melodies, aiming to uncover their un-

\(^1\)See (Treitler, 1974), pp. 358, 361.
derlying regularities. Borrowing techniques from computational linguistics and machine learning, we seek a model that is as complete and rigorous as that of Chen, and that can fully characterize a much more complex melodic system. The psychological significance of our research remains to be addressed.

6.2 Towards a Cognitive Model of Melody

We will continue with a discussion of our model's psychological implications, particularly in relation to chunks and decision points.

Let us begin with decision points. We should emphasize that in our model, such points are simply formal devices that help organize a solution family; they need not be interpreted as conscious choices that take place in the course of performance. We have found however that, once identified by a chanter, a decision point can actually assume a mnemonic role. For instance, we have seen a chanter mark a certain place in his text to remember that he must move in a given direction. Likewise, in my personal experience of practicing chant improvisation, I have found that a persistent impasse can easily be fixed once I correct a specific poorly chosen note; the rest of the melody then easily falls into place.

Let us now turn to chunks, a term which we have actually borrowed from psychology. In an influential paper, George Miller (1956) introduced the idea of chunking in the mental representation of sequences. More specifically, Miller argued that the short-term recall of a sequence deteriorates as its length increases,
CHAPTER 6. IMPLICATIONS AND FUTURE DIRECTIONS

but can be greatly facilitated if the sequence is hierarchically organized into segments, which he called chunks. Miller attributed this phenomenon to capacity limits in short-term memory; presumably a well-learned chunk is mentally coded more compactly than the sum of its parts, and this allows for more efficient use of memory in a real-time task. A frequently cited article by Deutsch and Feroe (1981) has applied directly Miller’s ideas to melody.

The chunks of our model are consistent with Miller’s theory, including their typical size, which Miller estimated to be five to nine items. (In our case, it is six to nine notes.) Many controversial issues still exist concerning sequence representations: the exact chunk size, and even the very notion of capacity limit have recently been questioned (Cowan, 2001). However, it seems reasonable to suppose that our chunk structure somehow facilitates real-time processing: chunks represent frequent and well-learned patterns that can be produced relatively effortlessly; as a result, only decision points require heightened attention in performance. This could apply to both the recall of pre-existing melodies and to the improvisation of new ones.

In this dissertation, we have shown how to construct a model of Greek chant melody, based on analysis of a written corpus. Melodies in that corpus have been presumably recorded—and corrected—based on a chanter’s knowledge of the style, or competence in Chomsky’s terms. To pursue the psychological implications of our model further, we eventually have to come to terms with oral
output—including unintended one—as it occurs in real time, what Chomsky (1965) has described as *performance*. This may involve applying our algorithm to a corpus of transcribed improvised chants; the resulting HMM could then be compared to ones obtained from written corpora. There is good reason to believe that the oral models will not be as clean and simple as the one presented in Section 5.2. For instance, an unintended change of *Echos* can happen in performance, which a single-*Echos* model like the one in Figure 5.1 cannot capture. Nevertheless, on the basis of the psychological evidence mentioned earlier, we might still expect to observe the chunk structure of Section 5.3, with its associated points of heightened attention. It is perhaps precisely at these points that errors are more likely to occur.

Experiments involving perception tasks are also possible and could prove to be a useful complement to the above production task studies. For example, one could ask native carriers of the idiom to evaluate melodies produced by a HMM in order to determine whether the model’s typicality rating agrees with that given by people. Or one could experimentally measure a native carrier’s melodic expectation at various points in the melody, and see whether that expectation relates to the transition and output probabilities at the corresponding HMM state.

In addition, data collected from different people performing both production and perception tasks could be analyzed individually. One should expect this anal-
ysis to identify individual differences that reflect different levels of skill. Understanding skill acquisition is one of the most important goals of cognitive modeling and could have profound pedagogical consequences for Greek chant in particular, and most likely for ear training in general.

6.3 The Question of Model Selection

The question of model selection raised in Chapters 3 and 4 has been discussed extensively by philosophers of science, who seek to understand what drives the development of scientific theories. We have raised the question in the present dissertation because we believe it is regrettably absent from most music theoretic discussions. One notable exception is the work of Matthew Brown (2005), who approaches the question from the philosophy of science perspective. It is perhaps appropriate to say a few words about Brown’s approach and its possible connection to ours.

In his forthcoming book “Explaining Tonality: Schenkerian Theory and Beyond,” Brown discusses what form music theories may take, how they are built, and how they are evaluated. In the process of theory development, Brown believes that music theorists typically try to balance what Quine has described as “the drive for evidence and the drive for system.” According to him, the former demands that “theoretical terms should be subject to
observable criteria, the more the better, the more directly the better, 
other things being equal” while the latter insists that these terms 
“should lend themselves to systematic laws, the simpler the better, 
other things being equal.”

Following Kuhn, Quine, and others, Brown presents six criteria for theory selec-
tion that give a more concrete expression to Quine’s dichotomy. The criteria are 
**accuracy**, **scope**, **fruitfulness**, **consistency**, **simplicity**, and **coherence**.

**Accuracy** expresses how well a theory measures up to our experience, both 
past and future: a good theory fits the data well and also makes good predic-
tions. **Scope** refers to the array of phenomena or range of properties that a 
theory may cover. **Fruitfulness** relates to a theory’s ability to offer connections 
and predictions outside of its original context. **Consistency** refers to a theory’s 
ability to make unequivocal predictions. **Simplicity** is a property of the theory’s 
structure. Finally, **coherence** expresses how well a theory resonates with theories 
in different but related domains.

Even though our list of model selection criteria from Chapters 3 and 4 does not 
seem to align completely with those of Brown’s, some correspondences are read-
ily observed. Thus, our **completeness** and its refinement, **goodness-of-fit**, corre-
spend to Brown’s accuracy with respect to past experience; accuracy with respect 
to future occurrences corresponds to our **predictive power**. In our quantitative 
framework, we have taken consistency for granted. We believe that our **model**
simplicity is a quantitative expression for Brown’s simplicity. Brown’s coherence relates to what we have called interpretability. Moreover, once we ensured the latter, we explicitly addressed our model’s coherence with theories of cognition. Brown’s scope and fruitfulness do not explicitly correspond to qualities identified in our approach. Finally, and perhaps most remarkably, Quine’s “drive for evidence vs drive for system” are in close correspondence with our two qualities of goodness-of-fit and model simplicity, as quantified by the Data Description Length and Model Description Length respectively; the process of balancing the two opposing forces assumes algorithmic form through state-merging and the MDL termination criterion.

The differences between Brown’s list and ours should perhaps not be surprising, as the two frameworks emerge out of distinct theoretical traditions, namely that of the philosophy of science and that of machine learning. It is precisely for this reason that we would like to view the two pictures as complementary rather than at odds with each other. In this dissertation, we could have presented our algorithm based entirely on the two criteria of goodness-of-fit and model simplicity. But in seeking to expand our list, we have sought to come closer to the philosophers’ concerns, partly inspired by, and responding to, Brown’s work. The correspondences between the two frameworks could perhaps be pursued further, but this belongs in a future project.
6.4 The Value of Computational Modeling for the Study of Music

In this dissertation, we have undertaken a study of an idiom that is fairly complex and not well understood from the theoretical standpoint. Moreover, on the grounds of its history and performance practice, we have argued that a music theory of Greek chant will require input from cognitive studies. Needless to say, neither the complexity nor the cognitive ramifications should be considered special to Greek chant. One could in fact argue that they pertain to all music. The present methodology was not conceived as special-purpose, but rather as one of a much broader scope. It is offered here as a case study that testifies to the value of computational modeling for the study of music.

The advantages of the computational approach begin with some of the more mundane aspects, such as the possibility offered by computers to handle large volumes of data quickly, accurately, and exhaustively. But in this work we have gone beyond the aspect of practical convenience, and have tried to address the question of how to build and evaluate models. We have achieved this through algorithms that have allowed us to see patterns in the data that were very hard or impossible to see otherwise. Without the help of our framework, we might not have been able to articulate our intuitions about the chant’s melodic behavior beyond the informal discussion of Sections 2.2 and 3.1. Our computational framework has thus enabled us to pursue our observations and intuitions to their
logical conclusions, and helped bring out their latent potential.

