Modelling Hierarchical Musical Structures With Composite Probabilistic Networks

Michèle Weiland

Doctor of Philosophy
Music
School of Arts, Culture and Environment
University of Edinburgh
2007

Acknowledgements

I would like to thank my supervisors, Professor Peter Nelson and Dr Alan Smaill, for their invaluable input and ideas throughout the course of this research.

I would also like to thank the Luxembourg Ministry of Culture, Higher Education and Research, and the Chevening Programme of the British Council for making this research possible through their scholarships.

Finally, I would like to express my gratitude to my family, who are always extremely supportive. Thanks also to all my friends, especially Al, Bob, Leana, Robert and Steve, who were a constant source of encouragement over the years. Last but not least, the biggest 'thank you' has to go to Tree for being a great office mate, proof reader and friend.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Michèle Weiland*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Learning and generating musical structures computationally is a challenging task that has taken the interest of several research projects since the start of modern computation. This research aims at learning large-scale structures of musical properties with probabilistic models, and thus generating believable musical material.

Many different approaches have been used for the generation of musical structures, some of them trying to find and use parallels between music and natural language (Cope, 1991), representing musical structures with stochastic grammars (Bod, 2002), others building expert systems whose rules describe musical processes as closely as possible (Ebcioglu, 1993; Spangler, 1999), and some even using genetic algorithms (Burton & Vladimirova, 1999). Probabilistic models have in the past been used successfully in applications such as speech recognition (Rabiner, 1989), natural language processing (Jelinek, 1985) or the modelling of biological sequences (Bengio, 1999), but they have also been used in musical applications, addressing counterpoint or harmonisation problems (Farbood & Schoner, 2001; Allan & Williams, n.d.). Following this same idea, our research uses stochastic processes to learn large-scale musical structures from existing musical data sets. Unlike expert systems, rule bases or generative theories, our approach is based on the idea that musical properties should be extracted from data sets, while as little musical expert knowledge as possible is encoded in the system's structure and parameters. The use of explicit knowledge, i.e. knowledge that is coded into the system, should be minimised. Using the existing concept of Hierarchical Hidden

Markov Models (HHMMs) and introducing the hierarchical Input-Output Hidden Markov Models (hIOHMMs), we present networks of statistical models that can learn correlated musical properties and generate new material accordingly.

Standard Hidden Markov Models (HMMs) are efficient at modelling local dependencies in data sequences (Rabiner, 1989; Bengio, 1999), but cannot represent those of a large-scale nature in an efficient way. Local dependencies are important in musical data for the encoding of style signatures (Cope, 1991), but a notion of large-scale dependencies is essential for the creation of believable musical structures; HHMMs can represent both the local and the large-scale dependencies in a data set. We introduce the concept of hIOHMMs, which provide the hierarchical structure of an HHMM coupled with the additional constraint of an Input-Output HMM (IOHMM); hIOHMMs are a valuable tool that helps modelling hierarchical data when a minimum of control over the output generation is desired.

Building separate models for different levels of description, we are using a network of probabilistic models that is made up from HHMMs and hIOHMMs to learn the dependencies of the elements inside the structures as well as the dependencies between the different layers of structures that a musical work consists of. A major focus of this research is the desire to have only a minimum of expert knowledge encoded in the system; we want to show that, using the right tools and structures of understanding, it is possible to learn sufficient information from musical data itself to generate new material.

The existing concepts of HHMMs and IOHMMs are combined and extended to the novel concept of hIOHMMs. Both the hierarchical and the IO models are then used in composite networks, whose building blocks can be reused to make up different architectures of networks. This new approach allows the flexible and dynamic modelling of hierarchical musical structures, representing both the dependencies between structures (i.e. the connection between two models in a network) and inside each structure (i.e. the structural probabilities of each model).

The project aims to devise a system that is able to create new pieces of music,

which should ideally have a clearly defined global structure, without providing it with musical context or patterns during the procedure of generating new pieces of music. The knowledge the system has access to, which is essential to write new musical pieces, is implicit rather than explicit: the knowledge, i.e. the structural probabilities and dependencies, is made available to the system via a training corpus. The amount of explicit knowledge, for instance musical rules hard-coded into the system, are kept to a necessary minimum. The system is evaluated using cross-entropy measures to determine the accuracy with which the training data is represented, as well as by conducting a listening experiment on some of the results generated by the networks.

Outline of Thesis

The thesis is organised as follows:

- **Chapter 2** provides background information on existing research in the field of computational music harmonisation and generation, as well as some theoretical background on musical structures. Finally, the chapter concludes with an outline of the scope and aims of this research.
- **Chapter 3** provides a short overview of the field of Machine Learning, explaining concepts such as entropy measures and smoothing. The definitions of Markov chains and Hidden Markov models are introduced together with their methods of inference.
- **Chapter 4** begins with the definition of Hierarchical Hidden Markov models and techniques for linear time inference. It continues by introducing the new concept of Input-Output HHMMs, an extension to the hierarchical models that is derived from Input-Output HMMs.
- **Chapter 5** is a short chapter that shows the importance of the music representation and model structures for this research, and gives details of the representation.
- **Chapter 6** outlines the design of the software used for the HHMM modelling, and gives details of the software implementation and use.

- **Chapter 7** describes how dynamic networks of models were used for the generation of new pieces of music using a "random walk" approach. Several different types of networks are presented, exploring the different possibilities of layering the musical structures and organising the networks.
- **Chapter 8** tries to evaluate musical examples that were generated with several different types of networks. The evaluation process is both subjective and objective, using the results of a listening experiment as well as cross entropy measures and musical theoretical rules.
- **Chapter 9** offers a discussion of the methodology of the approach, the configuration and design of networks and models as well as the learning and generation of the new musical structures.
- **Chapter 10** concludes the thesis by summarising the research's contributions, evaluating whether the project scope has been fulfilled and the major goals of the research have been met.

Chapter 2

Background

Applications in AI and music can range from composition and harmonisation to sound processing and signal analysis, and can thus be divided into two major groups: music analysis and music synthesis (Kramer, Stein, & Wall, 2006). Kramer et al. present a taxonomy of problem classes, distinguishing applications by problem types, modelling levels or levels of abstraction, and arrangements directions, i.e. horizontal and vertical tasks. The task of composing a melody is thus described as being a synthesis task that is horizontal in direction and symbolic in its level of abstraction. Harmonising a melody is seen to be a task that is both horizontal, by following the progression of the melody, and vertical, by building the harmonies between different parts.

The scope and methodology of the present work were influenced by a diverse range of related research projects. This chapter provides an overview of these projects, trying to highlight both their strengths and shortcomings, which we try to address in this research. Several research projects, using different techniques such as neural networks, genetic algorithms or rule-based systems, concentrate on the harmonisation of simple melodies, notably Bach chorales. The insight gained by analysing the results and methodologies of these harmonisation systems is extremely valuable when devising a system for the generation of new pieces of music. The problem of generating new music is naturally more complex than the harmonisation problem: while a system that harmonises a given melody can benefit from the structures already incorporated into this melody, e.g. phrasing structure or cadential patterns, a

generative system has to create these structures from scratch. However, the techniques used in systems that successfully solve the harmonisation problem can potentially be reused and altered to fit the more difficult problem. Some background is also given on research that addresses the problem of generating music computationally and on theories of musical structures that influenced this work. Finally, we outline the scope and aims of this research.

2.1 Hierarchical Structures in Music

The hierarchical properties of musical structures can be used to represent the musical data on several levels of description, and thus play a vital role in any system that tries to address the problem of generating music computationally. Human cognition tends to organise complex structures, such as natural language, music or visual input, into hierarchies (Bod, 2002). These hierarchies can be represented with tree-like structures. The complexity of the represented knowledge domain decreases with every level of the tree structure, i.e. the leaves of the tree are the smallest and simplest units of representation that can describe the domain. The lowest levels of a hierarchy represent the small-scale structures that make up a complex organisation. According to their relative places in a hierarchy, small-scale structures can be combined to build the more complex medium-scale and large-scale structures. The root of the hierarchy is equivalent to the entire knowledge domain that the hierarchy reflects.

In their "Generative Theory of Tonal Music", Lerdahl and Jackendoff (1983) identify the main large-scale structures that build musical surfaces and define them using 'well-formedness rules' and 'preference rules': *grouping structure* divides a piece of music into a hierarchy of motives, phrases and sections; *metrical structure* expresses several hierarchical levels of strong and weak beats; *time span reduction* assigns levels of 'structural importance' to events given their relative positioning in groups and phrases; finally, *prolongational reduction* defines a hierarchy of harmonic and melodic tension and progression. They define a hierarchy as a structure, composed of discrete elements. An element of the hierarchy that contains other elements is superordinate to those elements, which in return are subordinate to the

element that contains them. Lerdahl and Jackendoff also define hierarchical levels in music:

> "Elements ... that are about equally subordinate within the entire hierarchy can be thought of as being at a particular hierarchical *level.* A particular level can be spoken of as *small-scale* or *large-scale*, depending on the size of its constituent elements ..." (Lerdahl & Jackendoff, 1983)

An event on a small-scale level is a motive or a specific cadential pattern, a large-scale event on the other hand can be a phrase or an overall harmonic structure that progresses towards a cadence.

The hierarchical nature of the grouping structure helps a listener identify and understand the overall structure of a piece of music by dividing it into smaller, characteristic entities such as motives, themes or phrases. The metrical structure represents the relationships between alternating weak and strong beats; events that occur on strong beats are perceived to be structurally more important than events that occur on weak beats. The function of an event is thus influenced by its relative placing in the metrical structure. Temperley (2001), who builds his work onto Lerdahl and Jackendoff's theories, underlines this with some of his metrical preference rules. The harmony rule, for instance, states that changes in harmony should be aligned with strong beats, the stress rule states a preference for stressed events on strong beats. In conjunction with metrical and grouping structure, time-span reduction assigns different levels of importance to musical events depending on their relative place in a group and their metrical position's strength.

2.2 Rule-Based Systems

Rule-based systems, also called knowledge-based or expert systems, use rules and facts to represent a specific part of the real world. An inference engine matches facts against rules using an inference mechanism, thus solving given problems. The major issues in developing rule-based systems are the representation and the acquisition of the expert knowledge that defines a problem space. Knowledge acquisition focusses on transferring the expert knowledge from a source to the system. If, for instance, the source is a human expert, the knowledge is directly, or

explicitly, coded into the expert system in the form of rules. However, knowledge acquisition can also use induction to extract information from a data set and implicitly create sets of rules. Knowledge representation deals with representing the expert knowledge inside the system. A common representation technique is the use of production ("if-then") rules: *if* a certain *condition* is true, *then* execute a given *action*.(Jackson, 1990)

2.2.1 CHORAL

Ebcioglu (1993) developed the expert system CHORAL for the harmonisation of chorales in the style of Bach. CHORAL is a knowledge-rich system that uses a rule base, in conjunction with heuristics, to generate harmonisations. The principle of multiple viewpoints is introduced to simplify the complex musical knowledge and divide it into distinct groups. Each viewpoint is then represented by its own knowledge base. The main viewpoints for the harmonisation process are the chord skeleton and the individual voices. The chord skeleton viewpoint, represented as a sequence of rhythmless chords with phrasing structure, generates one chord per step. The voice leading viewpoint, which generates the actual notes, contains three lower-level viewpoints to represent ornamentation, harmonic constraints and melodic lines.

The generation process for each viewpoint has three steps: first, production rules generate possible solutions; then a set of constraints verifies the validity of those solutions; finally, a heuristic search method seeks the solution with the most desirable properties. The processes that generate the results from the different viewpoints are run simultaneously, the chord skeleton viewpoint being the clock process, which all other processes must follow. The solution is generated step-by-step according to the clock process; if no solution can be found for one step in the process, backtracking is employed to return to an earlier stage and change the solution.

Ebcioglu states that the knowledge-base of CHORAL, whose rules were explicitly coded into the system, is very large and complex and that the heuristics are the most important step in creating aesthetically pleasing music. Every situation that the system might encounter while harmonising a given melody needs to be

defined and hard-coded into the knowledge base. Also, while the production rules and the constraints are responsible for the correctness of the results, the heuristics define the style by giving preference to desirable solutions. The amount of domain and style-specific knowledge, that has to be encoded into CHORAL explicitly, is vast. However, the subdivision of the task into separate, hierarchical viewpoints greatly reduces the complexity of the problem. Ebcioglu evaluates the competence of his system as approaching "that of a talented student who has studied the Bach chorales".

2.2.2 Minuets in Classical Style

Löthe (1999) addresses the problem of composing music from scratch using a rule-based system. Due to their strict and regular form, he focusses on minuets in the early classical style. Löthe distinguishes two levels of composition: a microstructure that represents the horizontal (melodic) and vertical (harmonic) relationships between notes, and a macrostructure, i.e. the horizontal relationships between larger parts of the music. He also highlights the importance of the relative place of an event in a score:

> "Decisions in the microstructure of a part depend on the role the part plays in the macrostructure of the piece. Therefore, musical parts cannot just be concatenated together, they have to be purpose-built to fit into their role." (Löthe, 1999)

The compositional process is subdivided into a number of subtasks, such as planning the large-scale harmonic structure or constructing the melody or bass lines. The system's rule-base defines all the valid events and progressions of a minuet and new works are generated note by note based on those rules.

Both Löthe's and Ebcioglu's approaches use knowledge-bases that define every aspect of the systems' behaviour. In order to simplify the harmonisation and composition procedures, the main problems are split into smaller, simpler subtasks. Dividing a large complex problem into a number of more manageable problems implicitly decreases the complexity of the system. However, the main drawback of the knowledge-intensive approach, i.e. the large amount of explicit knowledge that

is hard-coded in the system, remains and the need for more flexible solutions that can adapt more easily, for instance to different musical styles, becomes evident.

2.3 Genetic Techniques

Genetic Algorithms use techniques based on a process of evolution to search a large domain for optimal results to a given problem. The starting point for the search is the first generation of a population of often randomly initialised candidate results. These candidates are also called chromosomes, which evolve during a mutation and cross-over cycle. Their overall quality, or fitness, is then assessed and only the fittest candidates of a population continue to live to the next generation. The fitness evaluation is generally based on domain-specific constraints, specifying the rules or properties an optimal solution should respect. Chromosomes that do not respect those constraints are filtered out in favour of candidates that are closer to the optimal result. The evolution cycle is repeated until the population's overall fitness ceases to improve and an optimum has been found. Genetic algorithm systems learn to improve a sample of results for a given domain based on expert knowledge or subjective evaluation. A detailed introduction to genetic algorithms and their application to music can be found in (Russell and Norvig, 1995, Chapter 20) and (Burton & Vladimirova, 1999) respectively. The following paragraphs describe two different approaches to the genetic technique applied to music harmonisation and composition.

2.3.1 Harmonising Melodies with Genetic Algorithms

Phon-Amnuaisuk et al. (1999) and Wiggins et al. (1999) describe a system for the four-part homophonic harmonisation of simple melodies, based on genetic algorithms. The main aim of their research is the simulation and understanding of the human behaviour involved in the harmonisation process. The quality of the produced musical results is important, however the research also focusses on how and why those results were achieved.

The system presented is a knowledge-rich, non-interactive genetic algorithm. A non-interactive algorithm does not allow human evaluation of the fitness of any in-

chromosome length

Soprano	[0,0,3]	[0,0,3]	[4,0,2]	[0,0,3]	[1,0,3]	[4,0,2]
Alto	[2,0,2]	[2,0,2]	[2,0,2]	[2,0,2]	[4,0,2]	[1,0,2]
Tenor	[4,0,1]	[4,0,1]	[2,0,1]	[0,0,2]	[7,0,1]	[1,0,1]
Bass	[2,0,1]	[0,0,1]	[0,0,1]	[4,0,1]	[4,0,1]	[7,0,0]
Duration	1	2	1	1	2	2

Figure 2.1: *Schematic diagram of a four-voice harmony chromosome.(Phon-Amnuaisuk et al., 1999)*

termediate results. Instead, it uses a fitness evaluation function which relies heavily on expert knowledge from the musical domain. This expert knowledge is mainly encoded in the reproduction operators and the fitness function, however the representation of the chromosomes is also rich in domain specific knowledge. Figure 2.1 shows an example of a chromosome used in this system: it describes the four parts of a harmonised melody, with each column representing a chord. The soprano line is fixed, the three remaining parts are initialised at random and evolve with each generation through crossover, mutation and selection operations. Each pitch is described with three integers which denote its relative pitch, accidental and octave, e.g. `[5,0,2] = A4`.

The reproduction operators use mutation and crossover techniques adapted to musical material, for instance allowing voices to swap, changing pitch by semitones or changing chords based on the melody. After the reproduction operators have evolved the chromosome population, the fitness function assesses each chromosome's fitness according to a large number of criteria. Avoiding parallel fifths/octaves between voices, hidden unisons or leaps of augmented/diminshed intervals, as well as the preference of stepwise motion or the doubling of the root are some of the evaluation criteria in the fitness function. Each mistake is penalised, thus decreasing a chromosome's overall fitness and its likelihood to survive into the next generation.

The overall results generated by the system were found to be of acceptable quality,

lacking mainly in coherent large-scale progression. The assessment of the large-scale structure of the harmonisations however is not a part of the fitness function's criteria, thus the system cannot be expected to generate large-scale progressions. It was also found that even after 300 generations, the population could not satisfy all the constraints set by the fitness function. In music, the context of an event is crucial to the event's function. Therefore for instance, changing (mutating) one chord in a progression according to the fitness function changes the entire progression and does not necessarily increase a chromosome's fitness. The algorithm often gets stuck in a local optimum without being able to find the best possible solution for a problem.

Although the results generated by this genetic algorithm system are to some degree musically acceptable, its shortcomings and limitations are quite clear. The lack of evaluation of large-scale structure and the system's weakness in handling musical context severely limit the quality of the results that can be produced with this system and are the main discrepancies when comparing the system to human behaviour. The relatively large amount of expert knowledge encoded into the system helps to evolve the local structures and reduces mistakes. The overall quality of the music however suffers because of the lack of large-scale progression and the often poor use of context, thus underlining the importance of large-scale structure and dependencies as well as the relative timing of events.

2.3.2 GeNotator

Thywissen (1999) developed an interactive compositional tool, which uses a genetic algorithm to evolve musical structures, such as melody, harmony or rhythm, within a grammar-based model. The goal is to evolve musical structures through interactive evaluation by the system's users rather than a fitness function. Instead of initialising the population randomly, this system tries to narrow down the search space by giving the chromosomes some degree of musical structure, pre-defined through a generative grammar. Each chromosome's fitness is evaluated separately, therefore the more pre-defined the candidate solutions are, the smaller the number of generations that could lead to satisfactory results. This reduces the time spent evaluating intermediary results.

GeNotator's generative grammar is defined by the system's users, and can include basic grammatical constructs, such as iteration or concurrency, as well as music specific constructs and transformational rules, e.g. scales, rhythms or phrases and transposition or modulation. The grammar is compiled into a parse tree[1], which is used to initialise the chromosome population. The chromosomes are created by scanning through the parse tree and adding a gene for every possible choice route in the tree. The algorithm uses the standard mutation and cross-over techniques to alter the population, and the users can influence the evolution by controlling the probabilities of the reproduction operators.

The GeNotator's ability to generate interesting music is heavily dependent on the quality of the generative grammar and the subjective evaluation of intermediary results, thus the quality of the results can be expected to vary considerably with different users. GeNotator is a compositional tool that allows users to create their own search space for the genetic algorithm interactively, defining musical styles and rules. Hierarchical structures in the generative music grammar are highlighted through the parse tree. GeNotator relies more on using specified knowledge in the generative grammar than on extracting optimal results with the genetic algorithm based on general rules of music theory. However, this is a valid approach for a compositional tool that is designed to reflect each user's personal choice of music.

2.4 Neural Networks

Artificial neural networks are a mathematical representation of the interconnection of neurons in a (human) brain, i.e. a representation of the network of cells that are responsible for processing information. A neural network is thus a network of artificial neurons, which are represented as nodes that are connected by links, each associated with a weight[2]. Every node has both an input and an output link

[1] A parse tree represents the syntactic structure of a string of words. Its internal nodes represent phrases, its leaf nodes represent single words. The links between the nodes represent the grammar rules.(Russell and Norvig, 1995, Chapter 22; Jurafsky and Martin, 2000, Chapter 9)

[2] The weight of a link is a numeric value that is associated with that link. Weights are often initialised arbitrarily and later updated by the neural net according to the training data. (Russell and Norvig, 1995, Chapter 19)

to allow it to communicate with the remaining nodes in the network. A node also has an activation threshold: based on the input from neighbour nodes, a node can be activated if the sum of the input weights surpass its activation threshold. The learning process in a neural net involves updating the weights of the network in order to reflect the training data. (Russell and Norvig, 1995, Chapter 19) Initial knowledge can be encoded into a neural net through the network's structure, for instance by choosing a specific number of nodes or restricting the links between nodes, as well as the initial weight settings for the links and the representation of the domain specific knowledge inside each node. However, a large amount of information is extracted by the neural net and incorporated into the system by adjusting the connections and the weights attached to them.

2.4.1 HARMONET

Hild et al. (1991) developed HARMONET, a system of neural networks, designed to learn how to harmonise Bach chorale melodies by extracting short-term structural probabilities from existing harmonisations, i.e. learn by example. The harmonisation problem was divided into three subtasks: first, the harmonic structure is predicted; then, given the harmonies and the melody, the chords are built; finally, the ornamentation is added. Creating the harmonic skeleton, i.e. finding an appropriate harmony for every beat of a melody, is treated as the most important step in the harmonisation process. Therefore, HARMONET uses context windows to predict harmonies (see Figure 2.2): the new harmony H_t at time t is based both on the harmonic context of the three previous harmonies H_{t-3}, H_{t-2} and H_{t-1} and on the three-note melodic context s_{t-1}, s_t and s_{t+1} around the current note s_t. The harmony H_t contains information about harmonic function, inversion and characteristic dissonances, i.e. non-harmony notes. Additionally, the process has information about the current note's relative place in a phrase and its positioning on a weak or a strong beat.

The second neural net addresses the problem of filling in the chords given the melody and the newly created harmonic structure. Information about chord inversion is included in the harmonies, thus the bass line is implied by the harmonic structure and only the alto and tenor parts need to be inserted. A neural net gen-

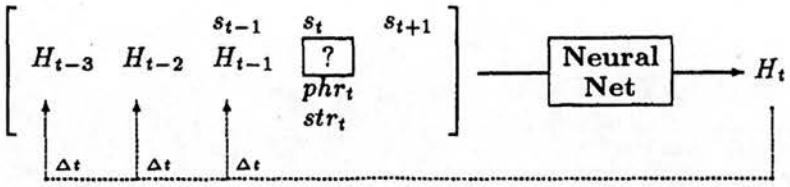


Figure 2.2: *HARMONET uses both the harmonic context (H_{t-3}, H_{t-2} and H_{t-1}) and the melodic context (s_{t-1}, s_t and s_{t+1}) to predict the harmony H_t. phr_t and str_t contain timing and metric information.(Hild et al., 1991)*

erates the chords and a symbolic algorithm[3] selects the best solution. The final network is used for the ornamentation step, using several attributes, such as the intervals between the chords C_t and C_{t+1} and voice leading rules, as input.

HARMONET was trained on a small number of chorales only, however the networks were not trained on entire chorales. Short, overlapping time windows of four or five beats were used instead. The twenty chorales used as training data thus amount to approximately 1000 examples to learn from. As a result, the system does not learn the overall large-scale structure of the Bach chorales, but rather the short-term dependencies that define the melodic and harmonic structures inside the time window. However, HARMONET counterbalances the lack of large-term dependencies by encoding knowledge that will help overcome the problems that result from training on small-scale structures only. The system keeps track of the relative place of an event in the score, thus helping to distinguish the different functions of events based on their timing. Although the time window is only four or five beats, the system can distinguish between the beginning or the end of a piece.

2.5 Probabilistic Models

Sequences of events can be represented by probabilities that define the frequencies and contexts of the events as well as the transitions between them. Context models,

[3]Symbolic algorithms are algorithms that are taken from symbolic AI, e.g. expert systems or Bayesian networks, as opposed to connectionist AI, e.g. neural networks. (Gurney, 1997)

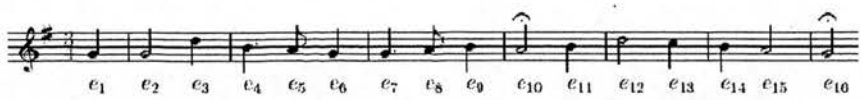
a simple type of probabilistic models, store the frequency of occurrence of each event given certain contexts in a database. These probabilities are then used to predict the next event in a sequence given a context of a fixed number of previous events. Markov models (also Hidden Markov models, see sections 3.4 and 3.5) represent sequential data through states and the probability of transition between those states. The context information is limited to the order of the Markov model, i.e. a model of order 1 uses a context of size 1.

2.5.1 Context Models and Multiple Viewpoints

Based on the conjecture that systems with high predictive power also have high generative power, Conklin and Witten (1995) introduce a multiple viewpoint system of context models for the prediction and generation of music. A context model has a database that contains all the sequences of a possible event space and a frequency count attached to each one of those sequences. The model also provides an inference method to calculate the probability $p(e|c)$ of an event e given a context c. This particular system is trained on a sample set of Bach chorales.

Context models that work with only one musical feature are in themselves not powerful enough to represent all the musical surfaces that make up a chorale, e.g. pitch, duration or phrasing. Conklin and Witten therefore define several viewpoints, which are all represented by their own underlying context model. Some of the basic viewpoints are 'pitch', 'start time', 'key signature' and 'duration'. These viewpoints can be linked together, or derived into more specialised viewpoints such as 'position in bar' (derived from 'start time') or 'sequential interval' (derived from 'pitch'). The entire group of viewpoints used in a system is referred to as the system's event space. The system also distinguishes between long-term effects and short-term effects by representing them with different context models, or combinations of different models.

Figure 2.3 shows the application of the multiple viewpoint approach to a simple chorale melody. Each viewpoint separately encodes one level of musical representation for every event in the melody. The viewpoint information for every sequence, i.e. each event and its context, is then stored in a database together with the



Type	Event number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
st	8	12	20	24	30	32	36	42	44	48	56	60	68	72	76	84
pitch	67	67	74	71	69	67	67	69	71	69	71	74	72	71	69	67
duration	4	8	4	6	2	4	6	2	4	8	4	8	4	4	8	8
keysig	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
timesig	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
fermata	F	F	F	F	F	F	F	F	F	T	F	F	F	F	F	T
deltast	⊥	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2.3: *Multiple viewpoint information for the events in a simple melody. (Conklin & Witten, 1995)*

sequence's probability[4]. In order to find the best generative system, the predictive power is tested using a number of combinations of different viewpoints and measured with an entropy profile. The entropy value (see section 3.2) for each event reflects the predictability of the event for a given viewpoint and context. The event space for the Bach chorales is created by linking the pitch, key signature, time signature, fermata, start time and duration viewpoints. The system generates new examples of music by concatenating a given context c and a newly created event e, thus producing a new context $c+e$. Figure 2.4 shows an example of a melody created by the multiple viewpoint system. The system was given the first seven events of the melody and the rhythmical structure for the entire piece as input.

The multiple viewpoint approach is very successful in separating the different levels of musical description and thus simplifying the process of generating a musical surface. Some of the works mentioned earlier, e.g. (Ebcioglu, 1993) and (Hild et al., 1991), employ a similar way of dividing their main task into more manageable subtasks. However, Conklin and Witten's system allows a flexible and dynamic combination of viewpoints that best describe a musical structure. The system needs to be provided with an underlying structure and context in order to

[4]A detailed description of the system's inference method is described in (Conklin & Witten, 1995).

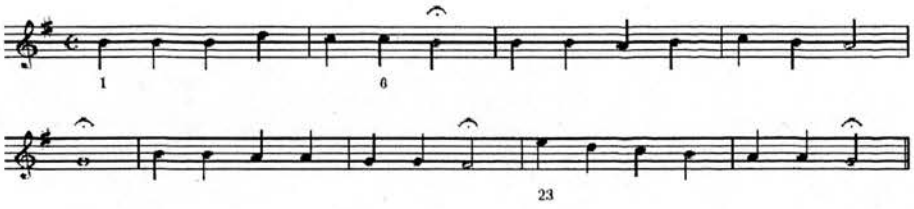


Figure 2.4: *Short melody generated by the multiple viewpoint system, using the first seven events and the rhythmical structure of the entire piece as input.(Conklin & Witten, 1995)*

generate new examples of music; it is limited to adding new events to an already existing sequence of events. The different viewpoints are encoded with a maximum of musical knowledge in order to obtain a detailed description of the musical surfaces. The context models do not use this knowledge directly, they simply store sequences and their probabilities. It is rather the combination of the information stored in the viewpoints, represented by the models, that makes the system powerful enough for generative purposes.

2.5.2 Harmonising Bach Chorales with HMMs

Allan (2002; n.d.) developed a system of HMM (see section 3.5 for details) for the harmonization of Bach chorale melodies, that aims at creating harmonizations based on the short-term dependencies extracted by the models. Pre-encoded knowledge of harmonization rules and music theory are avoided. In order to simplify the task of adding extra voices to the given melody, the process is split into three parts, based on the research by Hild et al. (1991): a first HMM models the harmonies depending on the melody, a second model fills in the chords and a final model is responsible for the ornamentation. The Viterbi decoding method is used to generate the harmonizations: for a given melody, the models search for the best possible sequence of harmonies, then chords and ornamentations. The result returned is a global optimum, i.e. a harmonization which has the highest overall likelihood given the melody.

The first HMM models the dependencies between melody notes and harmonic functions. This model then generates the best possible harmonic sequence for the melody, which in return is used together with the melody to fill in the three remaining voices. Alto, tenor and bass are not treated as separate parts, but rather as single events. These single events encode the melody pitch and the three additional voices as well as the intervals between them, represented by the distance in half-tones. For instance, given the melody note `G`, a tonic chord with a `C` as the tenor and bass, `E` as the alto and `G` as soprano would be described `19:16:12:0:T`, i.e. 12 half-tones between bass and tenor, 16 between bass and alto and 19 between bass and soprano. The ornamentation model inserts passing and neighbour notes to all four voices separately by diving each beat into four parts, i.e. a crotchet becomes four semi-quavers, and filling the space between the current pitch and the first pitch of the following beat. Figure 2.5 shows the completed harmonisation of a Bach chorale melody.

The division of the harmonisation task into three easier subtasks and the representation of the music are the only pieces of expert knowledge that this system makes use of. The HMMs extract the probabilities that connect the harmonic structure with the melody and chords, basing the generation of new harmonisations onto those probabilities. The rigid representation of the three newly added voices can be seen as a drawback: because the new parts are not treated separately, there can be a lack of horizontal flow in those parts, possible resulting in awkward leaps. The system simply adds the chords like vertical building blocks, not taking the structure of each individual voice into consideration directly. Allan states that learning the short-term dependencies in the musical data should be sufficient to create satisfactory harmonisations, however the lack of large-scale dependencies becomes apparent especially in poor phrase structuring. Nevertheless, Allan shows that, even with minimal supervision and domain specific knowledge, a relatively simple probabilistic system can be a powerful tool to solve aspects of the harmonisation problem.

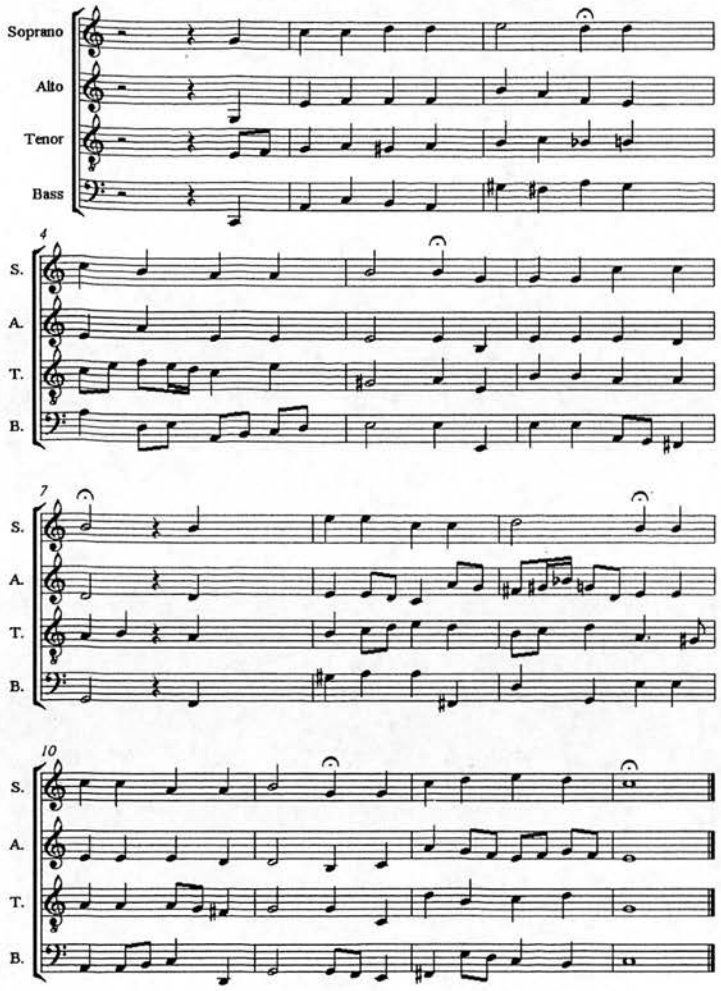


Figure 2.5: *Example of a harmonisation created by the HMM system. The melody was taken from chorale K211, BWV367. (Allan, 2002)*

2.6 Project Scope and Goals

The scope of this project was to devise a system that can learn how to create new music from an existing set of musical examples, then using this knowledge to generate the structures of simple musical works. The project's emphasis lies on three main points. Firstly, the system should be able to create new pieces of music entirely from scratch, i.e. no musical context or patterns should be provided. The motivation for this restriction is the challenge to generate new material that reflects the structural characteristics of the training data without the influence of any pre-existing patterns of musical context. If musical context is provided during the generation procedure, this context is likely to influence the outcome of the generation and thus the newly created material does not solely reflect the information and knowledge extracted from the training corpus. Secondly, the generated music should ideally have a clear large-scale structure, i.e. melodies for instance should progress towards a cadence. As this research tries to set up a system that is capable to generate believable musical structures, the generation of event sequences that progress towards cadences is defined to be a minimal condition that has to be fulfilled in order to achieve results of acceptable quality. Finally, the system should use as little domain specific knowledge as possible, learn from examples and automatically extract the knowledge needed to create new musical examples, thus make use of implicit rather than explicit knowledge. Rather than explicitly defining all the information and domain knowledge that a system needs to have access to to be able to create musical pieces from scratch, it is more desirable and flexible to develop a system that can extract and store the necessary information from a training data set.

From the works described in this chapter, it becomes clear that creating a new musical work computationally is a data-intensive and complex problem; certain AI techniques are more likely to be successful in solving the problem, given all the constraints outlined in the project aims. For instance, given that the system should take minimal advantage of domain specific knowledge, the use of an expert system is ruled out. Similarly, for a musical problem, a genetic algorithm needs a certain amount of expert knowledge to evaluate intermediary results with its fitness func-

tion. It has been shown that music generation (or harmonisation) is a problem that cannot easily be solved in a single step (Ebcioglu, 1993; Hild et al., 1991; Löthe, 1999; Allan & Williams, n.d.). As musical surfaces are defined by both small-scale and large-scale structures, the system should be able to support and underline the timing of events, their relative placing in a musical structure and the hierarchies in music.

Hierarchical Hidden Markov Models respect all the constraints set up for this project. The hierarchical structure facilitates the learning of large-scale structures in data through multiple levels of states that describe sequences at different levels of precision (Skounakis, Craven, & Ray, 2003). Allan and Williams (n.d.) have shown that simple HMMs can learn the structural probabilities of musical data without needing a large amount of domain specific information. Also, HHMMs (just like simple HMMs) allow the division of a problem into simpler subtasks by addressing each of those subtasks by a separate model. By feeding the output of one model into another, dynamic networks can be constructed to create entire examples of layered musical structures.

The system is expected to learn the structural probabilities and interdependencies that define the training data, namely Bach chorales, and generate simple musical examples (two and four parts) from scratch, reflecting musical properties and displaying a certain degree of large-scale structure. The rules of composition in the style of Bach should be reflected in the models and the results created from them. We hope to demonstrate that it is feasible for a computational system to extract and learn the musical rules and properties necessary to generate acceptable new musical works by using the right techniques and levels of description, but only a minimum of expert knowledge.

Chapter 3

Machine Learning and Probabilistic Modeling

Machine Learning (ML), a sub-field of Artificial Intelligence (AI), is the study of systems that can learn from examples, their environments or their experiences and adapt to changing requirements, allowing them to improve their (re)actions over the change of time. ML is especially beneficial for finding solutions to problems which cannot be solved directly, namely when the necessary expertise to solve the problems either does not exist or cannot easily be explained. This is the case for instance in areas such as speech recognition or computer vision: although we cannot explain how we understand speech and why we recognise faces, machines can be programmed to learn both tasks using ML algorithms. Methods from ML are also used in data mining applications to detect patterns and regularities in large data sets (Alpaydin, 2004). This chapter gives a very brief introduction to the field of ML, focussing especially on domain specific notions, such as smoothing and entropy, that are important for the understanding of major aspects of this research.