In addition, once we obtained our model, we were able to calculate its consequences in the form of predictions that show us how the model measures up to our experience. This has opened up the possibility for empirical testing, which provides further input to the model development process. The model’s consequences in particular have shown us some intriguing connections with cognitive psychology. The computational approach has thus demonstrated its power to integrate our knowledge, bringing ideas from one discipline to bear upon the other.

In showing how music theory and music psychology can work in synergy to achieve much more than each can achieve alone, the computational framework offers us new languages to carry our discourse, often transforming our original problems in ways we had not anticipated. And perhaps the value of computation is seen at its greatest once it allows us to ask questions that were not possible to ask before.
Bibliography


Appendix A

The HMM Implementation

In encoded form, each chant is represented by a list of symbols. Each symbol is a
composite entity that represents a note by combining all the textual and melodic
variables relating to that note. Each symbol is a list of variables, and has the
following form:

\[(Stress \ (Duration \ Interval \ Pitch))\]

Variable Stress receives values 2, 1, 0, corresponding to a stressed syllable, an
unstressed syllable, and no syllable change respectively. Variable Duration en-
codes the duration of the previous note in so called reciprocal representation,
with 4 for a quarter note, 8 for an eighth note, etc. Variable Interval contains
the diatonic interval from the previous note measured in steps, where 0 represents
unison, and negative numbers represent descending intervals. Variable Pitch is
itself composite, and has the following form:

\[(LetterName \ DiatonicN \ ChromaticN)\]
Variable *LetterName* contains the letter name of the note used in the Western transcription; it is redundant and is only included to make the encoding easier to read. Variable *DiatonicN* is a *diatonic pitch number*, with 0 representing C2, 1 representing D2, etc. The choice of zero is arbitrary, and simply reflects the low end of the comfortable singing range. Likewise, variable *ChromaticN* is a *chromatic pitch number* used to distinguish among the different chromatic inflections of the same scale step. In this representation, a whole tone is divided into six parts, with 0 representing uninflected C2, 6 representing uninflected D2, etc. The complete symbolic representation of a note thus takes the form

\[ (Stress \ (Duration \ Interval \ (LetterName \ DiatonicN \ ChromaticN))) \]

For example, the symbol

\[ (1 \ (4 \ -1 \ (F \ 10 \ 51))) \]

is interpreted as follows, reading from left to right: an unstressed syllable (1) is realized after waiting a quarter note (4) by stepping down (-1) to a diatonic F (10 51).

In this project, HMM’s are implemented mostly in the *Common Lisp Object System* (CLOS), the standard object-oriented extension of Common Lisp. For more information on this programing framework, see (Steele, 1990). Only a small portion of the code is written in Prolog (Bratko, 1990), and will be discussed below (p. 105). Complete code listings appear in Appendix B. The remainder of
this appendix explains what that code does and addresses some implementation issues.

File \texttt{HMM-data.lisp} (p. 114) contains class definitions for objects implementing the data. Class \texttt{data-point} corresponds to individual notes as described above and has the following fields: \texttt{input}, containing the level of word stress; \texttt{output}, containing the rhythm, interval, and pitch variables; \texttt{comment}, containing information that does not take part in the computation and helps the human reader mark and recognize the different data elements; and \texttt{state}, containing the label of the HMM state encountered when the data set is taken through a HMM path. Class \texttt{data-sequence} corresponds to a complete chant and has the following fields: \texttt{data-points}, containing a list of the individual notes comprising the chant; and \texttt{size}, containing the number of notes in the chant. Class \texttt{data-set} corresponds to a corpus of chants and has the following fields: \texttt{data-sequences}, containing a list of the individual chants comprising the corpus; and \texttt{size}, containing the number of chants in the corpus. The corpus is encoded manually in a text file, and is loaded by function \texttt{load-data-set}, which reads the file and creates the appropriate data-set object.

The classes for the data structure representing a HMM are defined in file \texttt{HMM.lisp} (p. 118). Each HMM contains several sub-components (\texttt{HMM-members}) that can be \texttt{HMM-elements}, \texttt{HMM-containers} for other \texttt{HMM-elements}, or both. A \texttt{HMM-element} can be a \texttt{HMM-output}, a \texttt{HMM-input}, a \texttt{HMM-transition}, or a
HMM-state. A HMM-container can be a HMM-input, a HMM-transition, or a HMM-state, or the HMM itself. Each HMM-container has an elements field that contains a list of its HMM-elements. Thus, each HMM has a set of HMM-states; each HMM-state has a set of outgoing HMM-transitions; each HMM-transition has a list of HMM-inputs attached to it, and each HMM-input is paired up with one or more HMM-outputs.\(^1\) As it may take several hours or days to calculate a HMM from data, it is important to be able to save a HMM and retrieve it for later use. A HMM can be saved to a file using function `save`, and loaded from a file using function `HMM-load`.

Fitting a HMM to a set of data according to the criteria discussed in Chapter 4 is a process referred to as training the HMM on the data. The code that performs the training forms the heart of our HMM system, and is found mostly in file `HMM-SO-training.lisp` (p. 136). Function `make-SO-model` takes a data-set as input and creates the maximal HMM used at the start of the Stolcke-Omohundro (SO) algorithm. (See Figure 3.8, p. 42.) During training, the user is given the option of trying manually a merge using function `try-merge`; alternatively, the user may ask the system to recommend a merge using function `suggest-merge-with-min-total-length`. In both cases, the description length of the resulting model after the merge is calculated and displayed as a possible criterion for accepting the merge. Once the merge is accepted, function `do-merge`.

\(^1\)Recall the definition of a Finite State Transducer, p. 37.
permanently updates the model. This user-supervised mode is useful in the early part of the training in which the state merges may be chosen by hand according to the criteria of Section 5.1. In the later part of the training, the model runs on its own, accepting at each step the merge that minimizes the total description length, until a minimum is reached or until the process is terminated by the user.

Files `HMM-coding.lisp` (p. 128) and `HMM-data-coding.lisp` (p. 131) contain the code used to calculate description lengths as outlined in Section 4.3.

Once a satisfactory HMM is obtained, it is important to be able to calculate all the chants generated by that model. This portion of the code was written in Prolog, which makes the code remarkably simple. The program appears in file `FSM.pro` (p. 151) and is less than a page long! This code was used to obtain results like those reported in Section 5.3. The program that converts the HMM data structure from Lisp to Prolog appears in file `HMM-to-prolog.lisp` (p. 148).

It is very important to be able to visualize a HMM using graphs such as those of Figures 5.2–5.3 and C.2–C.8. The program used to produce these figures is `GraphViz`, created at the AT&T Bell labs.\(^2\) File `HMM-to-graph.lisp` (p. 146) contains code that converts a Lisp HMM data structure to the input format accepted by `GraphViz`.

As it stands, the implementation is not user-friendly: both the chant encoding and the SO search are extremely time consuming. A new implementation

is currently under way, using the programming framework *Lush* by Yann Le-Cun and Leon Bottou.\(^3\) *Lush* is a Lisp-based system that combines the flexibility of symbolic manipulations in Lisp with high performance number-intensive computations in the programming language C. *Lush* includes an extensive library of graphics; making it possible to program a user interface that is more friendly and efficient. The improved future version of my program will be made available on the Internet. More information is forthcoming on my web site at http://theory.smusic.nyu.edu.