3.1 Supervised vs Unsupervised Learning

Algorithms in ML can be classed into two major categories: *supervised* or *unsupervised* learning. In supervised learning, both the input and the output of a system are known and the output can be classified. This is the case, for instance, if a system uses labeled data as training examples and the correct output is known *a priori*.

In unsupervised learning, the correct output of a system, or the classification of the data, is not known. The pair $(x, f(x))$ represents a training example, with x being the input and $f(x)$ being the output of a function f applied to x. *Inductive learning* tries to find a function or model h, also called *hypothesis*, which best approximates f (Russell and Norvig, 1995, Chapter 18). *Partially supervised* algorithms makes use of both labeled and unlabeled training data. This approach is used when large amounts of data are available, but only a fraction of the data is labeled, thus making maximal use of sparse knowledge in order to optimise the learning process.

3.2 Entropy Measures

The successful performance of an ML algorithm is dependent on the algorithm's ability to predict future events, make decisions and generate new data according to information learnt from previous data or experience. The performance quality can be assessed with the measure of *entropy*, commonly used in information theory. Entropy measures the amount of information contained in a data set and assesses how accurately an algorithm represents the data. It is also a measure of uncertainty and predictability; the more uncertain and unpredictable the data, the higher its information content. For a set of events X, the measure of entropy is defined thus:

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{3.1}$$

with $p(x)$ being the probability of event x.

The *entropy rate* of a sequence $w_1 w_2 ... w_n$, i.e. the average information content per symbol, is calculated by dividing the entropy $H(X)$ by the number of symbols in the sequence. For a process that is both ergodic[1] and stationary, i.e. does not change over time, the entropy rate is:

$$H(L) = \lim_{n \to \infty} -\frac{1}{n} \log p(w_1 w_2 ... w_n) \tag{3.2}$$

[1] An ergodic process is a process that does not have the possibilities of endless loops, i.e. that cannot reach sub-processes from which it is impossible to escape. (Manning & Schütze, 1999)

with $p(w_1 w_2 ... w_n)$ being the probability of sequence $w_1 w_2 ... w_n$.

The entropy rate can also be regarded as a measure of the complexity of a language L. The more complex and diverse a language, the higher its entropy rate.

The measure of *cross entropy* is useful when the actual probability distribution p of a language L is not known. Instead a model m, an approximation of p, can be used:

$$\begin{aligned} H(p,m) &= \lim_{n\to\infty} -\frac{1}{n} \log m(w_1 w_2 ... w_n) \\ &\approx -\frac{1}{n} \log m(w_1 w_2 ... w_n) \end{aligned} \tag{3.3}$$

Because m is a model of p, the cross entropy value can never underestimate the real entropy. The model m of a set of events can never be more accurate than the real set of events and its probability distribtuion p:

$$H(p) \leq H(p,m) \tag{3.4}$$

The closer $H(p,m)$ is to $H(p)$, the better the model m represents the language L. The cross entropy measure can be used to compare the performance of two algorithms on the same data set. Given two models m_1 and m_2, the model with the lower cross entropy is the one that represents the data more accurately (Manning and Schütze, 1999, Chapter 2).

3.3 Smoothing

A system's predictive power is directly dependent on its ability to classify unseen events and to make rational choices when facing uncertainty (Mitrani, 1998). Training data often only represents a fraction of the knowledge that an ML system is supposed to reflect. Smoothing techniques are used to balance the probabilities of seen and unseen events.

Smoothing takes a part of the probability space from previously seen events and assigns it to unseen events, which were previously estimated by a training algorithm to have zero probabilities; smoothing is therefore also referred to as *discounting*. A commonly adopted solution for smoothing zero probability events in statistical models is *additive smoothing*: a certain value λ is added to all probabilities, which are then re-normalised (Manning and Schütze, 1999, Chapter 6). The oldest version of this solution is Laplace's law, where λ is set to 1:

$$P_{Lap}(w_1,...,w_n) = \frac{C(w_1,...,w_n)+1}{N+B} \tag{3.5}$$

with N being the number of observations in the training data, B being the number of values possible for the observations, and $C(w_1,...,w_n)$ being the frequency count of the sequence $(w_1,...,w_n)$. The new probability of sequence $(w_1,...,w_n)$ is thus its frequency of occurrence in the training data *plus* 1, divided by the sum of all possible values for the observations and the size of the training set.

The main problem with simply adding 1 to all probabilities is that the probability space assigned to previously unseen events is too large, and the discounting effect deflects a considerable portion of the probability space from events that occur in the training data. To adjust this effect, λ can be reduced to any value between 0 and 1, and used as stated in *Lidstone's law of succession*[2]:

$$P_{Lid}(w_1,...,w_n) = \frac{C(w_1,...,w_n)+\lambda}{N+B\lambda} \tag{3.6}$$

This definition can be transformed into a linear interpolation between a uniform prior $\mu = N/(N+B\lambda)$ and the Maximum Likelihood Estimator (MLE) $\frac{C(w_1,...,w_n)}{N}$:

$$P_{Lid}(w_1,...,w_n) = \mu \cdot \frac{C(w_1,...,w_n)}{N} + (1-\mu)\frac{1}{B} \tag{3.7}$$

The smaller the value of λ, the smaller the influence on the probability space. For example, consider the training data $Tr = (X,Z,X,X,Z)$ and the event set $Obs = \{X,Y,Z\}$. The size of the training set is $N = |Tr| = 5$ with $B = |Obs| = 3$ different

[2]For $\lambda = 0.5$, this definition is called Jeffrey-Perks law.

	$P(\cdot)$	$P_{Lap},(\lambda = 1)$	$P_{JP},(\lambda = 0.5)$	$P_{Lid},(\lambda = 0.1)$
X	0.6	0.5	0.54	0.59
Y	0.4	0.375	0.38	0.39
Z	0.0	0.125	0.08	0.02

Table 3.1: *Comparison of different types of additive smoothing. For $\lambda = 1$, a significant amount of probability space is assigned to an unobserved event. For small λ, the smoothed probabilities converge towards the absolute probabilities.*

possible events. Table 3.1 compares the results of the different types of additive smoothing, showing the absolute probabilities of all events, as well as the smoothed probabilities for different values of λ.

The ideal value for λ is dependent on the nature of the training data and choosing it can be difficult. If the data is sparse, i.e. if there are few high frequency and many low frequency events, the probability space set aside for zero probability events is significant and can be reduced by using a small value for λ. However, if too small a value is chosen for λ, potentially important and interesting events become highly unlikely.

3.4 Markov Models

There are several types of techniques for solving ML problems, among them Neural Networks (NN), Decision Trees and Genetic Algorithms. One group of techniques, which are based on the modelling of dependencies between the variables or events in a system, is called Bayesian Networks (BN), also referred to as Belief Networks or probabilistic graphical models (Alpaydin, 2004). A BN is a directed acyclic graph, the nodes representing variables and the arcs representing the dependencies between variables. The network is a representation of the joint distribution[3] over all the variables.

[3]The joint distribution $P(X,Y)$ is the probability of the intersection of the random variables X and Y and is defined $P(X,Y) = P(X)P(Y|X)$.

Markov models[4] are a special type of belief network. Markov models are stochastic processes that were first introduced in 1906 by *Andrej Markov*, who used them in the linguistic task of modelling letter sequences. Markov models have a finite number N of states q, with transition arcs between the states representing dependencies between the variables (or states) with respect to the *Markov property*:

$$P(q_{t+1} = j \,|\, q_1, q_2, \ldots, q_t) = P(q_{t+1} = j \,|\, q_t = i) \tag{3.8}$$

i.e. the state j at time $t+1$ is only dependent on the state i at time t. In other words: the future is only dependent on the past given the present. Equation 3.8 shows the property for a first-order chain, however higher-order processes are possible. In a second-order Markov model, the state at time $t+1$ is not only dependent on the previous state but also on the state at time $t-1$. A Markov model can be visualized as a directed graph where the edges are assigned the transition probabilities between nodes. The elements of the transition matrix $A = \{a_{ij}\}$ represent the probabilities of moving from state i to state j:

$$a_{ij} = P(q_{t+1} = j \,|\, q_t = i) \tag{3.9}$$

with $a_{ij} \geq 0$ and $\sum_{j=1}^{N} a_{ij} = 1$.

The vector $\Pi = \{\pi_i\}$ stores a model's initial state activation probabilities:

$$\pi_i = P(q_1 = i) \tag{3.10}$$

with $\pi \geq 0$ and $\sum_{i=1}^{N} \pi_i = 1$. An example of a simple 3-state Markov model is shown in Figure 3.1.

3.5 Hidden Markov Models

In a visible Markov model the variables are observable and it is unambiguous which states are active at any time. In some processes however, it might not al-

[4] A Markov model is also referred to as Markov chain for a discrete-time process, or a Markov process for a continuous-time process.

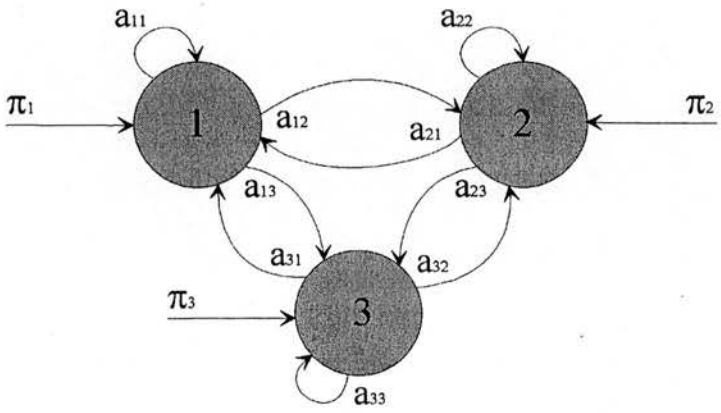


Figure 3.1: *Example of a 3-state Markov model. The probability of the model starting in state S_i is π_i; the probability of moving from S_i to S_j is a_{ij}.(Alpaydin, 2004)*

ways be possible to observe the states a system is in. HMMs are Markov models that offer an additional layer of hidden states to represent the unobservable variables in a system. They are useful when a process can be divided into underlying, or hidden events which generate the observable events on the surface (Manning and Schütze, 1999, Chapter 9). A HMM thus has one layer of hidden states and one layer of observable states. Figure 3.2 shows the interactions between the layers of a HMM. The parameters that define a HMM $\lambda = (A, B, \Pi)$ are the transition probability matrix $A = \{a_{ij}\}$, the observation or emission probability vector $B = \{b_j(k)\}$ and the initial probability vector $\Pi = \{\pi_i\}$ (Rabiner, 1989). The elements of matrix A define the probabilities for moving from hidden state i at time t to hidden state j at time $t+1$:

$$a_{ij} = P(q_{t+1} = j \,|\, q_t = i) \tag{3.11}$$

The vector B for state j defines the probabilities of observing event O_k in state j at time t:

$$b_j(k) = P(O_k \,|\, q_t = j) \tag{3.12}$$

The initial probability vector Π holds the probabilities for activating state i at time $t = 1$:

$$\pi_i = P(q_1 = i) \tag{3.13}$$

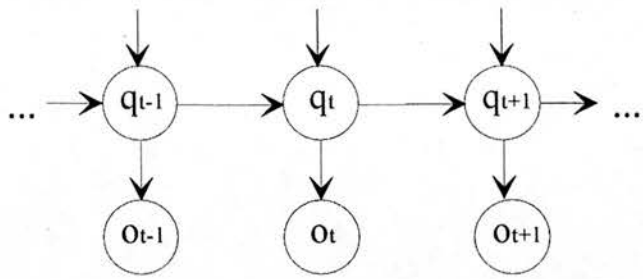


Figure 3.2: *Example of a HMM. The state q_1 is initially activated according to the probabilities in Π. The process then moves through the states q while time increases. Each time a hidden state is entered, the corresponding observable state is activated.*

Inference with a HMM is mathematically well-founded and three main algorithms solve the common questions of stochastic learning procedures:

1. Given an observation sequence $\bar{O}$ and a model $\lambda = (A, B, \Pi)$, what is the overall sequence probability $P(\bar{O}|\lambda)$? The overall sequence probability is calculated with the Forward-Backward algorithm.

2. Given an observation sequence $\bar{O}$ and a model $\lambda = (A, B, \Pi)$, what state sequence $q_1, ..., q_T$ best explains the observations? The Viterbi algorithm returns the state sequence most likely having produced $\bar{O}$.

3. Given an observation sequence $\bar{O}$, what set of parameters (A,B,Π) maximises the probability $P(\bar{O}|\lambda)$? This most complex of all problems, the training problem, can be solved using the Expectation-Maximization algorithm.

For a full account of all three algorithms the reader is referred to (Rabiner, 1989).

3.6 Summary

This chapter gives a brief introduction to the field of Machine Learning and provides detailed descriptions of ideas and techniques that are essential for the understanding of this research, such as entropy measures and smoothing. The definitions and theory of Markov chains and HMMs provide the foundation knowledge for the following chapter, which will introduce the extended concept of HHMMs.

Chapter 4

Hierarchical Hidden Markov Models

HHMMs are the hierarchical extension to standard HMMs and were first introduced by (Fine, Singer, & Tishby, 1998), with the goal of facilitating the representation and visualization, and thus the learning, of hierarchical data. HHMMs are therefore advantageous to the representation and the modelling of musical structures, allowing to make use of the hierarchies present in musical data. HHMMs are multi-levelled and can be seen as having a tree-like structure; the nodes of the tree[1] are the states of the model, the tree's edges represent the transitions between those states. Figure 4.1 shows an example of a HHMM with three levels. HHMMs define three different types of states, as well as three different types of transitions. States have different purposes according to their position in a model: the internal nodes of the tree structure are internal states; the tree's leaves are production states; and the final states on each sublevel of the model are the end states. The internal states are the only states that have children, i.e. for every internal state there exists at least one state on a lower level that can be reached from this internal state. The production states do not have any children, however they are the only states that can actively emit observations. End states are simply states that, when entered, force a return to the next higher level in the tree[2]. The transitions, which define

[1] Even though the structure is not a tree in the way that it is understood in computer science, we use this description to visualize the model's tree-like structure.

[2] The root state is the state on the highest level of the tree structure, therefore there is no end state on the root level. End states exist only from the second level onwards.

paths through a model and enable the activation of its states, are categorised thus: *vertical transitions* connect pairs of parent and child states and always move one level down in the model; *horizontal transitions* exist between two brother states, i.e. two states which have the same parent and are situated on the same level; *forced transitions* connect end states with the next higher level, therefore always moving one level up in the model.

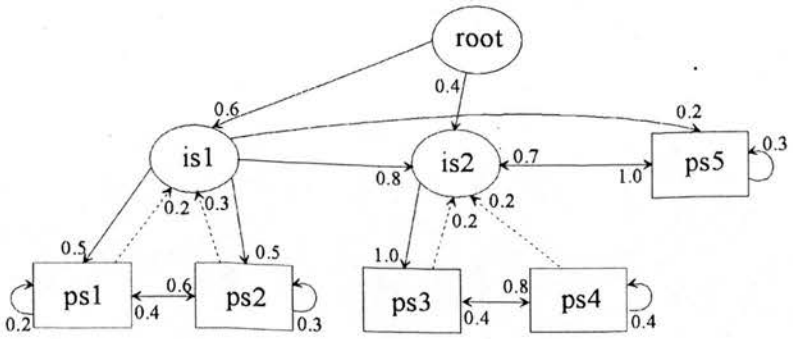


Figure 4.1: *Example of a three-level HHMM with three internal states and five production states. (The dashed transitions replace the transitions to and from the end states, which have been omitted for clarity.)*

Most importantly, throughout the model children states can be considered as being contained inside their corresponding parent states, making each internal state a probabilistic model of its own. Contrary to standard HMMs where states emit single symbols, the internal states of every HHMM recursively emit complete sequences of those observations made by the production states associated with them. Internal states can therefore learn the probability structures of whole sequences, starting from the single observations at the production state level.

4.1 Linear Time Inference

However, the strong point of HHMMs, namely their hierarchical topography, is directly related to one of the main problems of working with them: the computational expense. Due to their hierarchical structure, the conventional algorithms

that were defined by Rabiner (Rabiner, 1989) for standard HMMs and adapted to hierarchical structures by Fine et al. (1998) prove to be highly impractical. For instance, when executing the Forward-Backward (FB) algorithm to calculate the overall probability of a given sequence of length T, the machine has to process the forward variables for every possible path through the model, given each symbol in the sequence. Additionally, the Expectation-Maximization (EM) algorithm requires the calculation of backward variables, which involves processing every possible path for every possible sub-sequence of the initial input sequence. For long sequences, i.e. training data sets, and models with a large number of states, the computational time exceeds reasonable limits. The runtime performance for the algorithms by Fine et al. (1998) are $O(NT^3)$, N being the number of states in the model and T being the size of the input sequence or the training data set.

Several research projects concentrated on finding alternative ways of inference and training HHMMs (Murphy & Paskin, 2002; Ueda & Sato, 2001). However, the most simple and efficient approach was discovered by Wierstra (2004): suggesting that every HHMM in minimally self-referential form (see section 4.2) can be transformed into an equivalent flat HMM, he devised algorithms that use the flat equivalent of a HHMM, switching between the hierarchical structure and the flat representation of the hierarchical structure.[3] Runtime for the major algorithms is hereby improved from $O(NT^3)$ to $O(N^2T)$ and is only linearly dependent on the size of the training data set.

4.2 Minimally vs Maximally Self-Referential HHMM

Maximally self-referential (maxSR) HHMMs are models that have self-referential loops on internal states, i.e. internal states have horizontal transitions that point back to themselves. As opposed to maxSR HHMMs, minimally self-referential (minSR) HHMMS are not allowed to have self-referential loops on internal states, although they are permitted on production states. A HHMM that is in minSR form has only one possible path between each pair of production states and the hierar-

[3] We do not have formal proof that the transformation from maxSR to minSR to a flat model (and vice versa) is *always* possible for every type of model structure, however we strongly believe this to be true through the experience of successfully converting a large number of different models.

chical form of the model can be easily transformed into a flat model by calculating a new set of transition probabilities between all production states in the model. If a HHMM is not in minSR form, the probabilities of the self-referential loops can be shifted to the production state level, giving the model the required structure.

4.2.1 maxSR to minSR

While converting an HHMM from maximally to minimally self-referential, the equality of both models needs to be respected, i.e. sequences generated by both models need to have the same overall probability. The fundamental difference between a minSR and a maxSR HHMM is that, in a model that is minimally self-referential, the shortest path between any two production states of that model is at the same time the *only* path between those production states. This property allows the transformation of a minSR hierarchical HMM into a flat HMM. In a maximally self-referential model, on the other hand, there is more than one possible path. The new horizontal transition probability between the states i and j in the minSR model, both substates of q (and with i and j possibly being the same state, but j not being an *end* state), is:

$$a^{q*}_{(i,j)} = a^{q}_{(i,j)} + \left(a^{q}_{(i,end)} \cdot a_{(q,q)} \cdot \pi^{q}_{j}\right) \tag{4.1}$$

The new horizontal transition probability between i and j is the existing transition probability between those states plus the probability to go from state i to state j via their parent state: the product of the probabilities from state i to the *end* state, the self-referential loop in the parent state and the vertical activation of state j. The new transition probabilities towards *end* states need to balance out the fact that internal states are not allowed to have self-referential loops. The old probability of an internal state loop therefore has to propagate back through the system:

$$a^{q*}_{(i,end)} = a^{q}_{(i,end)} \cdot \left(1 - a_{(q,q)}\right) \tag{4.2}$$

After updating the horizontal transition probabilities between state q's children, simply set the probability of q's self-referential loop to zero:

$$a^*_{(q,q)} = 0 \tag{4.3}$$

After calculating the new probabilities, it is necessary to normalize the new values by dividing each value by the sum of all transitions from state i[4]:

$$A^{q*}_{(i,j)} = \frac{a^{q*}_{(i,j)}}{\sum_{j=1}^{N} a^{q*}_{(i,j)}} \tag{4.4}$$

These steps are repeated for every internal node, starting at the level closest to the production state. The horizontal transition probabilities between the *root* state's children do not need to be re-calculated: because there is no end state on the second level and no self-referential loop to the *root*, the above mentioned equations are not valid for those states. Instead, the horizontal probabilities between the *root*s children states simply need to be normalized to balance out the lack of self-referential loops.

4.2.2 minSR to maxSR

The transformation of a minimally self-referential into a maximally self-referential HHMM follows the same principle, using the existing equations to extract the probabilities:

$$A^{q}_{(i,end)} = \frac{A^{q*}_{(i,end)}}{1 - A_{(q,q)}} \tag{4.5}$$

and

$$A^{q}_{(i,j)} = A^{q*}_{(i,j)} - \left(A^{q}_{(i,end)} \cdot A_{(q,q)} \cdot \pi^{q}_{j}\right) \tag{4.6}$$

The value for $A_{(q,q)}$, the probability of a self-referential loop on parent state q, can be chosen to be either the value of that particular transformation in the previous maxSR version of the model, or, if no previous maxSR form of the model exists, an estimated value.

[4] In equations 4.4, 4.5 and 4.6, the capital letter A is used to represent a (re)normalised value of a horizontal transition probability a. It is not to be confused with a horizontal transition probability matrix.

4.3 Conversion into flat HMM

When a HHMM is in minSR form each pair of production states in the model will be connected by one single path. The flat model will not contain any internal or end states. Transforming the HHMM into an equivalent flat HMM thus means building a HMM from the production states only, calculating the new *direct* transitions that have not been present in the HHMM. Figure 4.2 shows the transformation of a hierarchical model to its flat equivalent. The transitions between brother states, i.e. children of the same state i, will remain unchanged. All other transitions that were possible in the HHMM between production states, via *end* and internal states, need to be computed thus:

$$a^*_{ij} = a_{i,end} \cdot a_{q_i q_j} \cdot \pi^q_j \tag{4.7}$$

i.e. the probability of going from state i to state j is the product of the horizontal probability of going from state i to the *end* state, from state i's to state j's parent, and the vertical probability of activating state j.

The vertical transition probabilities need to be transformed into initial activation probabilities by computing the product of the vertical probabilities from the *root* state to every production state q^D:

$$\pi^*_{q^D} = \pi^{q^{D-1}}_{q^D} \cdot \pi^{q^{D-2}}_{q^{D-1}} \cdot \ldots \cdot \pi^{root}_{q^2} \tag{4.8}$$

(with D being the production state level).
After a hierarchical model has been flattened, the standard FB and Viterbi algorithms can be used for the calculation of overall sequence probabilities and best overall states sequences respectively. The re-estimation of the model parameters has to be adapted to rebuild the hierarchical structure.

Although technically the hierarchical structure of a HHMM is lost during the flattening process because the tree structure is given up, logically the hierarchical ideas are still present in a HMM that has been built from flattening a HHMM; the models

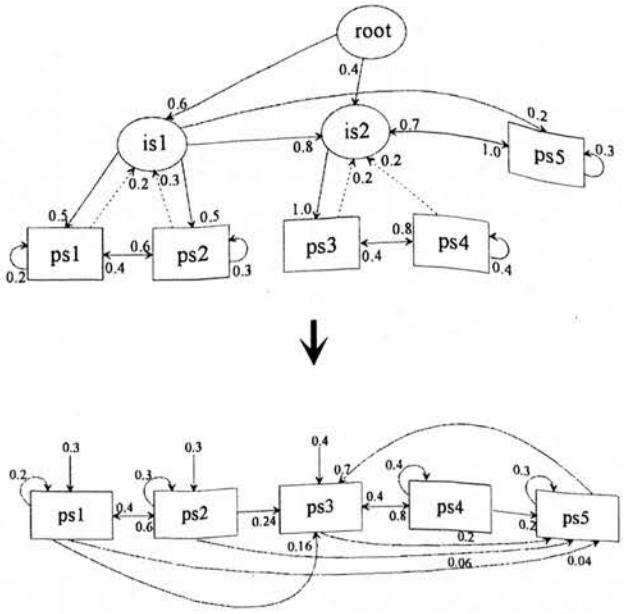


Figure 4.2: *Example of a HHMM and its equivalent flat HMM. (The dashed transitions in the hierarchical model replace the transitions to and from the* end *states, which have been omitted for clarity.) The vertical transition probabilities are converted into initial probabilities; existing direct transitions are copied into the flat model. Paths between two production states are transformed into direct transitions by adapting the probabilities.*

are equivalent regarding the information they can learn. However, the tree structured HHMM is restored after the training process as a way to visualize the hierarchical structures in the training data and reflect the probabilities that connect the internal nodes in the model, because the flat model cannot represent hierarchical structures in the same logical way that a HHMM can. The models are not equivalent in the way they represent hierarchical structures. A hierarchical model can learn the interdependencies of states on different levels. In a flat model that is theoretically equivalent to a hierarchical model, the probabilities that represent movements between internal states are lost. For instance in Figure 4.2, the probability of moving from internal state $is1$ to internal state $is2$ is 0.8. In the equivalent flat model, this probability is incorporated into all the probabilities that connect

the children states of $is1$ and $is2$, for example the path from $ps1$ and $ps2$ to $ps3$ or $ps4$. However, the original transition probability between the two internal states, which reflects an important process of a higher order than a transition between two production states, cannot be extracted from the flat model and would be lost unless the original hierarchical structure of the model is restored. Using the flat version of the model only (i.e. a flat model that does not incorporate any information of hierarchical structures) would mean an additional loss of information regarding the probabilities between the different levels of the hierarchy, which provide an additional view of the internal structure of a data set.

Due to the flattening procedure, the definitions of both the FB and the Viterbi algorithms for hierarchical models are identical to the algorithms for flat models. However, the EM algorithm, used for training models, has to be adapted to the hierarchical structure. After scanning through the training data, the EM algorithm re-estimates a model's structural parameters, i.e. its horizontal, vertical and emission probabilities. The transition probabilities are adjusted so that the training data will receive the highest possible overall probability, thus representing the data's properties through a model's parameters. Additionally to the horizontal transitions between the production states, parameter re-estimation needs to take the vertical transitions and the horizontal transitions between internal states into consideration. The algorithm for the flat HMM is not sufficient and needs to be adapted to a hierarchical model structure, as outlined in (Wierstra, 2004).

4.4 Hierarchical IOHMMs

Although HMMs and HHMMs provide a rather effective means of modeling and learning (hierarchical) structures in data, the parameters in these types of model are adjusted in an unsupervised way by the EM algorithm. HHMMs extract dependencies on a local as well as a larger scale, i.e. inside each level and between the levels. Sometimes however, it might be helpful to add some supervision or constraints into the process of parameter learning and estimation; for the learning of musical structures this means that it might be helpful to use the constraint of

one level of parameters to learn the next level. Synchronous IOHMMs[5] offer the constraint of mapping the models' observation sequences to a given input, thus modeling the dependencies of two sequences (see Figure 4.3) and conditioning the output to an input sequence (Bengio & Frasconi, 1995). The probability of entering state Q at time t is dependent on the input at time t and the state at time $t-1$; the output at time t is dependent on both the state and the input at time t. In musical data, this added feature allows us to learn the interdependencies of the different levels of musical structures.

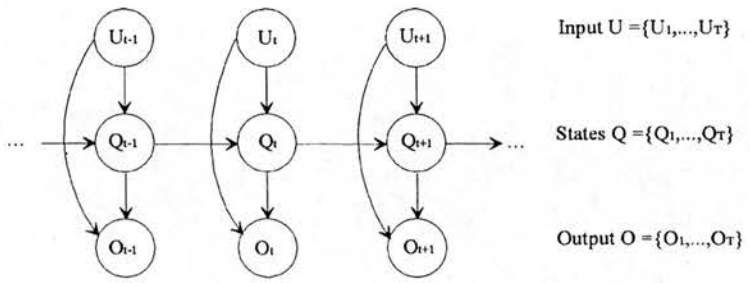


Figure 4.3: *Dependencies between input, output and states in an IOHMM.*

Using the improved approach for inference with HHMMs and the idea of IOHMMs, we introduce the concept of hIOHMMs, combining the power of hierarchical structures in HHMMs with the constraint of an added input layer in IOHMMs. The hIOHMM λ has the topography of an HHMM with an underlying input sequence $U = (U_1, ..., U_T)$[6]; both vertical and horizontal transitions, as well as the observations, are directly dependent on the input sequence. The input sequence U is divided into partial sequences over all the levels of the model; at the production state level, single input symbols are matched against the states and their observations (see Figure 4.4). Time flow in an hIOHMM is similar to HHMMs - time only increases after an observation symbol has been emitted from a production state

[5] In a synchronous IOHMM, the input and output sequences need to consist of the same number of elements and every input element is mapped against an output element. An asynchronous IOHMM allows the input and output sequences to be of different lengths. It is possible to imitate the behaviour of an asynchronous model by introducing a "null" symbol in the sequence alphabets, therefore only synchronous models are discussed here.

[6] Notation: T is the length of the input or output sequence, or the size of the training data set.

because every output symbol has to be matched against a symbol of the input sequence, i.e, on horizontal transitions out of a production state. During any other transitions through the model time stands still.

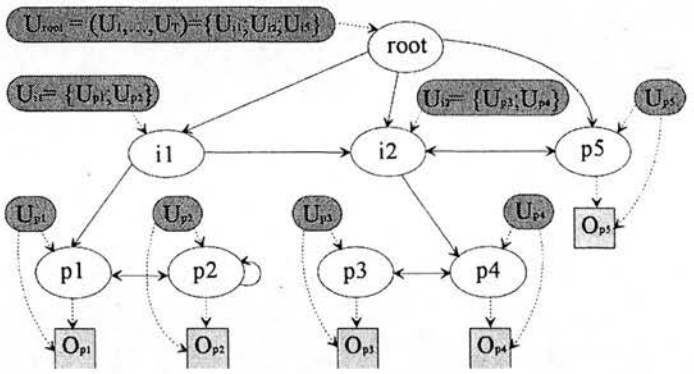


Figure 4.4: *Example of a three-level hIOHMM (end states are omitted for clarity). Solid lines show the transitions between states, dashed lines show the dependencies between input, output and states.*

The vertical probabilities π define the transitions between a parent i and a child state j, given a symbol U_t from the input alphabet U:

$$\pi^i_j(U_t) = P(j, U_t \mid i, U_t) \tag{4.9}$$

As time does not pass when states are activated vertically, the child states are conditioned on the same symbols as their parent states. The definitions for the horizontal transition probabilities differ for production states and internal states because only after an output symbol is emitted does the model look one step ahead, i.e. it looks at the next input symbol, and chooses the following state accordingly. Transitions between states Q at production state level are defined as

$$a_{ij}(U_t) = P(Q_t = j \mid Q_{t-1} = i, U_t) \tag{4.10}$$

i.e. the probability of being in production state i at time $t-1$ (represented by $Q_{t-1} = i$) and moving to state j ($Q_t = j$, either a production or an end state) given

the input U at time t, thus moving a step forward in the input sequence. The horizontal transition probabilities between internal states are defined as

$$a_{ij}(U_t) = P(Q_t = j \,|\, Q_{t-1} = i, U_{t-1}) \tag{4.11}$$

i.e. both states j and i are dependent on the same input symbol. Emission probabilities at time t are conditioned on the input symbol at time t:

$$b_i(O_t \,|\, U_t) = P(O_t \,|\, U_t, Q_t = i) \tag{4.12}$$

Flattening a hIOHMM allows the use of the algorithms for sequence probability and best state sequence calculation defined for IOHMMs. When the model is in minSR form, the new transition probabilities for the equivalent flat model can be calculated for every symbol in the input alphabet U:

$$a_{ij}(u)^* = a_{i,end}(u) \cdot a_{p(i),p(j)}(u) \cdot \pi_j(u) \tag{4.13}$$

i.e. the product of the transition probabilities between state i and the corresponding end state, the parent states, and the vertical transition to state j, given input u.

Similarly, the vertical transition probabilities have to be transformed into initial activation probabilities for all the production states:

$$\pi^*_{q^D}(u) = \pi^{q^{D-1}}_{q^D}(u) \cdot \pi^{q^{D-2}}_{q^{D-1}}(u) \cdot \ldots \cdot \pi^{q^{root}}_{q^2}(u) \tag{4.14}$$

i.e. the product of all the vertical probabilities on the path from the root state to the production state q^D (D being the production state level of the model).

The FB and the Viterbi algorithms for the hIOHMM are similar to the algorithms defined for the IOHMM, using the flattened equivalent of the hIOHMM. However the learning and re-estimation of the model's parameters with the EM algorithm need to be adjusted to the novel structure.

As part of the EM algorithm, the forward variables α and backward variables β need to be processed. The α terms, the probabilities of observing a partial sequence $O_1, ..., O_t$ given state i at time t and an input sequence $U_1, ..., U_t$ are defined thus:

$$\alpha_1(i) = \pi_i(U_1) \cdot b_i(O_1 \mid U_1) \qquad t = 1 \tag{4.15}$$

$$\alpha_t(i) = \left[\sum_{j=1}^{N} \alpha_{t-1}(j) \cdot a_{ji}(U_t) \right] \cdot b_i(O_t \mid U_t) \qquad 1 < t \leq T \tag{4.16}$$

The overall probability of an observation sequence O given a model λ and an input sequence U is calculated by summing over the terminal forward values:

$$P(O \mid \lambda, U) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.17}$$

The predictive power of a hIOHMM can be measured with the cross entropy, the negative average logarithmic likelihood per symbol (Shannon, 1948; Allan, 2002), which provides a measure of probability and predictability of a sequence given a certain model (see Chapter 3.2). It is estimated by summing over the negative logarithmic probability of every symbol in a sequence, given the model, and dividing by the length of the entire sequence:

$$H(O \mid \lambda, U) = -\frac{1}{T} \sum_{i=1}^{N} log_2 \alpha_T(i) \tag{4.18}$$

The backward variables β, the probabilities of observing a partial sequence $O_{t+1}, ..., O_T$ given state i at time t and an input sequence $U_{t+1}, , U_T$ are defined similarly:

$$\beta_T(i) = 1 \tag{4.19}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}(U_{t+1}) \cdot b_j(O_{t+1} \mid U_{t+1}) \cdot \beta_{t+1}(j) \qquad 1 \leq t < T \tag{4.20}$$

Equations 4.15 to 4.20 are taken from (Bengio & Frasconi, 1995) and are provided here for ease of reference. Based on (Wierstra, 2004), we define the EM algorithm for a hIOHMM as follows.

Starting from the production states of the hierarchical model, the first step is to calculate the state transition probabilities ξ, i.e. the probabilities of being in state i at time t and moving into state j given the input symbol U at time $t+1$:

$$\xi_t(i,j|U_{t+1}) = \frac{\alpha_t(i) \cdot a_{ij}(U_{t+1}) \cdot b_j(O_{t+1}|U_{t+1}) \cdot \beta_{t+1}(j)}{P(O|\lambda,U)} \tag{4.21}$$

The state occupation probabilities γ, the overall probabilities of being in a state i at time t, can then be calculated using the state transition probabilities. On production state level these values are processed by summing over all the probabilities of exiting production state i and entering production state j (notation $j \in Pr$):

$$\gamma_t(i) = \sum_{j \in Pr} \xi_t(i,j|U_{t+1}) \tag{4.22}$$

For the state occupation probabilities of internal states, the hierarchical structure of the hIOHMM needs to be taken into consideration; the probability of being in state k is the sum of all the probabilities of being in one of the descendant states of k[7] and moving into a non-descendant of k ($j \in d(k)$):

$$\gamma_t(k) = \sum_{i \in d(k)} \sum_{j \notin d(k)} \xi_t(i,j|U_{t+1}) \tag{4.23}$$

Using those definitions, the re-estimation equations for horizontal transition probabilities are:

$$\hat{\alpha}_{ij}(U_m) = \frac{\sum_{t=1}^{T-1} \sum_{k \in d(i)} \sum_{l \in d(j)} \xi_t(k,l|U_{t+1} = U_m)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{4.24}$$

$$\hat{\alpha}_{i,end}(U_m) = \frac{\sum_{t=1}^{T-1} \sum_{k \in d(i)} \sum_{l \notin d(p(i))} \xi_t(k,l|U_{t+1} = U_m)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{4.25}$$

[7]Notation: $i \in d(k)$. A descendant is any state that can be reached through vertical transition. If k is a production state, $d(k) = k$.

The new horizontal transition probabilities between two production or two internal states i and j have to be computed for every symbol U_m of the input alphabet and are calculated by summing over the descendants of i and j respectively. However, the horizontal transition probabilities between an internal or production state i and an end state need to be processed by summing over i's descendants and the non-descendants of i's parent state, thus reflecting the exiting of a sublevel of the hierarchical structure.