Appendix B

Complete Code Listings

;;; ===========
;;; load-HMM.lisp
;;; ===========

;;; -- Constants ---------------------------------------------

(defconstant *project-root*
    "Mac OS X:Users:pm:Documents:Projects:_Development:HMM:"
)

(defconstant *code-root*
    (concatenate 'STRING *project-root* "Code:HMM:"
)

(defconstant *data-root*
    (concatenate 'STRING *project-root* "Code:Data:"

(defconstant *training-root*
    (concatenate 'STRING *project-root* "Code:Training:"

;;; -- Code ---------------------------------------------

(load (concatenate 'STRING *code-root* "HMM-utilities.lisp"))
(load (concatenate 'STRING *code-root* "HMM-generic.lisp"))
(load (concatenate 'STRING *code-root* "HMM-data.lisp"))
(load (concatenate 'STRING *code-root* "HMM.lisp"))
(load (concatenate 'STRING *code-root* "HMM-coding.lisp"))
(load (concatenate 'STRING *code-root* "HMM-data-coding.lisp"))
(load (concatenate 'STRING *code-root* "HMM-50-training.lisp"))
(load (concatenate 'STRING *code-root* "HMM-to-graph.lisp"))
(load (concatenate 'STRING *code-root* "HMM-to-prolog.lisp"))

107
;;; MACROS

;;; Assignment

(defun setq+ (lhs rhs)
  '(setq ,lhs (+ ,lhs ,rhs)))

(defun setq- (lhs rhs)
  '(setq ,lhs (- ,lhs ,rhs)))

(defun setq* (lhs rhs)
  '(setq ,lhs (* ,lhs ,rhs)))

(defun setq/ (lhs rhs)
  '(setq ,lhs (/ ,lhs ,rhs)))

(defun setf+ (lhs rhs)
  '(setf ,lhs (+ ,lhs ,rhs)))

(defun setf- (lhs rhs)
  '(setf ,lhs (- ,lhs ,rhs)))

(defun setf* (lhs rhs)
  '(setf ,lhs (* ,lhs ,rhs)))

(defun setf/ (lhs rhs)
  '(setf ,lhs (/ ,lhs ,rhs)))

(defun sum-mapcar (function-symbol &rest lists)
  '(apply #'+
    (mapcar ,function-symbol ,@lists)))
;; For loops

(defun n-times (n thing)
  (let ((result nil))
    (do ((i 0 result))
        ((= i n) result)
      (push thing result))))

(defun safe-string (in-string)
  (let* ((n (length in-string)))
    ;; For making strings that are valid identifiers
(out-string (make-string n :initial-element #\_)))
(dotimes (i n)
  (let ((curr-char (char in-string i)))
    (if (alphanumericp curr-char)
      (setf (char out-string i) curr-char)))))
out-string))

(defun very-safe-string (in-string)
  (let* ((n (length in-string))
    (out-string (make-string n :initial-element #\0)))
    (dotimes (i n)
      (let ((curr-char (char in-string i)))
        (if (alphanumericp curr-char)
          (setf (char out-string i) curr-char)))))
out-string))

(defun safe-first-label (in-list)
  (let ((state-n (first in-list)))
    (if (= state-n 0)
      "begin"
      state-n)))

(defun safe-last-label (in-list)
  (let ((state-n (first in-list)))
    (if (= state-n 0)
      "end"
      state-n)))

(defun safe-from-label (in-list)
  (let ((state-n (first in-list)))
    (if (= state-n 0)
      "begin"
      (write-to-string state-n))))

(defun safe-to-label (in-list)
  (let ((state-n (first in-list)))
    (if (= state-n 0)
      "end"
      (write-to-string state-n))))

(defun safe-input-label (input-label)
  (concatenate `'STRING
    "")

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(defun safe-output-label (output-label)
  (concatenate 'STRING
    "\""
    (write-to-string output-label)
    "\""))

(defun old-safe-list-label (in-list)
  (very-safe-string (string-trim '(#\( #\))
    (write-to-string in-list))))

(\(write-to-string input-label)
 "\")

(\(write-to-string input-label)
 "\")
;;;; ===============
;;;;  HMM-generic.lisp
;;;; ===============

(defun display (object &optional ostream))

(defun list-form (object))

(defun clone (object))

(defun save (object file-name))

(defun same-symbol (element-1 element-2))

(defun rename-element (old-name new-name element))

(defun add-element (element container))

(defun delete-element (element container))

(defun delete-element-with-symbol (symbol container))

(defun find-element-with-symbol (symbol container))

(defun find-a-duplicate (container))

(defun combine (element-1 element-2 container))

(defun merge-elements (container))

(defun update (container))

(defun update-after-merge (data-object
                         new-state-label
                         old-state-label))

(defun code-length (container)
n-states
n-input-symbols
n-output-symbols)

(defgeneric constraint (object))
;;; =================
;;; HMM-data.lisp
;;; =================

; --------------------------------------------------------------------------
; Data points

(defclass data-point ()
  ((input :reader input
           :initarg :input
           :initform nil)
   (output :reader output
            :initarg :output
            :initform nil)
   (comment :reader comment
             :initarg :comment
             :initform "")
   (state :accessor state
          :initarg :state
          :initform nil)))

(defmethod constraint ((the-data-point data-point))
  (constraint (output the-data-point)))

(defmethod display ((the-data-point data-point)
                    &optional (ostream t))
  (format ostream " &3a > ~16a : ~6a ; ~a~%"
            (input the-data-point)
            (output the-data-point)
            (state the-data-point)
            (comment the-data-point)))

(defmethod list-form ((the-data-point data-point))
  (list (input the-data-point)
        (output the-data-point)
        (comment the-data-point)
        (state the-data-point)))
(defun make-data-point (the-list-form)
  (make-instance 'data-point
    :input (first the-list-form)
    :output (second the-list-form)
    :comment (third the-list-form)
    :state (fourth the-list-form)))

(defun same-symbol ((data-point-1 data-point)
  (data-point-2 data-point)
  (and (equal (input data-point-1) (input data-point-2))
       (equal (output data-point-1) (output data-point-2))))

;------------------------------------------------------------------
; Data sequences

(defun display ((the-data-sequence data-sequence)
  (optional (ostream t))
  (format ostream "&---

  (dolist (item (data-points the-data-sequence))
    (display item ostream)
    (format ostream
      "&---"))

(defun list-form ((the-data-sequence data-sequence))
  (mapcar #'list-form
    (data-points the-data-sequence)))

(defun make-data-sequence (the-list-form)
  (make-instance 'data-sequence
    :size (length the-list-form)
    :data-points (mapcar #'make-data-point
    :state (fourth the-list-form)))

(defun same-symbol ((data-point-1 data-point)
  (data-point-2 data-point)
  (and (equal (input data-point-1) (input data-point-2))
       (equal (output data-point-1) (output data-point-2))))
the-list-form)))

; Data sets

(defclass data-set ()
  ((size :reader size
         :initarg :size)
   (data-sequences :reader data-sequences
              :initarg :data-sequences)))

(defun make-data-set (the-list-form)
  (make-instance 'data-set
    :size (length the-list-form)
    :data-sequences (mapcar #'make-data-sequence
                               the-list-form)))

(defun save ((the-data-set data-set) file-name)
  (with-open-file (ostream (concatenate 'string
                                    *data-root*
                                    file-name
                                    "_.data.lisp")
                        :direction :output)
    (format ostream ";; ~a.data.lisp~%"(quote " file-name)
    (pprint (list-form the-data-set))}
 APPENDIX B. COMPLETE CODE LISTINGS

 ostream) (format ostream "")~%"))

 (defun load-data-set (file-name) (with-open-file (istream (concatenate 'string *data-root* file-name ".data.lisp") :direction :input) (make-data-set (eval (read istream)))))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
;;; =========
;;;  HMM.lisp
;;; =========

;----------------------------------------------------------
; Parent class 'HMM-member'

(defun HMM-member ()
  ((data-count :accessor data-count
    :initarg :data-count
    :initform 0)))

;----------------------------------------------------------
; Parent class 'HMM-element'

(defun HMM-element (HMM-member)
  ((symbol :accessor symbol
    :initarg :symbol
    :initform nil)
   (probability :accessor probability
    :initarg :probability
    :initform 0.0)))

(defun same-symbol ((element-1 HMM-element)
  (element-2 HMM-element))
  (equal (symbol element-1) (symbol element-2)))

(defun rename-element
  (old-symbol new-symbol (the-element HMM-element))
  (if (equal (symbol the-element) old-symbol)
    (setf (symbol the-element) new-symbol)))

;----------------------------------------------------------
; Parent class 'HMM-container'

(defun HMM-container (HMM-member)
((elements :accessor elements
   :initarg :elements
   :initform nil)
 (entropy :accessor entropy
   :initarg :entropy
   :initform 0.0)))

(defmethod display :after
   ((the-container HMM-container) &optional (ostream t))
   (display-elements-in-list (elements the-container) ostream))

(defun display-elements-in-list (the-elements
   &optional (ostream t))
   (unless (null the-elements)
     (display-elements-in-list (rest the-elements) ostream)
     (display (first the-elements) ostream)))