The vertical transitions from i to j are re-estimated by dividing the sum of the state occupation and state transition probabilities for the child state j by the sum of those same probabilities for the parent state i:

$$\hat{\pi}^i_j(U_m) = \frac{\sum_{k\in d(j)} \gamma_1(k) + \sum_{t=1}^{T-1} \sum_{k\notin d(i)} \sum_{l\in d(j)} \xi_t(k,l \mid U_{t+1} = U_m)}{\sum_{k\in d(i)} \gamma_1(k) + \sum_{t=1}^{T-1} \sum_{k\notin d(i)} \sum_{l\in d(i)} \xi_t(k,l \mid U_{t+1} = U_m)} \tag{4.26}$$

The re-estimation of the emission probabilities is not dependent on the hierarchical structure of the hIOHMM, therefore the equation for the flat model is sufficient. The new emission probability for symbol k from state i is the probability of being in state i and observing symbol k given input m, divided by the overall probability of being in state i:

$$\hat{b}_i(O_k \mid U_m) = \frac{\sum_{t=1, O_t=O_k, U_t=U_m}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.27}$$

The EM algorithm shown above is only valid for synchronous hIOHMMs. The input and output sequences need to have the same number of elements, as every output symbol needs to be matched against an input symbol.

4.5 Summary

HHMMs are an extension to standard HMMs, which allows the hierarchical and layered representation of complex data. Every HHMM that is in minimal self-referential form can be transformed into a flat HMM to reduce the computing time necessary for inference and training. Although both types of models are equivalent in terms of the predictions that they make, the transformation into a flat model

needs to be temporary as the hierarchical representation and the structural probabilities on intermediate levels are lost in the flat HMM. The chapter then introduces the novel concept of hIOHMMs, which allow to map input to both a hidden state sequence and an observed output sequence, outlining the definitions and algorithms necessary for inference and training.

Chapter 5

Music Representation and Model Structures

Probabilistic models are used to learn the structural probabilities that define a data set which reflects the knowledge of a specific part of the real world. The representation of the explicit knowledge has to be chosen carefully so as to be unambiguous and precise. The quality of a representation directly influences a model's ability to extract dependency probabilities. However, the choice of model structure is just as important as the type of knowledge and its method of representation. The numbers of states, layers and transitions in a hierarchical model are crucial to its performance and to the training procedure's success in capturing the characteristics of a data set. The implications of choosing the right representation and model structures, as well as the choices made for this project, are outlined in sections 5.2 and 5.3.

5.1 Music Corpus

The music corpus used for this research is a set of 384 Bach chorales[1]. The chorales are available both in MIDI and text format, with the text files containing information on metre, rhythm, phrasing, harmonies and pitch for the four parts of a chorale, i.e. soprano, alto, tenor and bass. Table 5.1 shows the distribution of modes and

[1] Available from ftp://ii1ftp.ira.uka.de/pub/neuro/dominik/midifiles/bach.zip

time signatures among the chorales.

	major key	minor key	
4/4	181	162	343
3/4	25	15	40
6/4	0	1	1
	206	178	384

Table 5.1: *Distribution of Bach chorale time signatures and modes.*

There are a number of reasons for choosing Bach chorales as a training corpus for this project. The chorales have been the basis of many research projects that concentrate on the analysis or generation of music with computers. Therefore it is possible to compare directly the results of different approaches, which are all based on the same material. Additionally, Bach chorales are complex enough to provide a challenging and interesting basis for a machine learning approach, yet the compositional rules behind them are well established and understood. Most Bach chorales are very short, with an average length of around 24 to 40 bars, however there are a high number of chorales available. For a project that aims to learn large-scale structures in music this certainly proves to be an advantage, because it allows the training of models with a high number of complete works, thus making it possible to extract the high-level dependencies that define entire works of music.

5.2 Music Representation

Although the Bach chorales are available in text form, the text cannot be used directly as input for the HMMs. The information contained in the text has to be extracted and transformed in order for it to become suitable training data for the models. As this research tries to use minimal *a priori* knowledge of musical structures only, part of that transformation process includes the ident fication of the minimum of information necessary for the machine learning approach to be workable. Additionally, although the improved algorithms for HHMMs reduce the computational

expense considerably, a small and efficient representation vocabulary helps minimise the size of the training sequences and thus reduces the algorithms' runtime. In order to capture the large-scale dependencies of the events inside the musical structures, it is necessary that the representation allows for the distinction of the timing of those events, i.e. the relative place of an event in a score needs to be included in the representation of that event.

The representation of the metrical structure has to include not only the number of beats per bar, thus defining a time signature, but also the relative place of a beat inside a bar. It is necessary to be able to distinguish between strong beats and weak beats, or the beginning or end of a bar. The most simple way of representing the relative place of a beat is by numbering each beat according to its position. The accuracy of the metrical structure can be varied by using different sampling rates. For instance, a bar in a `4/4` metre is represented as `1234` if the sampling rate is a crotchet, however it becomes `12345678` for the more precise sampling rate of a quaver.

Using the standard pitch symbols (`C`, `C#`, `Db`, `D`,...) is a simple and unambiguous way of representing pitch structure. In order to the keep the number of possible symbols, and thus the computational expense during training at a minimum, we decided against using register information together with pitch. Each voice in a chorale only has a range of approximately one and a half octaves, therefore the additional information provided by the register is negligible. If a model can learn the probabilities connecting the sequence `B4 C5 D5`, it can extract the same probabilities from a more general pitch sequence without register. The harmonic structure of a chorale is implied by the pitch structures of the four voices. As part of this research aims at modelling the interdependencies of different structures, such as the pitch structures of the soprano and bass parts, the harmonic structure is not represented independently.

The phrasing structure needs to define the beginning and the end of each phrase in a chorale, as well as the end of a chorale. A phrase is highlighted by an opening and closing bracket, '`(`' being the beginning and '`)`' being the end of a phrase. The

end of a chorale needs to be marked not only to separate the chorales in the training data but also to help reinforce the importance of the relative place of a phrase inside a chorale. The symbol # is used as 'end of chorale' marker.

Similar to the harmonic structure, the rhythmical structure is not represented independently, but rather is implied using metre and pitch with the sampling rate of the metrical structure providing a basic time unit. If the sampling rate is a quaver, a pitch of a duration of a minim is four 'beats' long. In order to distinguish between a pitch that is attacked on the beat and a pitch that is held over from a previous beat, the latter is represented in square brackets, `[]`. For example, `C` with a duration of a crotchet at a sampling rate of a quaver is denoted `C [C]`, while it would simply be represented as `C` at a sampling rate of a crotchet.

The expert knowledge included in this representation of music is mainly limited to the importance of the timing of events. The representation allows for the possibility to learn distinct characteristics of musical structures by mapping those characteristics to a relative time in a bar, phrase or entire score.

5.3 Choice of Model Structure and Initialisation

An important factor in the process of training a HMM is the choice of the model's structure, i.e. the number of its states, layers, and transitions. However, there is no simple, correct way of choosing the structure (Rabiner, 1989). In order to obtain the best possible results from training a model, its structure has to reflect certain structural properties of the training data. For musical structures, this means that a model should have the possibility to learn the importance of time, as outlined in previous paragraphs. The model's structure should also represent the hierarchical structures in music, for instance the division of an entire piece into phrases, which in return are subdivided into bars.

Choosing a model structure at random without taking the nature of the training data into consideration most definitely results in a poor performance of the model. Additionally, an intelligent choice of structure means that the size of a model, which

directly influences the runtime of the inference algorithms (see section 4.1), can be kept in reasonable boundaries. It is important to point out however, that a model with a larger number of hidden states is likely to provide a better approximation of the training data than a model with a smaller number of states, because the characteristics and properties of the data can be mapped to a larger number of states. This is however not necessarily better for the overall representation of the musical structures, because this can result in overfitting. Additionally, given that very large models require a considerable amount of computational power, it is necessary to find a balance between model size, number of states and and compute time in order to keep the models at a manageable level.

The amount of domain specific knowledge used to construct the models for this research is limited to the minimum necessary to make the models workable. The models that are used in this research for instance reflect the basic phrasing structures of a piece of music, having separate internal states to represent the phrases at different points in time. This allows the models to distinguish between separate phrases and extract time specific information. The models' structures are thus influenced by the (explicit) expert knowledge that phrases, and the events inside them, have different functions and internal structures depending on the time of their occurrence inside a piece of music.

The model parameters are the transition and observation probabilities. When creating a new model, these parameters need to be given initial values that are the starting point from which the training process re-estimates new probabilities that fit a training data set. A danger for the estimation of such new probabilities is that the algorithm might not find a globally optimal solution, because any initial values push the calculation towards a local optimum. In order to avoid actively reaching a local optimum and therefore a sub-optimal solution, it was decided to initialise all model parameters through even distribution of transition and observation probabilities.

5.4 Summary

The representation of the knowledge domain and the structure of a model are both essential parts in the process of devising a system that can learn structural dependencies and probabilities using a minimal amount of expert knowledge. A poor representation of domain knowledge is likely to result in a suboptimal performance by the system. This research provides the composite networks with a small amount of domain specific knowledge, thus helping the models to extract those probabilities and dependencies that are structurally important.

Chapter 6

Software Design and Implementation

This chapter will describe the process of designing and implementing the software that was used throughout this research project to simulate the hierarchical models and composite networks. The differences in the implementation of a HHMM and an IOHHMM are minimal and their design follows the same structure, therefore only the HHMM implementation is described here.

6.1 Motivation

Very early on in this research project it was decided to develop new software that implements the hierarchical models we wanted to use, rather than reusing and expanding on existing tools. The main reason for this decision was that at that point in time, there existed no software which already implemented the HHMM algorithms. A number of software packages provided tools builiding HMMs, e.g. the Hidden Markov Model Toolkit (HTK)[1], but none of these packages contained an implementation of the hierarchical extension of the models. Having to choose between expanding on existing software or developing new software, it was decided that it would be in the best interest of the project to create a new tool, which would be custom built and could address all the requirements of the research.

[1] http://htk.eng.cam.ac.uk/

6.2 Requirements and Design

The first step of the design process is the identification of the requirements for the software and the definition of the basic functionalities. The functionalities involve inputting an initial model; using the HHMM algorithms on an input data set based on the model; generating new results; and storing the results and new model structure. As the models will be used in a feed-forward network, it is a requirement that all the information resulting from one model can be accessed and read by another model. The tool needs to implement all the HHMM algorithms and has to be able to build a model that follows a user-defined structure. The definition of a model's structure should be simple, so that many different types of models can be built and tested.

The software needs access to two types of information: firstly, the data sets that are needed for input (the musical structures that are used for training); secondly, the information that is necessary to build the initial model. This information is reusable and accessible outside the lifetime of the program. Any results generated by the program, as well as any changes made to the initial model, have to be stored. The format of the data input and output are an essential part of the design of the software.

Training a model is computationally very expensive and, given the same data sets, the reuse of probability distributions from previous training runs increases the ease of use for HHMM. The generation of a large number of results from many different production runs and initial model settings makes it an essential requirement, for both testing and verification purposes, to be able to easily reproduce scenarios that caused certain sets of results. The tool therefore should be able to create a model from input files that describe a certain model, as well as write the results of this model's training into output files of a similar format. The output files can then later be used to recreate the model and its state after training on a given data set. The information for the initial set-up of a model includes the basic model structure, such as the number of states and distribution of states over the different levels, and the initial probability values for the state transitions and emissions of symbols.

Three different platforms (Solaris 9, Mac OSX and Windows XP) were used during this research, and the portability of the code was important. It was decided to implement the software in Java[2], because this allowed for easy an migration of the code between machines. Additionally, a Java graph library called JGraphT[3], which implements mathematical graph-theory objects, is freely available. The library supports a large number of graph types, including directed, weighted graphs. The library was used as a basis for the HHMM code, making use of the available graph objects and graph manipulation algorithms and methods. Details of the implementation of the code will be given in the next few paragraphs.

6.3 Implementation

As mentioned earlier, the functionality of the HHMM tool can be divided into three basic parts: data input, computation, and data output. The data input is concerned with preparing and reading the data set, i.e. the musical structures, as well as getting the information on the initial model set-up. The computation involves the HHMM algorithms, i.e. the Forward-Backward, the Expectation-Maximisation and the Viterbi algorithms, as well as the flattening and rebuilding of the hierarchical model structure. Finally, the output handles storing the generated musical results and a model's parameters after training on a given data set. This section will briefly describe the implementation of the code for each of these three basic parts.

6.3.1 Input

The musical data that the model uses needs to be extracted from existing data files. The Bach chorales that were used here are available both in MIDI and in ASCII format. Given that the information that is included in the text files is sufficiently detailled, is was possible to extract the necessary data directly from these files by using a regular expression parser with pattern matching. The parser was implemented as a Java class as part of the main code, and can optionally extract metri-

[2]Java version 1.4

[3]http://jgrapht.sourceforge.net/

cal, phrasing or pitch structures. Pitch information that is extracted by the parser is automatically transposed into C major (or A minor respectively) to create a normalised data set. Figure 6.1 shows the first few bars of a Bach chorale in ASCII format. The file's header defines the chorale's name and key, as well as the key signature. The music is represented in seven columns: phrase, bar, the four voices (soprano, alto, tenor and bass) and the harmony. The regular expression parser scans through the file and extracts the needed metrical, pitch or phrasing information together with the key and key signature. Any pitch information is transposed according to the chorale's key.

Additionally to the musical data, the program requires access to the initial structure and parameter setup for any model it wants to build. This configuration data comprises information such as number of levels in the hierarchical model, number of states per level and each state's status (internal, productive or end state), and the initial values for the transitions between the states and the emission of observation symbols. The initial data is stored in a set of ASCII files and is read by the program at runtime.

6.3.2 Computation

The code is organised in an object-oriented design, separating the implementation of the model from the implementation of the computation algorithms and other utilities, such as the parser. The user specifies the path to the input files, which contain a model's parameters and structure, at runtime. Once these files have been read by the program, the initial model is created and the main code interacts with the user through a command line interface, from which a user can navigate through a choice of menu options that can be executed by the code: the three main algorithms (FB, EM and Viterbi); the generation of musical data and the size generated data; an option that allows to print the current state of the model, i.e. print a snapshot of the parameters that define the model at a certain point in time; and an option to exit the program. The code runs in a loop and returns to the main menu each time a computation has been executed until the user actively quits the program.

As mentioned earlier in this chapter, the implementation of the graph structures

```
Choralname = bch001
Anzahl Stimmen = 4
Tonart = A-dur
Takt = 4/4
Tempo = 100
Notentextausgabe in 16tel-Schritten:

PHRASE  TAKT     SOPRAN  ALT   TENOR  BASS    HARMONIK

    1            C#2    A 1    E 1   A -1     T


         1       C#2    A 1    E 1    A 0     T
                 D 2
                 E 2    H 1    E 1    G#0     D3


                 C#2    A 1    E 1    A 0     T


                 A 1    A 1    E 1    F#0     S3
                                D 1
         2       H 1    G 1    D 1    G 0     SS
                        F#1
                 C#2    E 1    A 0    A 0     T


                 D 2    F#1    A 0    D 0     S


    2            C#2    E 1    A 0   A -1     T


         3       H 1    E 1    G#0    E 0     D  7
                        D 1
```

Figure 6.1: *Beginning bars of a Bach chorale as represented in ASCII format. The content of the file is separated into a header, which holds key and key signature information, and a body, representing the actual musical structures that make up the chorale.*

was based on the JGraphT library, which contains a large set of methods for the manipulation of directed graphs. For instance, an efficient implementation of Dijkstra's "Shortest Path" algorithm, which allows the quick and easy calculation of the shortest path between two states in a model, is available as part of the library. The library's vertex objects are used as a basis for the states of a model, the edge objects represent transitions.

The algorithms are implemented as standalone classes, using the definitions presented in Chapter 4. The calculations performed in these algorithms mostly involve very small values, thus risking arithmetic underflow. On top of leading to false results in arithmetic operations, underflow can potentially cause severe disruptions to a program, such as runtime errors or hardware interrupts. In order to alleviate this problem, a scaling step that uses state-independent scaling coefficients was added to the calculations (implementing the scaling definitions in (Rabiner, 1989)). The scaling ensures that the floating point operations remain stable and accurate.

6.3.3 Output

The program directs its main output into files. This output consists of the generated data and the state of the model after training. The generated musical data is written into files that are defined by the user at runtime. These are simple text files that can easily be parsed into a music notation format, such as ABC[4] or MusicXML[5].

The second type of output is the snapshots of the state of the models after training. Once the EM algorithm has calculated a new set of parameters that best represent the model relative to the data set, this set of parameters is directed to a file. There are two reasons for creating these output files: firstly, it is helpful for any future setup of models to be able to see how the EM algorithm alters the initial setup of a model. However, the more important reason is that these files provide the user with the ability to recover a model's state after training. The training process is often computationally expensive, therefore saving the state of a model after this process

[4]The ABC notation is used to write music in ASCII format. Software for the transformation of ABC files in, for instance, MIDI format are freely available.

[5]MusicXML is an XML-based music notation format. Software for the manipulation of MusicXML files is readily available.

reduces the risk that comes with any software or hardware failure after training.

6.3.4 Testing

The correctness of the implementation of the algorithms was tested using the Hidden Markov Model Toolbox for Matlab[6], which provides a well tested implementation of the basic algorithms for HMMs. Given that HHMMs use the algorithms for HMM in large parts, the FB, Viterbi and part of the EM algorithms could be tested using this software. A number of artificial test data sets were created and run on basic models using both the HMM Toolbox and our own implementation of the algorithms, thus allowing the comparison and verification of results. As there was no existing implementation of a hierarchical HMM available at the time of the research, the conversion of the hierarchical model to the equivalent flat model (and vice versa) had to be tested manually by writing small test units that controlled each step of the algorithm. Once the test results were satisfactory on the artificially created test data sets, the implementation was checked on the real data, ensuring the robustness of the code.

6.4 Summary

This chapter gives an overview of the implementation of the HHMM code, written in Java 1.4, justifying its design and the reasons for a new implementation. The basic structure of the program is explained together with its use of data input and output. The following chapter will show how HHMMs and composite networks were used to generate musical data.

[6]http://www.cs.ubc.ca/ murphyk/Software/HMM/hmm.html

Chapter 7

Generation of Music with Composite Networks

Musical works have several defining levels of representation, or musical parameters, such as metre, duration, phrase structure, cadential patterns or pitch (see Chapter 1). Large-scale musical structures can be described in terms of several layers of such representations (Ebcioglu, 1993; Conklin & Witten, 1995). The concept of generative networks of probabilistic models proposed here uses the idea of 'layering' those parameters; by starting from one parameter layer, which is used as the basis for an entire piece, and adding more layers on top of the basis, complete structures are generated. For instance, using metrical structure as the basis for a piece, the layers rhythm, phrasing and pitch can be added to create a pitch structure that incorporates all four of these parameters. A single HHMM can represent several of those representation levels simultaneously; an HHMM that models metrical structure can represent bars, phrases and sections on its different levels, modelling the interdependencies of the parameters inside a musical work (Weiland, Smaill, & Nelson, 2005). A network of models, namely a combination of HHMMs and hIOHMMS, can thus learn the interdependencies of parameters that would not be represented in one model, such as for example melody, pitch and bar structure or phrasing structure.

In order to show the HHMMs and hIOHMMs performance on learning the statis-

tical regularities that define musical material, networks with a variety of topographies are built. They are used to create simple two-part and four-part works with appropriate phrasing, rhythms and passing notes, based on and according to the information that can be extracted from the musical data set used to train the networks. If the model networks are able to generate musical material that is believable on a large scale and respects the standard rules of harmony, e.g. if they can for instance create melodies that progress towards a cadence, this will show that the most important structural regularities that define the music (in the training set) can be extracted and represented by probabilistic models.

7.1 'Random Walk' Method

The process that generates new material from a trained model uses a 'random walk' approach: based on structural probabilities, the process walks through the model, making decisions on transitions and emissions using random numbers. Preference will thus be given to events with high probabilities, however chance may decide to choose events with low probabilities as well. The 'random walk' method has been described as "greedy" as it does not guarantee to produce results with high overall probabilities (Conklin, 2003). However, the models should preferably generate a large number of results, all of them representing the structural probabilities extracted from the training material, but also distinctly different from each other. Simply generating the most likely result at each stage of a network is a very limiting approach. The random walk method allows a cheap and quick generation of large numbers of results, and their overall likelihood given the model can be easily computed. The random numbers used in the decision making process ensure that the results are diverse and reflect different levels of quality.

7.2 Generative Networks

The generative networks of models are built from a combination of HHMMs and hIOHMMs. In order to create a musical work from scratch using the layering of structures, one of the structures has to constitute the basis for the entire work. A HHMM does not require an input sequence to work from, therefore it can be

used as the basis, i.e. the bottom layer, of a network. The data generated from the HHMM will then serve as input sequence for a hIOHMM in the network. A simple network for the creation of melody and bass lines could, for instance, consist of an initial HHMM that generates the harmonic structure, which then serves as an input for a hIOHMM that builds a bass line on top of the harmonic structure. This bass line could then in return serve as input for a hIOHMM that writes a melody given a bass line. Each layer in the final work is thus not only dependent on the layer upon which it has been built, but also on the previous layers. Considering the previous example: as the bass line is directly dependent on the harmonic structure, and the soprano is dependent on the bass, the soprano is implicitly also dependent on the harmonic structure. The models in a network are all trained separately on data from the same group of chorales. Only after training are they used as a feed-forward network. The fact that the models are trained independently allows them to be reused and to change the structures of networks dynamically. For instance, a HHMM that models metrical structure and that has been trained on a set of 50 chorales can be used in any network that uses this specific set of chorales, as the training procedure and its results are dependent on the training data only.

7.3 Smoothing vs Non-Smoothing Models

Input-output models map observation sequences to input sequences. Both input and output symbols have to be part of defined alphabets of valid events. For alphabet sizes of, for example, 50 symbols each (where each output symbol can be mapped against one input symbol), this results in $50 * 50 = 2500$ possible pairs of input/output symbols. However, it is likely that not all of those symbol pairs are represented in the training data. Those symbol combinations that do not occur in the training material are assigned zero probabilities during training. Additionally, not all symbols might be interconnected by valid transitions, and thus not every sequence made up from symbols of the input alphabet of a trained model is necessarily automatically a valid input sequence for that model.

For instance, if a model that was trained on the data

```
1 2 3 4 1 2 3 4     input
A B C D A B C D     output
```

was fed a new input sequence `'1 2 3 1 2 3'` to generate data from. Considering the symbols only, this sequence is correct. However, because the subsequence `'3 1'` is not in the training data, this input sequence will fail to run through the model successfully, because there will be no transition that allows a `'1'` to follow a `'3'`. The result can be a model that is very likely overfitting, i.e. that tends towards memorizing the training data. A way of dealing with overfitting is through using larger training sets or by smoothing the training process, i.e. assigning small probabilities to events that do not occur in the training data (see Section 7.3).

The degree of overfitting is influenced not only by the size of a training set, but also by the sizes of the input and output alphabets, with small alphabets implying a higher probability of occurrence for all possible events, and the sparseness of the training data. Data from natural language follows Zipf's law (Manning & Schütze, 1999):

$$f \propto \frac{1}{r} \Rightarrow f \cdot r = k \tag{7.1}$$

i.e. the frequency f of a word in a text is inversely proportional to its rank r and a constant k describes the relationship between frequency and rank. For instance, the 10^{th} most common word should have a frequency that is two times higher than the 20^{th} most common word. This implies that the frequency distribution is such that there are few words with very high frequencies and a large number of words with very small frequencies, resulting in data sparseness. If frequencies and rank form a near straight line when plotted in a double logarithmic graph, the data's frequency distribution follows Zipf's law. The frequency distributions of events in data sets outside natural language can also be compared to Zipf's law. For instance, Figure 7.1 shows the ranks and frequencies of the soprano and bass pitch symbols.

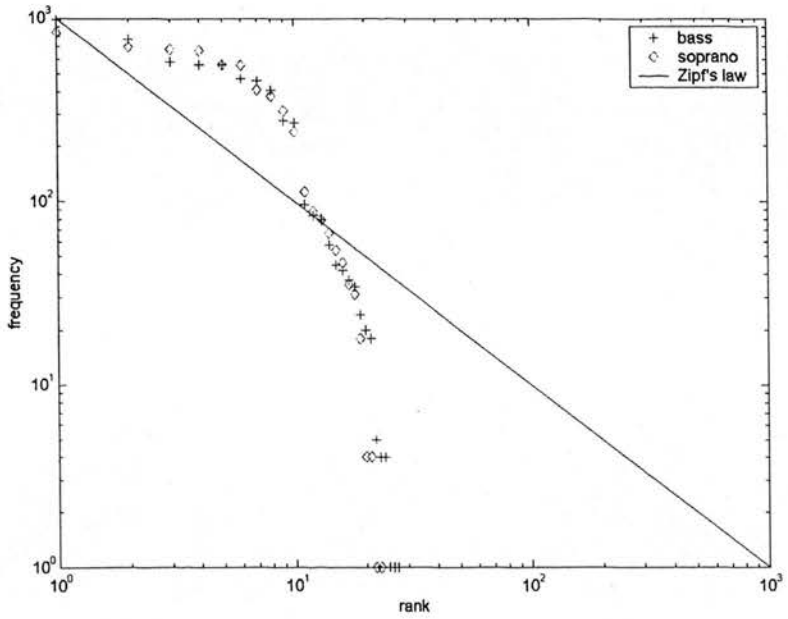


Figure 7.1: *Rank and frequencies of bass and soprano pitch symbols from 80 chorales (5880 symbols), compared to Zipf's law.*

The frequencies of the pitch symbols are taken from 80 chorales[1], sampled at crotchet rate. There is a total of 5880 pitch symbols per voice and 47 different symbols are possible (additional to 42 pitch symbols, there are 2 rest symbols and 3 phrasing symbols)[2] 27 of the bass events and 23 of the soprano events have non-zero frequencies, with the first 13 events having frequencies higher than suggested by Zipf's law (the highest ranked events for bass and soprano have frequency counts of 977 and 835 respectively). The frequencies of the remaining events however, rapidly drop to low numbers. The fact that 42.5% of the bass symbols and 51% of the soprano symbols are not represented in the frequency chart suggests that smoothing would be advisable. However, if these symbols have not

[1] All of the 80 chorales are in a major key with a 4/4 time signature. They were transposed to C major to achieve a relative pitch profile.

[2] The pitch symbols are 7 letters (C, D, E, F, G, A, B) used with three accidentals (natural, sharp and flat), representing notes both in normal and continued form. Similarly, the rest symbols are in normal and continued.

occurred in 80 chorales and a total of 5880 events, how important are they in the representation of the musical structures? A symbol with a frequency count of 1 certainly has an important function inside the bar, phrase or chorale to which it belongs, however its global function is minor.

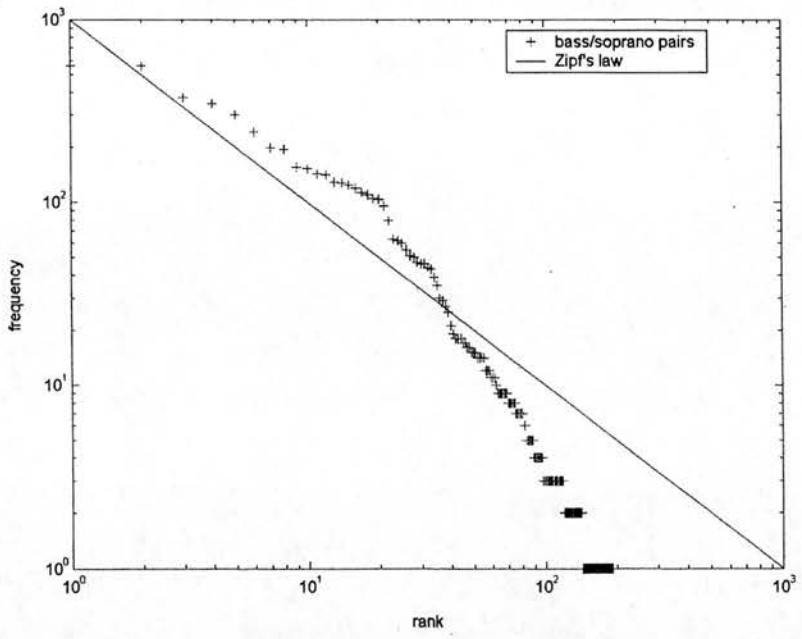


Figure 7.2: *Rank and frequencies of bass/soprano pitch pairs compared to Zipf's law.*

Figure 7.2 shows the more interesting distribution of input/output pairs of bass and soprano pitches, taken from the same set of chorales. The logarithmic frequency/rank ratio of the pitch pairs follows a line closer to Zipf's law. One cause is the much larger alphabet of symbols, which increases to 1939 possible symbols ($44 * 44 = 1936$ pitch symbol pairs, plus three pairs of phrasing symbols). Of all possible symbols however, only 192 have non-zero frequencies, i.e. approximately 90% of all possible symbol combinations are not represented in the training data. This can be explained in part by the fact that certain pitch pairs simply would not occur in a Bach chorale. For instance bass/soprano pitch pairs such as `Cb/B#` or

`E#/Ab` will not be found in a chorale, but they are included in the frequency count as valid combinations of two pitches. Pairs of symbols might have non-zero frequencies, however that does not guarantee transitions between them. As mentioned earlier, if a transition between input symbols is not represented in the training data, it will also reflect on the performance of the IO models by rejecting certain symbol sequences. The implications of overfitting are thus greater for IO models than for general HHMMs: not only might it not be possible to emit certain symbols because they have a zero frequency count in the training data, but the transitions between output symbols are directly dependent on the validity of the input symbol sequences.

Thus the following question arises: is smoothing a necessity for a successful generative model, or does it deflect attention, i.e. redistribute the probability space, away from the important structural regularities of the training material? In order to evaluate this question the models described in the following paragraphs are both used with and without smoothing, enabling the comparison of results and of the potential merits or drawbacks of both approaches.

7.3.1 Adjusting the Smoothing Parameter λ

The value of the smoothing parameter λ influences the generation of new musical structures with a trained model. Using the events for the 80 chorales sampled at crotchet rate (5880 separate events) as an example, Table 7.1 shows how the probabilities of events from the possible observation set that do occur in those 80 chorales are affected by introducing smoothing.

The table contains the probabilities for the events using their absolute frequency in the given data set, as well as the adjusted probabilities using Lidstone smoothing, with the smoothing parameter being set to different values. It can be noted that the probabilities are gradually reduced for the ten most frequently observed symbols and are increased for the less frequent events as well as those events that are not observed at all in the data set. In this case, even for the larger values of λ, the resulting increase or decrease in the probabilities for each symbol are very small. However, the overall effect is that out of 47 possible symbols, the probabilities of

Symbol	Frequency	P(λ = 0)	P(λ = 0.1)	P(λ = 0.3)	P(λ = 0.5)	P(λ = 1)
C	977	16.615	16.604	16.581	16.558	16.500
G	770	13.095	13.086	13.069	13.051	13.008
A	583	9.915	9.908	9.896	9.884	9.853
(	560	9.523	9.517	9.506	9.494	9.465
)	560	9.523	9.517	9.506	9.494	9.465
E	471	8.010	8.005	7.996	7.986	7.963
D	459	7.806	7.801	7.792	7.783	7.761
F	405	6.887	6.884	6.876	6.868	6.850
B	275	4.676	4.674	4.670	4.666	4.656
[C]	269	4.574	4.572	4.569	4.565	4.555
[G]	96	1.63	1.633	1.633	1.634	1.636
F#	84	1.428	1.429	1.430	1.431	1.434
#	80	1.360	1.361	1.362	1.363	1.366
[A]	58	0.986	0.987	0.9891	0.990	0.995
G#	45	0.765	0.766	0.768	0.770	0.776
Bb	42	0.714	0.715	0.717	0.719	0.725
C#	37	0.629	0.630	0.632	0.635	0.641
[E]	34	0.578	0.579	0.581	0.584	0.590
[F]	24	0.408	0.409	0.412	0.415	0.421
[D]	20	0.340	0.341	0.344	0.347	0.354
Eb	18	0.306	0.307	0.310	0.313	0.320
[Bb]	5	0.085	0.086	0.089	0.093	0.101
[F#]	4	0.068	0.069	0.073	0.076	0.084
[Eb]	4	0.068	0.069	0.073	0.076	0.084
[B]	1	0.017	0.018	0.022	0.025	0.033
[C#]	1	0.017	0.018	0.022	0.025	0.033
[G#]	1	0.017	0.018	0.022	0.025	0.033

Table 7.1: *Absolute and relative frequency of bass pitch symbols in 80 Bach chorales, sampled at crotchet rate, depending on the value of λ. This table only shows those pitch symbols that do occur in the chorales. For the 19 pitch symbols that do* not *occur, the percentages are: P(λ = 0) = 0, P(λ = 0.1) = 0.017, P(λ = 0.3) = 0.0051, P(λ = 0.5) = 0.0085,P(λ = 1) = 0.017.*

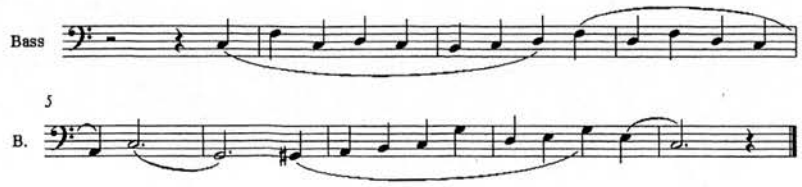


Figure 7.3: *Bass line generated on top of a simple metrical structure, with no smoothing being applied to the model. Most of the pitch symbols are part of the set of observations that occur frequently in the training data.*

the ten most frequent symbols are reduced and the value are redistributed to rare or non-occurring symbols.

The danger of picking a value of λ that is too high is that, when generating new structures using the random walk method, the generation process might start choosing too many low probability symbols and become stuck in these low probability events. The reason for this is that the smoothing process is applied to a model after training and thus the transitions between symbols that occur rarely (or never) in the data set are not re-estimated. This effect can be illustrated using a simple example of a bass line at crotchet sampling (using the data set reflected in Table 7.1) being generated on top of the following basic metrical structure:

```
(4 1 2 3 4 1 2 3) (4 1 2 3 4 1) (2 3 4 1 2 3) (4 1 2 3 4 1 2 3) (4
1 2 3) #
```

The Figures 7.3, 7.4, 7.5 and 7.6 show examples of bass lines generated with models that use the values $\lambda = 0$, $\lambda = 0.1$, $\lambda = 0.3$ and $\lambda = 0.5$ respectively.

When setting the parameter λ to 0, i.e. no smoothing, the models generate new structures that will only make use of symbols that are in the training data set (see

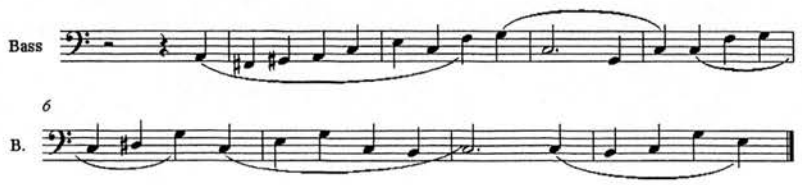


Figure 7.4: *For this example, the smoothing parameter* λ *was set to* 0.1. *As in the previous example in Figure 7.3, most of the events still occur frequently in the training data, however the occurrence of a* `D#` *shows that a symbol from outside the training data set can now be chosen.*

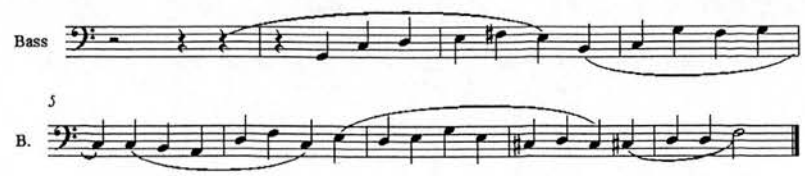


Figure 7.5: λ *was set to* 0.3 *for the generation of this bass line. The (initially) zero probability events are now chosen more frequently, the bass line starts on two pauses for instance, a symbol that is not present in the training data set.* `C#` *and* `[F#]` *are also two low frequency events.*

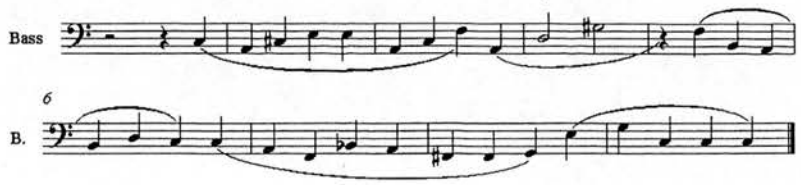


Figure 7.6: *The smoothing parameter λ was set to 0.5 for the generation of this bass. This is an example of the generation process becoming stuck within low probability events: the second phrase has a pitch sequence [D] G# [G#] P. The actual frequencies of these events in the training set are 20, 45, 1 and 0 (out of 5880 events).*

Figure 7.3). The bass line in Figure 7.4 shows the effect of a small amount of smoothing, with λ being set to 0.1. The majority of the represented symbols are still high probability events from the training set, however zero probability events are now possible. Figures 7.5 and 7.6 show how higher values for the smoothing parameter can result in a proportionally large number of low probability events even in short sequences, due to the generative process becoming stuck inside transitions between unlikely events. This can be observed especially in the second phrase of the example in Figure 7.6. Given observations when testing the effect of the level of smoothing on the generative process, it was decided that an appropriate, and with regards to musical structures successful, value for the smoothing parameter would be 0.1.