;  ―― ..................................................................
;  Methods for adding and deleting

(defmethod add-element ((the-element HMM-element)
   (the-container HMM-container))
   (push the-element (elements the-container))
   the-element)

(defmethod delete-element ((the-element HMM-element)
   (the-container HMM-container))
   (setf (elements the-container)
     (remove the-element (elements the-container)
       :test #'eq)))

(defmethod delete-element-with-symbol
   (the-symbol
   (the-container HMM-container))
   (let ((the-element (find-element-with-symbol the-symbol
       the-container)))
     (when the-element
       (delete-element the-element the-container)))))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
;; Methods for finding elements

(defun find-in-list-element-with-symbol (the-symbol the-element-list)
  (unless (null the-element-list)
    (let* ((first-element (first the-element-list))
           (first-symbol (symbol first-element))
           (remaining-elements (rest the-element-list))
           (duplicate-element (find-in-list-element-with-symbol
                                first-symbol
                                remaining-elements)))
      (if duplicate-element
          (values first-symbol first-element duplicate-element)
          (find-a-duplicate-in-list remaining-elements)))))

(defun find-a-duplicate ((the-container HMM-container))
  (find-a-duplicate-in-list (elements the-container)))

(defun find-a-duplicate-in-list (the-element-list)
  (if (null the-element-list)
      (values nil nil nil nil)
      (let* ((first-element (first the-element-list))
              (first-symbol (symbol first-element))
              (remaining-elements (rest the-element-list))
              (duplicate-element (find-in-list-element-with-symbol
                                   first-symbol
                                   remaining-elements)))
       (if duplicate-element
           (values first-symbol first-element duplicate-element)
           (find-a-duplicate-in-list remaining-elements))))

;; Methods for updating
(defmethod update ((the-element HMM-element))
  the-element)

(defun update-elements-in-list (the-elements)
  (if (null the-elements)
      0
      (let ((first-element (first the-elements)))
        (update first-element)
        (+ (data-count first-element)
          (update-elements-in-list (rest the-elements)))))))

(defun update-element-probabilities-in-list
  (the-elements
   the-container-data-count)
  (unless (null the-elements)
    (let ((first-element (first the-elements)))
      (setf (probability first-element)
        (coerce (/ (data-count first-element)
                   the-container-data-count)
            'float))
      (update-element-probabilities-in-list
       (rest the-elements)
       the-container-data-count))))

; ------------------------------------------------------------------------
; Class 'HMM-output'

(defun HMM-output (HMM-element)
(defmethod display ((the-output HMM-output) &optional (ostream t))
  (format ostream ""& Output: "a"% c = "a", p = "a"%" (symbol the-output) (data-count the-output) (probability the-output)))

(defmethod list-form ((the-output HMM-output))
  (list (symbol the-output) (data-count the-output) (probability the-output)))

(defun make-HMM-output (the-list-form)
  (make-instance 'HMM-output :symbol (first the-list-form) :data-count (second the-list-form) :probability (third the-list-form)))

(defmethod clone ((the-HMM-output HMM-output))
  (make-instance 'HMM-output :symbol (symbol the-HMM-output) :data-count (data-count the-HMM-output) :probability (probability the-HMM-output)))

; ---------------------------------------------------------------------
; Class 'HMM-input'

(defclass HMM-input (HMM-container HMM-element) ()

(defmethod display ((the-input HMM-input) &optional (ostream t))
  (format ostream ""& Input: "a"% c = "a", p = "a"% entropy = "a"%" (symbol the-input) (data-count the-input) (probability the-input) (entropy the-input)))

(defmethod list-form ((the-input HMM-input))
  (list (symbol the-input)
(data-count the-input)
(probability the-input)
(entropy the-input)
(mapcar #'list-form
     (elements the-input)))

(defun make-HMM-input (the-list-form)
  (make-instance 'HMM-input
    :symbol (first the-list-form)
    :data-count (second the-list-form)
    :probability (third the-list-form)
    :entropy (fourth the-list-form)
    :elements (mapcar #'make-HMM-output
      (fifth the-list-form))))

(defun make-HMM-output (the-list-form)
  (make-instance 'HMM-output
    :symbol (first the-list-form)
    :data-count (second the-list-form)
    :probability (third the-list-form)
    :entropy (fourth the-list-form)
    :elements (mapcar #'make-HMM-output
      (fifth the-list-form))))

(defmethod clone ((the-HMM-input HMM-input))
  (make-instance 'HMM-input
    :symbol (symbol the-HMM-input)
    :data-count (data-count the-HMM-input)
    :probability (probability the-HMM-input)
    :entropy (entropy the-HMM-input)
    :elements (mapcar #'clone
      (elements the-HMM-input))))

; ----------------------------------------------------------------------
; Class 'HMM-transition'

(defclass HMM-transition (HMM-container HMM-element) ()

(defmethod display ((the-transition HMM-transition)
  &optional (ostream t))
  (format ostream
    "\~& to state: \~a\% c = \~a, p = \~a\% entropy = \~a\%"
    (symbol the-transition)
    (data-count the-transition)
    (probability the-transition)
    (entropy the-transition)))

(defmethod list-form ((the-transition HMM-transition))
  (list (symbol the-transition)
    (data-count the-transition)
(probability the-transition)
(entropy the-transition)
(mapcar #'list-form
  (elements the-transition)))

(defun make-HMM-transition (the-list-form)
  (make-instance 'HMM-transition
    :symbol (first the-list-form)
    :data-count (second the-list-form)
    :probability (third the-list-form)
    :entropy (fourth the-list-form)
    :elements (mapcar #'make-HMM-input
      (fifth the-list-form))))

(defun clone ((the-HMM-transition HMM-transition))
  (make-instance 'HMM-transition
    :symbol (symbol the-HMM-transition)
    :data-count (data-count the-HMM-transition)
    :probability (probability the-HMM-transition)
    :entropy (entropy the-HMM-transition)
    :elements (mapcar #'clone
      (elements the-HMM-transition))))

; Class 'HMM-state'

(defun class HMM-state (HMM-container HMM-element)
  ()

(defun method constraint ((the-state HMM-state))
  (constraint (symbol the-state)))

(defun method constraint ((the-list cons))
  (first (last the-list)))

(defun method constraint ((other T))
  NIL)

(defun method display ((the-state HMM-state) &optional (ostream t))
  (format ostream "\"&State: \"a\"%c = \"a\", p = \"a\"%entropy = \"a\"\"
    (symbol the-state))
(data-count the-state)
(probability the-state)
(entropy the-state))

(defmethod list-form ((the-state HMM-state))
  (list (symbol the-state)
    (data-count the-state)
    (probability the-state)
    (entropy the-state)
    (mapcar #'list-form
      (elements the-state))))

(defun make-HMM-state (the-list-form)
  (make-instance 'HMM-state
    :symbol (first the-list-form)
    :data-count (second the-list-form)
    :probability (third the-list-form)
    :entropy (fourth the-list-form)
    :elements (mapcar #'make-HMM-transition
      (fifth the-list-form))))

(defun clone ((the-HMM-state HMM-state))
  (make-instance 'HMM-state
    :symbol (symbol the-HMM-state)
    :data-count (data-count the-HMM-state)
    :probability (probability the-HMM-state)
    :entropy (entropy the-HMM-state)
    :elements (mapcar #'clone
      (elements the-HMM-state))))

; ------------------------------------------------------------------------
; Class 'HMM'

(defunclass HMM (HMM-container)
  ((name :reader name
    :initarg :name
    :initform "")
   (n-states :accessor n-states
     :initarg :n-states
     :initform 0)
   (n-input-symbols :reader n-input-symbols
     :initarg :n-input-symbols
     :initform 0))
(defmethod display ((the-model HMM) &optional (ostream t))
  (format ostream "&~%------------------------~%")
  (format ostream "& HMM ~a~% ~a states~% entropy = ~a~%"
    (name the-model)
    (n-states the-model)
    (entropy the-model))

  (format ostream "~a input symbol(s)~% ~a output symbol(s)~% c = ~a~%"
    (n-input-symbols the-model)
    (n-output-symbols the-model)
    (data-count the-model))

  (format ostream "------------------------~%"))