7.4 Model Performance and Number of States

As mentioned previously in Chapter 5.3, a model with a larger number of hidden states is more likely to perform better when learning the characteristics of a given data set than a model with a smaller number of hidden states. The more hidden states a model has, the more it has the possibility to distinguish between different

properties and structures in a data set, single them out and represent them with probabilities. Figure 7.9 shows the top two levels of a model, with the second level giving a representation of phrases in musical structures. In this example, four hidden states describe all the phrases in a piece of music. Underneath such phrase states, on a lower level of the hierarchy, additional states represent musical structures of a higher granularity, for instance bars, motives and single musical events.

Taking the example from Figure 7.9, different phrase representations were experimented with when choosing an initial structure for the model. The more hidden states are available for the representation of the notion of a musical phrase, the more accurately each phrase in a data set will be learnt by the states and the model. It was thought to be important that a model should, as a minimum requirement, be able to distinguish the beginning and end phrases of a piece of music from any other phrases. Thus it was decided that a minimum of three states should be assigned to describe phrases. However, to increase accuracy, the number of states needs to be higher; experiments were run with models from three to six phrase states. The models with both three and six phrase states were rejected early; out of the models with four and five phrase states, the "four states" model was retained. Six states proved to make the overall model too large, increasing the computation of the training step to nearly twice the duration for that same step in a model with four states. The model with four hidden states available for phrasing information proved to be a good compromise between accuracy (i.e. more accurate than three states) and computation (i.e. less compute intensive and more manageable with regards to duration).

7.5 Two-Part Music

The design of a network that has to be capable of learning and generating two-part music, i.e. soprano and bass, is decided based on the musical structures that are needed to construct the music. Each voice can be represented by a layering of pitch structure, metrical structure, phrasing structure, rhythmical structure and harmonic structure. Figure 7.7 shows an example of some of the layers that can be extracted from a melody.

Figure 7.7: *Simple melody separated into its pitch, rhythmical and metrical structures.*

One of these structures, or possibly a combination of them, needs to constitute the basis for the generative network upon which the remaining structures can be built. The quality of the foundation structure will thus be passed on and reflected in further structures. The quality of the basis is important, thus choosing a structure that is relatively easy to learn and reproduce seems to be the best choice. Using harmonic structure as the foundation layer would be a musically sound choice. However, learning the harmonic structures of musical training material is a difficult task that has been addressed in a number of research projects (see chapter 2). It is an even more challenging task to create harmonic structures from scratch, therefore harmonic structure is not an optimal solution for a solid foundation of a generative network. On the other hand, metrical structure is made out of repetitive patterns of beats in bars, which can be represented easily by a probabilistic model. Metre follows strict repetitive patterns, e.g. the 2^{nd} beat of a bar will always follow the 1^{st} beat, thus metrical structure does in itself contain little information that a HHMM would need to extract. However, when combining metrical structure with phrasing structure, which can easily be represented together with the metre, a model can extract important features such as the lengths of phrases and the numbers of phrases in a piece, or the relative places in a bar in which phrases begin or end. Figure 7.8 shows how metrical and phrasing structures can be coupled.

The metre/phrasing structure can be learnt by a simple HHMM. The initial structure of the HHMM should allow the model to learn the properties of each of the

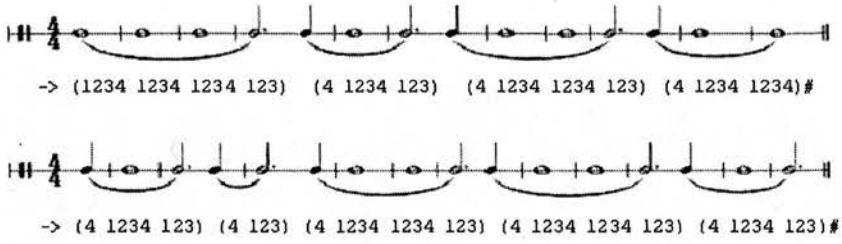


Figure 7.8: *Representation of metrical structure with added phrasing information.*

different phrases. The middle level is therefore divided into four internal states, assuming that every Bach chorale has a minimum of four phrases, and one production state. The production state on that level represents the end of a chorale; the internal states represent the phrases in a chorale. The middle level has a 'left-right' structure: the model is forced to start at the first phrase state and traverse the remaining states until it reaches the 'end of chorale' state. The first and last phrase states are not self-referential, i.e. there can only be one start phrase and one end phrase. On the lowest level the production states hold the observations, i.e. the phrasing and metre symbols. Each phrase state is superordinate to one set of production states representing the phrasing and metre information. After training the model with a non-smoothing EM algorithm, superfluous transitions are removed from the model by being assigned zero probabilities. A smoothing model would not remove these transitions, but merely assign them very low probabilities. Figure 7.9 shows the structure of the phrasing level of the model after training with a non-smoothing algorithm.

After learning and generating the basic framework of metre, bars (as implied by the metrical structure) and phrases, hIOHMMs will add the next layers of structures. A bass line together with a melody imply underlying harmonies, therefore it was decided that learning the harmonic structure is not strictly necessary if the relationships between the voices are modelled accurately (see section 5.2). A hIOHMM that represents the dependencies between metre/phrasing and pitch should therefore be sufficient. Figure 7.10 shows two possible types of networks to model both the bass and soprano parts. In the first network, the bass is directly dependent on

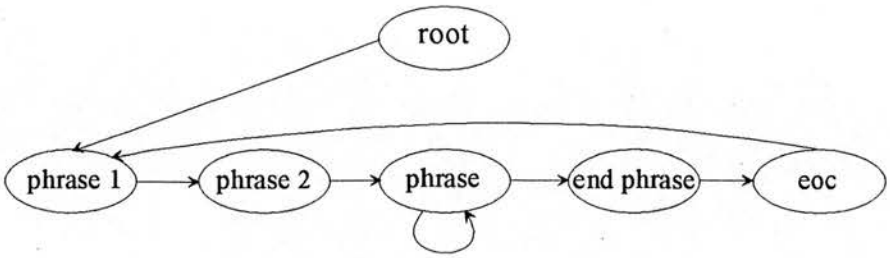


Figure 7.9: *Structure of the top two levels of the metre/phrasing HHMM after training (with a non-smoothing EM algorithm). The self-referential loop on the second phrase state, present in the initial model structure, was removed by the training process.*

the metrical structure and will be constructed first. The soprano is in turn directly dependent on the bass. In the second network the roles are reversed and the soprano is the first part to be constructed, followed by the bass.

Both network designs are equally valid and musically plausible, the first reflecting the creation of a melody on top of a bass line, whereas the second implies the harmonisation of a melody. Both types of networks were used here in order to be able to evaluate which sequence of models best captures the information held in the training structures. The initial structures of the models are similar to the metre/phrasing model. The hIOHMMs that represent bass and soprano also have four phrase states on the second level, each of them holding the production states that represent the pitch symbols. The phrasing information that was used in the metrical structure is added onto the pitch structure, therefore the phrasing symbols are also part of the hIOHMMs input and observation alphabets. Each absolute pitch is represented by one production state per phrase, i.e. both `B#` and `C` are observations from state `C`. Table 7.2 shows how the transition probabilities between events are influenced by their relative placing. The table lists the transition probabilities from the pitch `C` to any other pitch or phrasing symbol, given the input `4`, i.e. the probabilities of moving from pitch `C` on beat `3` to the next event on beat `4`.

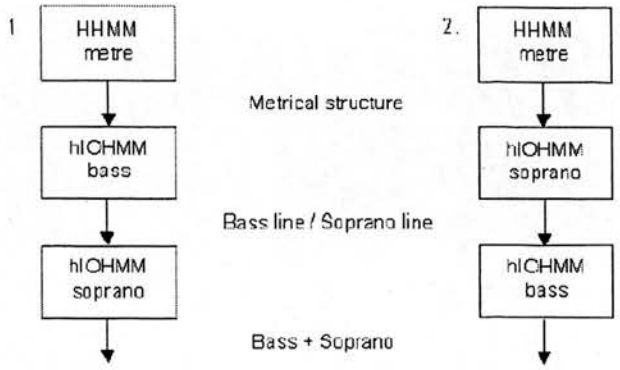


Figure 7.10: *Possible network structures for the creation of two-part music. (1) The bass line is modelled as being directly dependent on the metrical structure, while the soprano is directly dependent on the bass. (2) In the second network, the places of bass and soprano are reversed, with the soprano part being constructed before the bass part.*

Figure 7.11 is an example generated by a metre/bass hIOHMM[3]. Both the input and observation sequences, i.e. the metrical structure and the pitch structure, relate to each other in a hIOHMM that models the dependency between metre and bass. The sampling rate in this example is a crotchet, therefore the maximum value for the beat count in a 4/4 metre is 4. As every pitch symbol is dependent on a metre symbol, which denotes the relative place of an event in a bar, the regularities of the relative placing of pitch symbols can be extracted together with occurrence frequencies and transition probabilities. This allows for some degree of distinction between the different levels of importance of events on strong or weak beats (see section 2.1).

After the first voice (either bass or soprano) is created, the second part needs to be added. The final model in the network, a pitch/pitch hIOHMM, is trained on soprano/bass (or bass/soprano) sequences and the model's parameters are adapted

[3]No register information is included in the pitch representation (see section 5.2). However, in order to visualise the results with staff notation, the pitches are used with a register typical for the voice that is being represented.

phrase 1		phrase 2		phrase 3		phrase 4	
to	$P(\cdot)$	to	$P(\cdot)$	to	$P(\cdot)$	to	$P(\cdot)$
F	0.0447	C	0.0273	F	0.0125	G	0.0212
E	0.0582	F	0.0407	F#	0.0350	F	0.0425
A	0.0624	E	0.0637	C	0.0551	C	0.0425
C	0.0727	B	0.0863	B	0.0562	A	0.0851
B	0.1100	A	0.0960	D	0.0710	B	0.0851
G	0.1258	G	0.1300	A	0.1211	[C]	0.7234
D	0.2581	D	0.1568	E	0.1353		
[C]	0.2680	[C]	0.3990	G	0.1480		
				[C]	0.3657		

Table 7.2: *Transition probabilities $P(\cdot)$ in a (non-smoothing) metre/bass hIOHMM. The model was trained on 80 chorales in a major key, with a 4/4 time signature and crotchet sampling rate. The probabilities represent the transitions* ***from pitch C given input 4*** *in all of the phrase states, i.e. the probabilities of moving from the pitch C on the third beat of a bass line to a new event on the last beat in a 4/4 bar.*

to the training material's structural probabilities. The result from the first hIOHMM in the network is then fed into the final model as the input sequence. The result of using the bass line from Figure 7.11 as input for a bass/soprano hIOHMM is shown in Figure 7.12.

Music that was generated from a model that uses training data sampled at crotchet rate will lack passing and neighbour notes. A separate hIOHMM could be used to add those extra notes to an existing pitch structure. The model scans through the pitch structure, which is used as input, and decides, based on its parameters, whether additional notes should be inserted between already existing ones. All the notes of the pitch structures occur on the beat, as only those pitches have been extracted for the training data. For the passing notes hIOHMM however, the pitches that occur after a beat are also taken into account and all the groups of notes that appear inside one beat will be added to the output symbol alphabet. For instance,

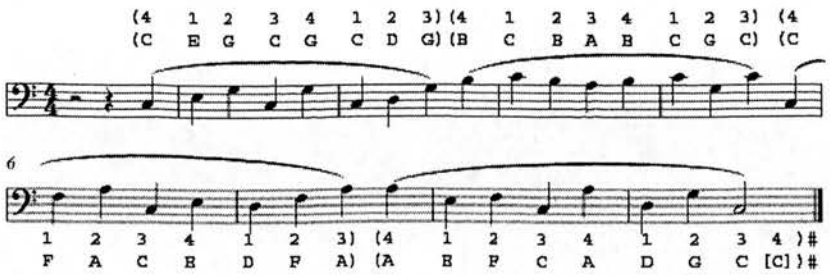


Figure 7.11: *Simple bass line, generated by a (non-smoothing) metre/bass hIOHMM, annotated with metrical, phrasing and pitch structures. In a hIOHMM that models bass pitch depending on metre, the metrical structure would serve as the input, the pitch structure would be the observation sequence.*

consider a bass line that contains the pitch context C E; the output alphabet of the hIOHMM will include the pitches that appear in the training set immediately between C and E. The hIOHMM can, but does not have to, insert a passing or neighbour note between two pitches by changing the pitch at time $t+1$ into a group of pitches, where the last element has to be the original pitch. Using the example of the notes C and E, the model can either decide to keep this progression unchanged, or to replace the E by the group DE, FE, DFE or similar.

The model that creates the passing notes is structured differently from the previous models. Although the results would probably benefit from a differentiation of phrases similar to the previous models, a high number of observation symbols (due to the many different patterns of passing notes) makes it preferable to use a smaller model. There is only one phrase state defined for this hIOHMM, resulting in the model not being able to distinguish between the beginning, middle and end of a piece. Figure 7.13 is an example of a bass line with added passing notes.

7.6 Four-Part Music

The networks designed to learn and generate two-part music accomplish their tasks successfully. It is therefore a natural choice to 'upgrade' those networks to repre-

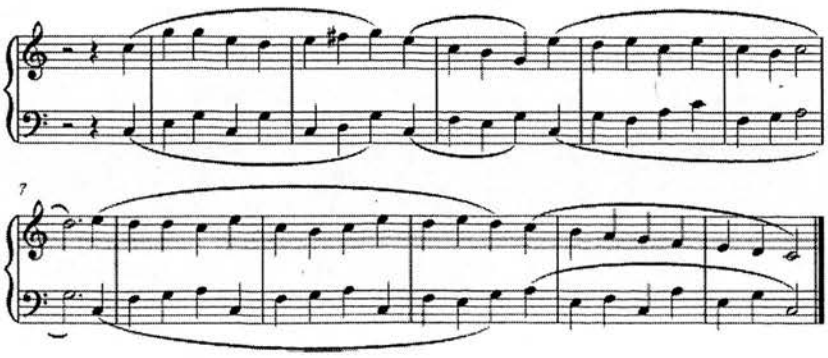


Figure 7.12: *Example of a new two-part piece generated by a (non-smoothing) network, using the bass line from Figure 7.11.*

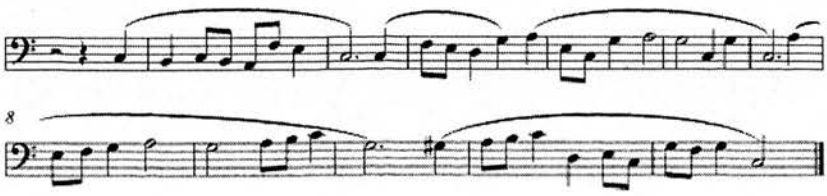


Figure 7.13: *Bass line with passing notes added by a hIOHMM. The model takes a context of two beats from the original (crotchet based) line and decides whether or not to insert additional notes between the beats.*

sent four-part chorales. With bass and soprano already represented with underlying metre and phrasing, the two remaining voices can be added through pitch/pitch hIOHMMs, making them dependent on the already existing voices. The basic initial structure of the models themselves does not change (see Figure 7.9). The major choice that has to be made when designing four-part networks is the order of dependencies in the network. For instance, if the bass is directly dependent on the metrical structure, and the soprano is dependent on the bass, the soprano is *implicitly* dependent on the metrical structure *through* the bass. However, implicit dependence is not as strong and influential as direct dependence. The composite nature of the networks allows us to experiment with different network structures. Figures 7.14 and 7.15 show two types of four-part networks that were used here. The first network uses a simple feed-forward architecture: the tenor is dependent

on the alto, which in return is dependent on the soprano. Assuming that the implicit dependency between the observations of two models becomes weaker with every step through the network, this design would result in a tenor part that is only very weakly dependent on the bass line for instance.

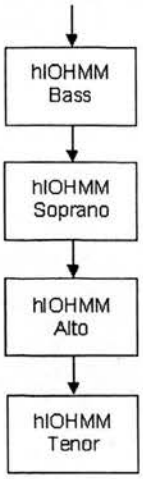


Figure 7.14: *Simple feed-forward structure for a four-part network (B-S-A-T). The tenor is directly dependent on the alto, which in return depends directly on the soprano.*

In order to obtain a possibly stronger dependency between all four parts, the distance between the models in the network can be reduced. The network structure in Figure 7.15 makes both the alto and the tenor dependent on the soprano, thus the distance between bass and tenor in the network is smaller. The drawback however is that, in this type of network, the connection between alto and tenor exists only via the soprano part.

The final network architecture that was used for four-part chorales is shown in Figure 7.16. In this network, bass and soprano are created in separate steps and models, however they are then used *together* as input for a model that creates both alto and tenor. Only one sequence can act as input, and only one observation sequence can be created, thus two pitch structures have to be merged into one:

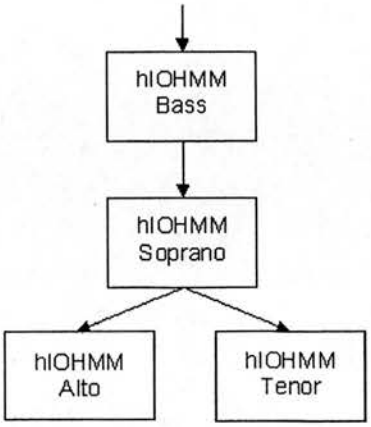


Figure 7.15: *Network structure in which both the alto and the tenor part are made directly dependent on the soprano (B-S-A-S-T).*

B S B S B ... *bass/soprano input*
A T A T A ... *alto/tenor output*

For instance, given the bass line CEFG and the soprano EGAG, the resulting input sequence becomes C E E G F A G G. In the observation sequence, the alto and tenor pitches are placed underneath the bass and soprano pitches respectively. Thus, every alto pitch is directly dependent on the bass and every tenor pitch is dependent on the soprano. The vertical connection, i.e. the connection between input and output symbols is valid. However, the horizontal connection, i.e. the connection between the symbols in either input or output, is more problematic. The model learns from a training set that uses merged input and output sequences, thus it can extract the properties of the data and the transition probabilities between the events of different voices. However, because the models used here are of the first order, i.e. in every point in time t the model only looks one step back at the immediately preceding event at time $t-1$, the direct dependencies between the pitches of one voice cannot be captured. For instance, when generating a new tenor pitch, the model will not be able to directly look back at the last event in the tenor, but it can only do so via the latest alto pitch. This will certainly lead to some obvious

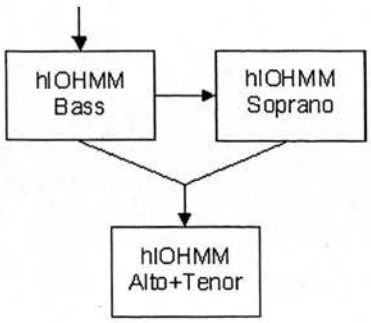


Figure 7.16: *BS-AT network which first creates bass and soprano separately, and then merges both voices together to use as one input sequence. The final model in the network generates both the alto and tenor part.*

mistakes in the material generated from this network, however the results are still likely to reflect some of the structures of the training data. A model of the second-order, which at time t takes events from both times $t-1$ and $t-2$ into account, would resolve the problem. However, as second-order hIOHMMs are beyond the scope of this research[4], we use the first-order models and try to assess this network's merits in comparison with networks introduced previously.