(defun make-HMM (the-list-form)
  (make-instance 'HMM
    :name (first the-list-form)
    :data-count (second the-list-form)
    :n-states (third the-list-form)
    :n-input-symbols (fourth the-list-form)
    :n-output-symbols (fifth the-list-form)
    :entropy (sixth the-list-form)
    :elements (mapcar #'make-HMM-state
      (seventh the-list-form))))

(defun clone ((the-HMM HMM))
  (make-instance 'HMM
    :name (name the-HMM)
    :data-count (data-count the-HMM)
(defmethod save ((the-model HMM) file-name)
  (with-open-file (ostream (concatenate 'string
    *data-root* file-name ".hmm.lisp")
    :direction :output)
    (format ostream ";;;; "a.hmm.lisp""%"%(quote " file-name)
      (pprint (list-form the-model) ostream)
      (format ostream ")"%"))))

(defun load-HMM (file-name)
  (with-open-file (istream (concatenate 'string
    *data-root* file-name ".hmm.lisp")
    :direction :input)
    (make-HMM (eval (read istream))))))

(defmethod combine :after
  (element-1 element-2 (the-HMM HMM))
  (declare (ignore element-1)
    (ignore element-2))
  (setf- (n-states the-HMM) 1))
;;; == = s = ==
;;; HMM-coding.lisp
;;; == = s = ==

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Basic utilities

(defun int-2-to-the (x)
  (values
   (exp (* x (log 2)))))

(defun integer-description-bits (n)
  (i+ (truncate (log n 2))))

(defun set-member-description-bits (set-size)
  (values
   (log set-size 2))

(defun integer-code-length (n)
  (when (and (integerp n) (<= 0 n))
    (cond ((= n 0) 1)
      (t (do* ((i (i+ n)
        (1- i-bits))
        (i-bits (integer-description-bits i)
          (integer-description-bits i))
        (result (i+ i-bits)
          (+ result i-bits))
        ((<= i-bits 2) result))))))

(defun sum-encoded-code-length (sum list-length)
  (combinatorial-code-length (+ sum list-length -1)
    (- list-length 1)))

(defun combinatorial-code-length (n m)
  (let ((m-best (min m (- n m)))
    (log-num 0.0)
    (log-den 0.0))
  (do (i m-best)
    (setq log-num (+ log-num (log (- (+ n 1 i) m-best) 2)))
    (setq log-den (+ log-den (log (- (+ n 1 i) m-best) 2))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX B. COMPLETE CODE LISTINGS

log-den (+ log-den (log (1+ i) 2)))
(values (ceiling (- log-num log-den))))

; HMM coding functions

(defun code-length ((the-element HMM-element)
  n-states
  n-input-symbols
  n-output-symbols)
  (declare (ignore the-element)
    (ignore n-states)
    (ignore n-input-symbols)
    (ignore n-output-symbols))
  0)

(defun code-length ((the-container HMM-container)
  n-states
  n-input-symbols
  n-output-symbols)
  (+ (sum-encoded-code-length
      (data-count the-container)
      (case (class-name (class-of the-container))
        (HMM-state n-states)
        (HMM-transition n-input-symbols)
        (HMM-input n-output-symbols)))
    (sum-mapcar #'(lambda (x)
                  (code-length x
                  n-states
                  n-input-symbols
                  n-output-symbols))
                (elements the-container))))

(defun model-code-length (the-model)
  (let ((data-count (data-count the-model))
    (n-states (n-states the-model))
    (n-input-symbols (n-input-symbols the-model))
    (n-output-symbols (n-output-symbols the-model)))
    (+ (integer-code-length n-states)
        (integer-code-length data-count)
        (integer-code-length n-input-symbols)
        (integer-code-length n-output-symbols))
(sum-encoded-code-length data-count n-states)
(sum-mapcar #'(lambda (x)
       (code-length x
       n-states
       n-input-symbols
       n-output-symbols))
       (elements the-model)))))
APPENDIX B. COMPLETE CODE LISTINGS

;;; =======================
;;;  HMM-data-coding.lisp
;;; =======================

(defun data-code-length (the-data-set
  the-model
  &optional
  new-state-label
  old-state-label)
  (values
    (ceiling (data-set-likelihood the-data-set
      the-model
      new-state-label
      old-state-label))))

#| (defun data-code-length (the-data-set
  the-model
  &optional
  new-state-label
  old-state-label)
  (data-set-likelihood the-data-set the-model
    new-state-label old-state-label)) |

;  -----------------------------------------------
;  Calculating - log p( i, o | M )

(defun data-set-likelihood (the-data-set
  the-model
  new-state-label old-state-label)
  (data-set-likelihood-from-list (data-sequences the-data-set
    the-model
    new-state-label
    old-state-label))

(defun data-set-likelihood-from-list (the-sequence-list
  the-model
  new-state-label
  ...)

(defun data-set-likelihood (the-data-set
  the-model
  new-state-label old-state-label)
  (data-set-likelihood-from-list (data-sequences the-data-set
    the-model
    new-state-label
    old-state-label))
APPENDIX B. COMPLETE CODE LISTINGS

(old-state-label)

(if (null the-sequence-list)
  0
  (+ (data-sequence-likelihood (first the-sequence-list)
       the-model
       new-state-label
       old-state-label)
      (data-set-likelihood-from-list (rest the-sequence-list)
       the-model
       new-state-label
       old-state-label))))

(defun data-sequence-likelihood (the-data-sequence
       the-model
       new-state-label old-state-label)
  (data-sequence-likelihood-from-list
   (data-points the-data-sequence)
   '(0)
   the-model
   new-state-label
   old-state-label))

(defun data-sequence-likelihood-from-list
  (the-points-list
   the-state
   the-model
   new-state-label
   old-state-label)
  (if (null the-points-list)
      (get-end-likelihood the-state
       the-model
       new-state-label old-state-label)
      (let* ((the-point (first the-points-list))
        (the-new-state (state the-point)))
        (+ (get-output-likelihood the-point
            the-state
            the-new-state
            the-model
            new-state-label old-state-label)
       (data-sequence-likelihood-from-list
        (rest the-points-list)))
     (get-end-likelihood the-state
      the-model
      new-state-label old-state-label))
  (null the-points-list))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(defun get-output-likelihood (the-point
  the-from-state-symbol
  the-to-state-symbol
  the-model
  new-state-label old-state-label)
  (let* ((the-from-state
    (find-element-with-symbol (replace-if-necessary
      the-from-state-symbol
      new-state-label
      old-state-label)
      the-model))
    (the-to-state
      (find-element-with-symbol (replace-if-necessary
       the-to-state-symbol
       new-state-label
       old-state-label)
       the-from-state))
    (the-input (find-element-with-symbol (input the-point)
      the-to-state))
    (the-output (find-element-with-symbol (output the-point)
      the-input)))
  (+ (- (log (probability the-to-state) 2))
    (- (log (probability the-input) 2))
    (- (log (probability the-output) 2)))))

(defun get-end-likelihood (the-from-state-symbol
  the-model
  new-state-label old-state-label)
  (let* ((the-from-state
    (find-element-with-symbol (replace-if-necessary
      the-from-state-symbol
      new-state-label
      old-state-label)
      the-model))
    (the-to-state
      (find-element-with-symbol '(0) the-from-state)))
    (- (log (probability the-to-state) 2)))))
; ------------------------------
; Total code length

(defun total-code-length (the-data the-model
  &optional
    new-state-label
    old-state-label)
  (+ (model-code-length the-model)
    (data-code-length the-data the-model
    new-state-label old-state-label)))

; ------------------------------
; Renaming states without changing the data set

(defun replace-if-necessary (state-label new-state-label
  old-state-label)
  (if (equal state-label old-state-label)
    new-state-label
    state-label))

; ------------------------------
; Permanently renaming states after a merge

(defmethod update-after-merge ((the-data-set data-set)
  new-state-label old-state-label)
  (update-data-list-after-merge (data-sequences the-data-set)
  new-state-label old-state-label))

(defmethod update-after-merge ((the-data-sequence data-sequence)
  new-state-label old-state-label)
  (update-data-list-after-merge (data-points the-data-sequence)
  new-state-label old-state-label))

(defmethod update-after-merge ((the-data-point data-point)
new-state-label old-state-label)
(setf (state the-data-point)
  (replace-if-necessary (state the-data-point)
    new-state-label old-state-label)))

(defun update-data-list-after-merge (the-data-list
  new-state-label
  old-state-label)
  (unless (null the-data-list)
    (update-after-merge (first the-data-list)
      new-state-label old-state-label)
    (update-data-list-after-merge (rest the-data-list)
      new-state-label
      old-state-label)))
APPENDIX B. COMPLETE CODE LISTINGS

;; ; ===============
;; ;  SO-training.lisp
;; ; ===============

; Global variables

(defvar *data-set*)
(defvar *model*)
(defvar *model-length*)
(defvar *data-length*)
(defvar *total-length*)
(defvar *best-state-1*)
(defvar *best-state-2*)
(defvar *best-trial-model*)
(defvar *best-data-length-so-far*)
(defvar *best-total-length-so-far*)
(defvar *phase-1*)
(defvar *phase-2*)
(defvar *phase-3*)

; Function ‘make-SO-model’

(defun make-SO-model (the-data-set &optional (name "---"))
  (let* (((begin-state (make-instance 'HMM-state
                                 :symbol '(0)))
           (n-states 1)
           (input-symbols '(:END))
           (output-symbols '(:END))
           (states '(:begin-state)))
    (dolist (the-data-sequence (data-sequences the-data-set))
      (let ((from-state begin-state)
            (to-state nil))
        (dolist (the-data-point (data-points the-data-sequence))
          (setf to-state
                (make-instance 'HMM-state
                               :symbol
                               '(:n-states ,n-states - ,(constraint the-data-point))))
          (setf+ n-states 1)
          )))
  )

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(push (input the-data-point) input-symbols)
(push (output the-data-point) output-symbols)
(push to-state states)
(setf (state the-data-point) (symbol to-state))
(add-element (make-instance 'HMM-transition
  :symbol (symbol to-state)
  :elements
  '(_, (make-instance 'HMM-input
    :symbol (input the-data-point)
    :elements
    '(_, (make-instance 'HMM-output
      :symbol
      (output the-data-point)
      :data-count 1))))))

  from-state)
(setf from-state to-state))
(add-element (make-instance 'HMM-transition
  :symbol '(0)
  :elements
  '(_, (make-instance 'HMM-input
    :symbol :END
    :elements
    '(_, (make-instance 'HMM-output
      :symbol :END
      :data-count 1))))))
to-state)))
(setq input-symbols (remove-duplicates input-symbols
  :test #'equal))
(setq output-symbols (remove-duplicates output-symbols
  :test #'equal))
(update (make-instance 'HMM
  :name name
  :n-states n-states
  :n-input-symbols (length input-symbols)
  :n-output-symbols (length output-symbols)
  :elements states)))

; ------------------------------------------------------------------------------------------------
; Functions for merging

(defmethod combine ((element-1 HMM-element)
(element-2 HMM-element)
  (the-container HMM-container))
(delete-element element-1 the-container)
(delete-element element-2 the-container)
(add-element (make-instance (class-name (class-of element-1))
  :symbol (symbol element-1)
  :data-count (+ (data-count element-1)
                   (data-count element-2)))
  the-container))

(defmethod combine ((element-1 HMM-container)
                     (element-2 HMM-container)
                     (the-container HMM-container))
  (delete-element element-1 the-container)
  (delete-element element-2 the-container)
  (add-element (make-instance (class-name (class-of element-1))
                             :symbol (symbol element-1)
                             :elements (append (elements element-1)
                                               (elements element-2)))
                             the-container))

(defmethod merge-elements ((the-element HMM-element))
  (declare (ignore the-element)))

(defmethod merge-elements ((the-container HMM-container))
  (multiple-value-bind (there-are-duplicates the-element
                        the-duplicate)
                        (find-a-duplicate the-container)
                        (when there-are-duplicates
                          (merge-elements (combine the-element the-duplicate
                                           the-container)))
                          (merge-elements the-container))))

; ----------------------------------------------------------------------
; User interface
; ----------------------------------------------------------------------

; Functions for creating, saving, and maintaining consistent state

(defun open-data-set (data-name model-name)
(declare (special *data-set*)
  (special *model*))
(setq *data-set* (load-data-set data-name)
  *model* (make-SO-model *data-set* model-name))
(update-code-lengths)
(show-code-lengths))

(defun save-data-set (data-name model-name)
  (declare (special *data-set*)
    (special *model*))
  (save *data-set* data-name)
  (save *model* model-name))

(defun update-code-lengths ()
  (declare (special *data-set*)
    (special *model*)
    (special *model-length*)
    (special *data-length*)
    (special *total-length*))
  (setq *model-length* (model-code-length *model*)
    *data-length* (data-code-length *data-set* *model*)
    *total-length* (+ *model-length* *data-length*)))

;; Functions for displaying

(defun show-data-set ()
  (declare (special *data-set*))
  (display *data-set*))

(defun show-model ()
  (declare (special *model*))
  (display *model*))

(defun show-code-lengths ()
  (declare (special *model-length*)
    (special *data-length*)
    (special *total-length*))
  (format t "~&Model length: "a~%'
    Total length: "a~%~
    *model-length* *data-length* *total-length*))
; Function ‘try-merge’

(defun try-merge (state-symbol-1 state-symbol-2)
  (declare (special *model*))
  (let* ((the-trial-model (clone *model*))
         (state-1 (find-element-with-symbol state-symbol-1
                   the-trial-model))
         (state-2 (find-element-with-symbol state-symbol-2
                   the-trial-model)))
    (combine state-1 state-2 the-trial-model)
    (dolist (state-i (elements the-trial-model))
      (dolist (transition-j (elements state-i))
        (rename-element state-symbol-2 state-symbol-1
                         transition-j))
        (merge-elements state-i))
    (update the-trial-model)))

; Function ‘suggest-merge-with-min-data-length’

(defun suggest-merge-with-min-data-length ()
  (declare (special *data-set*)
    (special *best-state-1*)
    (special *best-state-2*)
    (special *best-trial-model*)
    (special *best-data-length-so-far*))
  (let ((the-states (remove '(0) (mapcar #'symbol
                  (elements *model*))
                  :test #'equal))
        (setq *best-state-1* nil
               *best-state-2* nil
               *best-trial-model* nil
               *best-data-length-so-far* 1.0E+30)
    (do ((trial-state-2 (pop the-states) (pop the-states)))
        (null the-states)
      (dolist (trial-state-1 the-states)
        (when (equal (constraint trial-state-1)
                     (constraint trial-state-2))
          ;;(format t ""&Trying ~a ~a~%" trial-state-1
                      trial-state-2))

(let* ((trial-model (try-merge trial-state-1 trial-state-2))
(trial-data-code-length
 (data-code-length *data-set*
  trial-model
  trial-state-1
  trial-state-2)))
 (when (< trial-data-code-length
   *best-data-length-so-far*)
 (setq *best-state-1* trial-state-1
   *best-state-2* trial-state-2
   *best-trial-model* trial-model
   *best-data-length-so-far*
   trial-data-code-length)))))))
(format t "\"Suggested merge: \"a ~a\"% Data length: \"a-%\"
   *best-state-1*
   *best-state-2*
   *best-data-length-so-far*)))

; Function ‘suggest-merge-with-min-total-length’

(defun suggest-merge-with-min-total-length ()
 (declare (special *data-set*)
  (special *best-state-1*)
  (special *best-state-2*)
  (special *best-trial-model*)
  (special *best-total-length-so-far*))
 (let ((the-states (remove '() (mapcar #\'symbol
                                      (elements *model*))
                                       :test #\'equal)))
 (setq *best-state-1* nil
   *best-state-2* nil
   *best-trial-model* nil
   *best-total-length-so-far* 1.0E+30)
 (do (((trial-state-2 (pop the-states) (pop the-states)))
   ((null the-states))
 (dolist (trial-state-1 the-states)
 (when (equal (constraint trial-state-1)
               (constraint trial-state-2))
   (format t "\"&Trying \"a ~a\"% trial-state-1 trial-state-2)\n)
(let* ((trial-model (try-merge trial-state-1 trial-state-2))
        (trial-total-code-length (total-code-length *data-set*
                                   trial-model
                                   trial-state-1
                                   trial-state-2)))
  (when (< trial-total-code-length *best-total-length-so-far*)
    (setq *best-state-1* trial-state-1
          *best-state-2* trial-state-2
          *best-trial-model* trial-model
          *best-total-length-so-far* trial-total-code-length))))
(format t ""&Suggested merge: "a" Total length: "a"" *best-state-1*
        *best-state-2*
        *best-total-length-so-far*))

; ------------------------------------------------------------------------
; Function `do-merge'

(defun do-merge (&optional (states nil))
  (declare (special *data-set*)
           (special *model*)
           (special *best-state-1*)
           (special *best-state-2*)
           (special *best-trial-model*)
  (when states
    (setq *best-state-1* (first states)
          *best-state-2* (second states)
          *best-trial-model* (try-merge *best-state-1* *best-state-2*))
    (setq *model* *best-trial-model*)
    (update-after-merge *data-set* *best-state-1* *best-state-2*)
    (format t ""&Kept merge "a" "a""
            *best-state-1* *best-state-2*)
    (update-code-lengths)
    (log-merge)
    (show-code-lengths))
; -----------------------------------------------
; Function 'do-trial-merge'

(defun do-trial-merge (states)
  (declare (special *data-set*)
    (special *best-state-1*)
    (special *best-state-2*)
    (special *best-trial-model*))
  (setq *best-state-1* (first states)
        *best-state-2* (second states)
        *best-trial-model* (try-merge *best-state-1*
                                *best-state-2*))
  (let* ((model-length (model-code-length *best-trial-model*))
         (data-length (data-code-length *data-set*
                              *best-trial-model*
                              *best-state-1*
                              *best-state-2*))
         (total-length (+ model-length data-length)))
    (format t "&Model length: ~a Data length: ~a~%"~
            ~a ~%~
            model-length data-length total-length)))

; -----------------------------------------------
; Function 'phase-1-training'

(defun phase-1-training ()
  (declare (special *data-length*)
    (special *best-data-length-so-far*)
    (special *phase-1*)
  (setq *phase-1* T)
  (do ()
    ((not *phase-1*))
    (suggest-merge-with-min-data-length)
    (if (<= *best-data-length-so-far* *data-length*)
      (do-merge)
      (setq *phase-1* nil))))

; -----------------------------------------------
; Function 'phase-2-training'
(defun phase-2-training ()
  (declare (special *total-length*)
           (special *best-total-length-so-far*)
           (special *phase-2*))
  (setq *phase-2* T)
  (do ()
      ((not *phase-2*))
      (phase-1-training)
      (suggest-merge-with-min-total-length)
      (if (<= *best-total-length-so-far* *total-length*)
          (do-merge)
          (setq *phase-2* nil)))))

; -----------------------------------------------------------------------
; Function 'phase-3-training'

(defun phase-3-training ()
  (declare (special *best-total-length-so-far*)
           (special *phase-3*))
  (setq *phase-3* T)
  (do ()
      ((not *phase-3*))
      (phase-1-training)
      (suggest-merge-with-min-total-length)
      (if (<= 1.0E+30 *best-total-length-so-far*)
          (do-merge)
          (setq *phase-3* nil)))))

; -----------------------------------------------------------------------
; Functions 'log-merge' and 'log-nl'

(defun log-merge ()
  (declare (special *training-root*)
           (special *model*)
           (special *best-state-1*)
           (special *best-state-2*)
           (special *model-length*)
           (special *data-length*)
           (special *total-length*))
  (with-open-file (log-file (concatenate 'STRING

APPENDIX B. COMPLETE CODE LISTINGS

*training-root*
"log:"
(name *model*)
".log.lisp")
direction :output
;if-does-not-exist :create
;if-exists :append)
(format log-file "~&do-merge '(~a ~a)) ; ~a : ~a : ~a~%"
*best-state-1*
*best-state-2*
*total-length*
*model-length*
*data-length*)))

(defun log-nl ()
  (declare (special *training-root*)
    (special *model*))
  (with-open-file (log-file (concatenate 'STRING
    *training-root* 
    "log:" 
    (name *model*) 
    ".log.lisp")
    :direction :output 
    :if-does-not-exist :create
    :if-exists :append)
  (format log-file "~%")))
(defmethod save-as-graph ((the-model HMM) file-name)
  (with-open-file (ostream (concatenate 'string
                                *data-root*
                                file-name
                                ".dot")
               :direction :output)
    (format ostream "digraph ~a {"% rankdir=LR,"%~"%" (safe-string file-name))
    (to-graph the-model ostream)
    (format ostream ""&"%~"%"))
)

(defmethod to-graph ((the-model HMM) (ostream t))
  (dolist (current-state (reverse (elements the-model)))
    (to-graph-aux current-state ostream))
  (format ostream ""%)\n  (dolist (current-state (reverse (elements the-model)))
    (to-graph current-state ostream)
    (format ostream ""%"))
)

(defmethod to-graph-aux ((the-state HMM-state) (ostream t))
  (let ((state-label (safe-first-label (symbol the-state))))
    (format ostream ""& node ~a;"% state-label"))
)

(defmethod to-graph ((the-state HMM-state) (ostream t))
  (let ((state-label (safe-first-label (symbol the-state))))
    (dolist (current-transition (reverse (elements the-state)))
      (format ostream ""& ~a -> " state-label)
      (to-graph current-transition ostream))
  )
)

(defmethod to-graph ((the-transition HMM-transition) (ostream t))
  (format ostream "~a [ label = \"~a\" ];"% (safe-last-label (symbol the-transition))
            (transition-label (reverse (elements the-transition))))
)
(defun to-transition-label (input-list)
  (let ((result ""))
    (dolist (curr-input input-list result)
      (setq result
        (concatenate 'STRING
                      result
                      (to-input-label curr-input))))))

(defun to-input-label (the-input)
  (let ((outputs (reverse (elements the-input)))
         (result ""))
    (dolist (curr-output outputs result)
      (setq result
        (concatenate 'STRING
                      result
                      (write-to-string (symbol the-input))
                      " -\rightarrow "
                      (write-to-string (symbol curr-output))
                      "\n"))))))

#|

(setq test-hmm
  (make-SO-model
   (make-data-set '(((a) (b))
                   ((a) (b) (a) (b))
                   ((a) (b) (a) (b) (a) (b))))))

(save-as-graph test-hmm "test-graph-1")

|#
;;; ===================
;;; HMM-to-prolog.lisp
;;; ===================

(defun save-in-prolog ((the-model HMM) file-name)
  (with-open-file (ostream (concatenate 'string
                                *data-root*
                                file-name
                                ".pro")
                     :direction :output)
    (format ostream "%%%" file-name)
    (to-prolog the-model ";" 1.0 ostream)
    (format ostream "%%%")))

(defun to-prolog ((the-model HMM) string-so-far prob-so-far
                  (ostream t))
  (dolist (current-state (reverse (elements the-model)))
    (to-prolog current-state string-so-far prob-so-far ostream)
    (format ostream "\n")))

(defun to-prolog ((the-state HMM-state) string-so-far
                  prob-so-far
                  (ostream t))
  (let* ((state-label (safe-from-label (symbol the-state)))
         (new-string-so-far (concatenate 'STRING
                                           string-so-far state-label ",")))
    (dolist (current-transition (reverse (elements the-state)))
      (to-prolog current-transition new-string-so-far prob-so-far
                   ostream))))

(defun to-prolog ((the-transition HMM-transition)
                  string-so-far
                  prob-so-far (ostream t))
  (let* ((transition-label
           (safe-to-label (symbol the-transition)))
         (transition-prob (probability the-transition))
         (new-string-so-far (concatenate 'STRING
                                         string-so-far
                                         " "
                                         transition-label
                                         transition-prob))
         (new-string-so-far
          (append string-so-far
                   (list transition-label
                           transition-prob)))))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
transition-label
";
))
(new-prob-so-far (* prob-so-far transition-prob))
(dolist (current-input (reverse (elements the-transition)))
  (to-prolog current-input new-string-so-far new-prob-so-far
   ostream))))

(defmethod to-prolog ((the-input HMM-input) string-so-far
  prob-so-far
  (ostream t))
(let* ((input-label (safe-input-label (symbol the-input)))
  (input-prob (probability the-input))
  (new-string-so-far (concatenate 'STRING
    string-so-far
    " [ ":
    input-label
    ",")
  )
  )
(new-prob-so-far (* prob-so-far input-prob))
(dolist (current-output (reverse (elements the-input)))
  (to-prolog current-output new-string-so-far new-prob-so-far
    ostream))))

(defmethod to-prolog ((the-output HMM-output) string-so-far
  prob-so-far
  (ostream t))
(let* ((output-label (safe-output-label (symbol the-output)))
  (output-prob (probability the-output))
  (new-prob-so-far (* prob-so-far output-prob))
  (format ostream "\&transition( "a "a ], "6,4f )."%"
    string-so-far
    output-label
    new-prob-so-far))))

#| (setq test-hmm
  (make-SO-model
    (make-data-set '((((- a) (- b))
      (((- a) (- b) (- a) (- b))

((- a) (- b) (- a) (- b) (- a) (- b))))

(save-in-prolog test-hmm "test-graph-001")

(setq test-hmm (load-HMM "2004.10:Model_014a"))
(save-in-prolog test-hmm "2004.10:Model_014a")

|#
\[ \begin{array}{l}
\text{PATH} \text{( Q1, QN, [ [ Syll, Stress ] | Rem_input ], [ s( Q1 ) ], [ Syll, Stress, Intvl ] | Rem_path ], Prob_path, Depth ) :-}
\quad \text{Depth } \geq 0, \\
\quad \text{Depth } > 0, \\
\quad \text{New_depth } = \text{Depth } - 1, \\
\quad \text{PATH} \text{( Q2, QN, [ [ - , '0' ] | Rem_input ], Rem_path, Prob_rem, New_depth ),}
\quad \text{TRANSITION} \text{( Q1, Q2, [ Stress, Intvl ] , Prob_trans ),}
\quad \text{Prob_path } = \text{Prob_trans } * \text{Prob_rem.}
\end{array} \]

\[ \begin{array}{l}
\text{PATH} \text{( Q1, QN, [ [ - , '0' ] | Rem_input ], Path, Prob_path, Depth ) :-}
\quad \text{PATH} \text{( Q1, QN, Rem_input, Path, Prob_path, Depth ).}
\end{array} \]

\[ \begin{array}{l}
\text{PRINT\_SOLUTION} \text{( Solution ) :-}
\quad \text{PRINT\_NEXT\_NOTE} \text{([ [-,[-] | Solution ] ).}
\end{array} \]

\[ \begin{array}{l}
\text{PRINT\_NEXT\_NOTE} \text{([ ] ).}
\end{array} \]

\[ \begin{array}{l}
\text{PRINT\_NEXT\_NOTE} \text{([ Note, State | Rem_solution ] ) :-}
\quad \text{WRITE} \text{( Note ), write( ' : ' ), write( State ), nl,}
\quad \text{PRINT\_NEXT\_NOTE} \text{Rem_solution ).}
\end{array} \]
Appendix C

A Complete Sample Model of *Echos* 1

This appendix presents complete HMM graphs for the sample model used in Chapter 5. In these graphs, the transition outputs are labeled as follows:

\[
\text{Stress} \rightarrow \text{Duration Interval ( LetterName DiatonicN ChromaticN )}
\]

where the variable names *Stress*, *Duration*, etc. are explained in p. 101.

Table C.1 shows a sample chant from the corpus used to obtain the present model. The chant is aligned with the states of the most likely HMM path that generated it.
Figure C.1: The structure of the HMM modeling Echos 1, based on a sample corpus of fourteen chants. Opening and cadential formulas are identified as subgraphs and labeled OF\_x and CF\_x respectively. The central part of the graph corresponds to the phrase body. This Figure reproduces Fig. 5.1.
Figure C.2: Subgraph of Figure C.1 corresponding to the opening formula OF..01. This Figure reproduces Fig. 5.2 with more detailed output labels.
Figure C.3: Subgraph of Figure C.1 corresponding to the cadential formula CF.G01.
Figure C.4: Subgraph of Figure C.1 corresponding to the cadential formula CF.G02.
Figure C.5: Subgraph of Figure C.1 corresponding to the cadential formula CF.G03.
Figure C.6: Subgraph of Figure C.1 corresponding to the cadential formula CF.G04.
Figure C.7: Subgraph of Figure C.1 corresponding to the cadential formula CF_D01. This Figure reproduces Fig. 5.3 with more detailed output labels.
Figure C.8: Subgraph of Figure C.1 corresponding to the phrase body.
<table>
<thead>
<tr>
<th>Text</th>
<th>Word Stress</th>
<th>Duration</th>
<th>Diatonic Interval</th>
<th>Letter Name</th>
<th>Diatonic Pitch #</th>
<th>Chromatic Pitch #</th>
<th>States</th>
<th>Formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>-3</td>
<td>D</td>
<td>8</td>
<td>42</td>
<td>(703 - (D 8 42))</td>
<td>OF_01</td>
</tr>
<tr>
<td>gi</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>E</td>
<td>9</td>
<td>47</td>
<td>(705 - (E 9 47))</td>
<td>CF.G01</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>A</td>
<td>12</td>
<td>63</td>
<td>(742 - (A 12 63))</td>
<td>CF.D01</td>
</tr>
<tr>
<td>ma</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>A</td>
<td>12</td>
<td>57</td>
<td>(741 - (A 12 57))</td>
<td>CF.D02</td>
</tr>
<tr>
<td>ti</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>B</td>
<td>13</td>
<td>68</td>
<td>(742 - (B 13 68))</td>
<td>CF.D03</td>
</tr>
<tr>
<td>mi</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>A</td>
<td>12</td>
<td>63</td>
<td>(742 - (A 12 63))</td>
<td>CF.D04</td>
</tr>
<tr>
<td>kai</td>
<td>1</td>
<td>4</td>
<td>-2</td>
<td>A</td>
<td>12</td>
<td>57</td>
<td>(744 - (A 12 57))</td>
<td>CF.D05</td>
</tr>
<tr>
<td>do</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>A</td>
<td>12</td>
<td>57</td>
<td>(744 - (A 12 57))</td>
<td>CF.D06</td>
</tr>
<tr>
<td>xa</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>A</td>
<td>12</td>
<td>57</td>
<td>(744 - (A 12 57))</td>
<td>CF.D07</td>
</tr>
<tr>
<td>ox</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D08</td>
</tr>
<tr>
<td>per</td>
<td>1</td>
<td>4</td>
<td>-2</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D09</td>
</tr>
<tr>
<td>Pa</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D10</td>
</tr>
<tr>
<td>tri</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D11</td>
</tr>
<tr>
<td>pre</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D12</td>
</tr>
<tr>
<td>pei</td>
<td>1</td>
<td>8</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D13</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D14</td>
</tr>
<tr>
<td>ma</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D15</td>
</tr>
<tr>
<td>kai</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D16</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D17</td>
</tr>
<tr>
<td>o;</td>
<td>2</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D18</td>
</tr>
<tr>
<td>di</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D19</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D20</td>
</tr>
<tr>
<td>tou</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D21</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D22</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D23</td>
</tr>
<tr>
<td>so</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D24</td>
</tr>
<tr>
<td>men</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D25</td>
</tr>
<tr>
<td>ti</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D26</td>
</tr>
<tr>
<td>Tri</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D27</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D28</td>
</tr>
<tr>
<td>di</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D29</td>
</tr>
<tr>
<td>mo</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D30</td>
</tr>
<tr>
<td>no</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D31</td>
</tr>
<tr>
<td>kra</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D32</td>
</tr>
<tr>
<td>to</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D33</td>
</tr>
<tr>
<td>ri</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>F</td>
<td>10</td>
<td>51</td>
<td>(744 - (F 10 51))</td>
<td>CF.D34</td>
</tr>
<tr>
<td>an.</td>
<td>1</td>
<td>4</td>
<td>-2</td>
<td>D</td>
<td>8</td>
<td>42</td>
<td>(53 - (D 8 42))</td>
<td>OF_01</td>
</tr>
</tbody>
</table>

Table C.1: Sample chant from the corpus used to obtain the model of Figures C.1–C.8. The chant is aligned with the states of the most likely HMM path that generated it. The variable values appearing in the columns are explained on p. 101. A state label is formed by a state ID integer followed by a triplet representing the state’s pitch.