All of the network designs presented here were trained to learn and generate large-scale musical structures. They were used with smoothing and non-smoothing EM algorithms, and used musical data sampled at both crotchet and quaver rates. The results and the performance of the different types of network are evaluated and discussed in the following chapters.

7.7 Summary

Generative composite networks are a collection of different types of HMMs, in which the output from one model is fed into the next model, thus layering the struc-

[4] In first-order HHMMs (and hIOHMMs), the transition probabilities are held in two-dimensional matrices A_{ij}. For a second-order model however, the matrices have to be three-dimensional, i.e. A_{ijk}, to reflect the dependency of the current event on two, rather than one, previous events. Therefore the definitions of the inference algorithms would have to be adapted to using three-dimensional matrices, which is beyond the scope of this research.

tures that result from each stage of a network. This chapter introduces the concept of composite networks as used in the research and shows how these networks can be used to learn from existing data as well as generate new data. The potential advantages and drawbacks of using non-smoothing and smoothing models with musical data are highlighted. The basic structure of a model inside a network, as well as several different network structures are laid out to demonstrate the flexibility and adaptability of the composite network concept. The following chapter will evaluate a number of musical examples that were generated using networks of the types introduced here.

Chapter 8

Evaluation of Results

A network's ability to accurately model musical structures is reflected in the quality of the results it can generate. The evaluation of the results is divided into two main parts: a subjective evaluation, achieved mainly with a listening experiment, and a more objective measurement of how well the training data is learnt, using the cross-entropy measure and music theoretical rules to assess the models' ability to learn to musical data sets, the quality of the results and the merits of the composite network approach.

8.1 Listening Experiment

A listening experiment is an excellent method for receiving subjective feedback on musical works. Rather than concentrating on objective measures of quality, it highlights how the music is perceived. In order to get an overview of how well the different types of networks perform on both two-part and four-part music, the experiment was split into two sections. With a total of 27 subjects[1] available for the entire experiment, 13 subjects participated in the experiment for two-part music, while the remaining 14 subjects were assigned to the study of four-part examples. The number of subjects participating in this experiment is small, however all the subjects are musically trained and able to provide the desired subjective feedback. A greater number of subjects would certainly deliver more accurate results, as the

[1] Music students from the Royal Scottish Academy of Music and Drama (RSAMD) in Glasgow, UK.

impact of each individual's assessment is reduced the larger the number of subjects that are delivering assessments. If there is only a small number of participants available, a single subject's feedback has increased impact on the overall results, potentially reducing the significance of those results. The aim of the listening experiment was to establish an overall picture of the subjective perception of the machine generated musical structures. Every question in the experiment uses a control value, i.e. an original Bach chorale, to judge the significance of the participants' assessments.

The musical examples used in the experiment were generated by a range of different types of networks, thus allowing for a performance comparison of the various network designs. The test examples are chosen as to reflect a cross-section of the data generated by different models and different composite networks. The number of examples in the experiment is kept small for a number of reasons: firstly, listening experiments require a high level of concentration from all the participants and the duration of such an experiment should therefore be kept short; secondly, each musical example in the experiment should be assessed from more than one point of view, i.e. more than one question should be asked about that example. Both points led to the decision to have a small number of questions and musical examples, which are used throughout the experiment.

Aiming to get a broad view of how the different musical examples are perceived, six questions had to be answered for each section of the experiment:

- Is the piece an original Bach chorale?
- Does the piece progress towards a perfect cadence?
- Are the phrases well-structured throughout and do they progress towards a cadence?
- Is the piece harmonically well-structured throughout?
- Is the piece similar in style to a Bach chorale?
- How would you assess the overall quality of the piece?

For each question, four musical examples were played to the subjects[2], who then had to answer the question for all four examples. The examples were played only once; after all the examples for one question had been listened to, the subjects were allowed thirty seconds to finalise their answers before moving on to the following question. One original Bach chorale was part of each group of examples. The result given for the Bach chorale acts as the question's control value for remaining examples, i.e. the quality of the answers given by the subjects is dependent on the answers given for the original chorale, which is expected to always receive the highest score. In an effort to achieve a balanced evaluation, the subjects were not told that they were being presented with exactly one original chorale per question. The following sections provide a detailed analysis of the results of the listening experiment.

8.1.1 Two-part examples

The musical examples for either section of the listening experiment should reflect a variety of different network types and sampling rates, and also represent smoothing and non-smoothing models. The following examples were used in the first part of the experiment (all the examples can be found in staff notation in appendix A):

(2P1) Bach chorale K181, BWV 133, crotchet
(2P2) B-S, no smoothing, crotchet
(2P3) B-S, no smoothing, quaver
(2P4) S-B, smoothing, crotchet
(2P5) Bach chorale K209, BWV 145, quaver
(2P6) B-S, smoothing, quaver
(2P7) S-B, no smoothing, crotchet
(2P8) B-S, flat HMMs, no smoothing, quaver
(2P9) Bass added to melody of chorale K211, BWV 367, taken from (Allan, 2002)
(2P10) B-S, no smoothing, passing notes added separately

Examples 2P1 and 2P5 are the original Bach chorales that will act as the control values for the questions. Example 2P9 is taken from (Allan, 2002): the soprano is

[2]The music was encoded in MIDI format and played to the subjects from a CD, thus ensuring a uniform performance.

the original chorale melody, the bass was added on with a flat HMM. All except one of the remaining examples were generated by first creating the bass line, then adding the soprano. Example 2P8 was generated with a network of standard, flat HMMs in order to enable a direct comparison between the results of hierarchical models and flat models. Two different sets of training chorales were used for examples sampled at either quaver or crotchet rate, the former consisting of 50 chorales (6511 events), the latter consisting of 80 chorales (5880 events).

The first question the subjects were asked, "Is the piece a Bach original?", aims both at evaluating the subjects ability to recognise an original Bach chorale and at getting a first overview of the quality of the computer generated results. Table 8.1 shows the results for this question for examples[3] 2P1, 2P3, 2P4 and 2P2. As expected, all the subjects correctly recognised the last example as being an original Bach chorale. Out of the three remaining examples however, only example 2P4 (see appendix A, Figure A.4), created by an S-B network with smoothing, is unanimously identified as being computer generated. Examples 2P2 and 2P3, both generated by non-smoothing models that add the soprano onto the bass, score relatively high with 30% and 38.5% of the subjects respectively mistaking those pieces for original chorales.

It was expected that the example created from a network with smoothing models would receive a lower score, because these models potentially allow any succession of events to occur, albeit with a very small probability. The low probability events that are selected by the smoothing models can thus result in examples that are further removed from the structures of the training set. Example 2P3 (see appendix A, Figure A.3), which received the highest score of the computer generated pieces, is the only example in this set that is sampled at quaver rate. The models that generated this piece were trained on a larger set of chorales, and thus were enabled to extract a larger amount of structural information, which in return is reflected in the results.

[3]The numbers of the examples refer to the list of examples at the beginning of section 8.1.1. The order in which the examples are listed in the tables is the order in which those examples were played to the subjects.

Is the piece a Bach original?				
	yes		no	
Example 2P3	5	= 38.5%	8	= 61.5%
Example 2P4	0	= 0%	13	= 100%
Example 2P2	4	= 30%	9	= 70%
Example 2P1	13	= 100%	0	= 0%

Table 8.1: *Example 2P1, the only Bach chorale in this set of examples, was correctly recognised as the only original work. The three remaining examples are computer generated, with only example 2P4 being identified as such by all the subjects.*

One of the aims of this research is the generation of large-scale musical structures, e.g. musical pieces that are well-structured overall and progress towards cadences. For questions 2 to 5 the musical examples were given ratings from *1* to *5*, with *1* being "not at all" and *5* meaning "very much so". When answering question 2 (see Table 8.2), the subjects were asked to specifically concentrate on whether the entire piece *progresses* towards a perfect cadence. The original chorale again receives the highest rating, closely followed by example 2P7, generated from an S-B network with no smoothing and crotchet sampling. Example 2P6, from a smoothing network with quaver sampling rate, scores considerably lower. Although the models in the quaver network can learn from a larger training corpus, this effect is weakened by the smoothing applied during training. Example 2P8 was created from a network that uses flat HMMs only, thus no hierarchical information is represented in those models, which are unable to distinguish the relative timing of events in musical structures. A piece generated from such a network thus does not progress towards a perfect cadence at the end, because the models are unable to distinguish between the separate phrases of a musical piece.

Questions 3 and 4 (see Tables 8.3 and 8.4) look at the melodic and harmonic structuring of the phrases that make up the musical examples. As expected, example 2P6 (smoothing B-S network, quaver rate) receives a similarly lower result than for the previous question. Examples 2P3 and 2P10, both sampled at quaver rate,

Does the piece progress towards a perfect cadence?						
	1	2	3	4	5	μ
Example 2P6	2	5	3	3	0	2.54
Example 2P8	13	0	0	0	0	1
Example 2P5	0	2	4	4	3	3.61
Example 2P7	1	1	5	5	1	3.30

Table 8.2: *The original chorale, example 2P5, receives the highest rating with a mean of 3.61, closely followed by example 2P7 at 3.30. Example 2P8, the only piece that was generated by a network of flat HMMs, receives an overall rating of 1, and thus is identified as not progressing towards a perfect cadence at all.*

received the highest rating respectively.

Are the phrases well-structured and do they progress towards a cadence?						
	1	2	3	4	5	μ
Example 2P9	1	3	7	2	0	2.77
Example 2P3	2	1	4	5	1	3.15
Example 2P6	1	6	3	2	1	2.70
Example 2P5	0	2	1	3	7	4.15

Table 8.3: *Overall low ratings for the computer generated examples, with only the piece generated from a non-smoothing network being judged to be slightly above average.*

The final two questions are aimed at providing a general overview of the quality of the musical examples. Whereas for question 5, "Is the piece similar in style to a Bach chorale?" (see Table 8.5), none of the computer generated examples receives a rating above average, the results for question 6[4] are more revealing (see Table 8.6): assessing the overall quality of the original Bach chorale with a rela-

[4]For question 6, the value 1 equals to the assessment "very poor", 5 equals "very good".

Is the piece harmonically well-structured throughout?						
	1	2	3	4	5	μ
Example 2P1	0	1	2	4	6	4.15
Example 2P8	2	3	7	1	0	2.54
Example 2P2	0	5	6	2	0	2.77
Example 2P10	0	2	4	7	0	3.38

Table 8.4: *Example 2P8, generated from flat HMMs receives the lowest score as anticipated. Example 2P10, generated with a crotchet based network and a passing notes model, receives the highest rating of the computer generated examples.*

Is the piece similar in style to a Bach chorale?						
	1	2	3	4	5	μ
Example 2P2	0	4	8	1	0	2.77
Example 2P1	0	1	2	4	6	4.15
Example 2P7	0	5	4	4	0	2.92
Example 2P4	4	7	1	1	1	2.30

Table 8.5: *Ratings below average for the computer generated examples, and the original chorale again obtains the highest score. The style of the Bach chorales thus seems to be reflected poorly in examples 2P2, 2P4 and 2P7.*

tively low mean value of 3.54, the next best example (2P3) scores an average result of 3.23. Although the mean values are not very high, it shows that the quality of both examples is perceived to be quite similar. Example 2P3 has obtained high ratings throughout the experiment, and these ratings are reflected and summarised in the results for question 2P6.

Certain examples receive high scores throughout the experiment, thus suggesting that they are strong representations of the musical structures in the training data. For instance example 2P3, which was mistaken to be an original chorale by 5 out of 13 subjects, also receives high scores in questions relating to the phrase structures

How would you assess the overall quality of the piece?						
	1	2	3	4	5	μ
Example 2P6	0	4	6	3	0	2.92
Example 2P5	0	2	4	5	2	3.54
Example 2P9	2	7	3	1	0	2.23
Example 2P3	1	1	6	4	1	3.23

Table 8.6: *The original chorale receives a relatively low score, albeit the highest score for the four examples. Example 2P3, which has consistently received high ratings throughout the experiment, obtains a a score similar to the Bach chorale.*

and overall quality. Additionally, this underlines the validity of the results given by the listeners, who attribute consistent levels of quality to the various examples.

8.1.2 Four-part examples

The results of the two-part experiment show that certain networks are more successful than others in capturing musical structures, the dependencies inside those structures and those between the different layers that make up a piece of music. Judging the harmonic and cadential structures, as well as the general quality of the interdependencies of the different voices of two-part examples can be difficult, especially when the musical examples are sampled at crotchet rate. The four-part listening study thus provides a deeper insight into how the musical examples are perceived.

The following nine examples were set up for the four-part listening study:

(4P1) Bach chorale K181, BWV 133, crotchet
(4P2) B-S-A-T, no smoothing, crotchet
(4P3) B-S-A-S-T, no smoothing, quaver
(4P4) Bach chorale K209, BWV 145, quaver
(4P5) B-S-A-S-T, smoothing, quaver
(4P6) B-S-A-S-T, no smoothing, quaver
(4P7) B-S-A-S-T, flat HMMs, no smoothing, quaver

(4P8) Harmonisation of chorale melody K211, BWV 367, taken from (Allan, 2002)

(4P9) BS-AT, no smoothing, crotchet

Most of the examples are built on the existing two-part pieces; example 4P9 uses the melody and bass of Bach chorale K211. Reusing the musical structures from the two-part examples allows for the comparison of the results of both the two-part and the four-part experiment.

The subjects were asked the same questions as in the two-part experiment. The first question again asks whether a musical example is an original Bach chorale. Similarly to the two-part study the subjects identify two of the examples as original and machine generated chorales respectively (though not unanimously), leaving the remaining two examples with a more mixed result. Built on the soprano and bass structures from the third example of the two-part experiment, example 4P3 is again mistaken to be an original chorale by a high percentage of subjects. Example 9 however receives a low score, although it uses the soprano and bass parts of Bach chorale K211 and could therefore have been expected to confuse the listeners.

Is the piece a Bach original?				
	yes		no	
Example 4P9	1	= 7.15%	13	= 92.95%
Example 4P3	6	= 42.9%	8	= 57.1%
Example 4P2	2	= 14.3%	12	= 85.7%
Example 4P1	13	= 92.95%	1	= 7.15%

Table 8.7: *Example 4P1 was almost unanimously identified as being the only original chorale. Example 4P9 uses the soprano and bass parts of Bach chorale K211, the alto and tenor parts were machine generated. Example 4P3, which is based on the same example of the two-part experiment, is mistaken to be an original chorale by 6 out of 14 subjects.*

The following two questions target the overall structures and the phrase structures of the examples, as well as their progression towards a cadence at the end of each phrase or piece. Both examples that were generated with flat HMMs, i.e. examples 4P7 and 4P8, are assessed with the lowest result in each question, with a mean score of 1.85 and 2 respectively. They are followed by the pieces that were generated using smoothing models, leaving examples 4P3 and 4P6 with the best scores among the machine generated examples.

Does the piece progress towards a perfect cadence?						
	1	2	3	4	5	μ
Example 4P5	3	3	8	0	0	2.35
Example 4P7	8	3	1	1	1	1.85
Example 4P6	0	3	6	4	1	3.21
Example 4P4	0	0	1	4	9	4.57

Table 8.8: *As anticipated, the example generated from the network that uses flat HMMs received the lowest score with a mean value of* 1.85. *The results for the piece generated with a smoothing network (example 4P5) are also considerably lower than the results for example 6, which was generated without smoothing.*

Question 4 aims to obtain an assessment of the harmonic structuring of the musical examples. Reflecting the results of previous questions, example 4P7 is seen to be of poor quality and example 4P6 receives a consistently high score.

The final question aims at the pieces' overall perceived quality as well as their quality in comparison to the original Bach style. Surprisingly, although example 4P3 was mistaken by 6 out of 14 subjects to be an original Bach chorale, it only scores a mean value of 2.71 for question 5, "Is the piece similar in style to a Bach chorale". It thus receives the same score as example 4P2, which was estimated to be a Bach original by only 2 out of 14 subjects. However, when asked about the overall quality of the pieces, examples 4P3 is again the machine generated piece with the highest score.

Are the phrases well-structured and do they progress towards a cadence?						
	1	2	3	4	5	μ
Example 4P8	5	6	2	0	1	2
Example 4P3	1	4	3	7	0	3.28
Example 4P4	0	0	2	2	10	4.57
Example 4P5	3	2	8	1	0	2.50

Table 8.9: *Example 4P3 was mistaken by 6 out of 14 subjects to be an original Bach chorale, which is again underlined by this relatively high result in the assessment of the quality of the phrase structures. Both examples 4P5, which was generated by smoothing models, and 4P8, taken from (Allan, 2002), are considered by the subjects to be of a poorer quality.*

Several results can be drawn directly from the listening experiment. One clear observation is that musical examples generated from systems formed with flat HMMs are perceived to be of poorer quality than those generated with models and networks that make use of the hierarchical structures in musical material. The pieces generated with the flat models consistently received the lowest scores in each group of examples, and they were assessed to be of especially low quality for questions that targeted the overall structure of a piece and its progression towards a final cadence.

Additionally it can be extracted that examples created with non-smoothing models received higher scores overall than those examples from smoothing models. Non-smoothing models are stricter when generating structures, because transitions and progressions that do not exist in the training corpus are not allowed in the generation process. Finally, the choice of network composition also seems to influence the quality of the generated results. In the two-part experiment, the examples generated with an S-B network receive slightly lower scores than those generated with a B-S network. This can be explained thus: in Bach chorales bass parts are often

Is the piece harmonically well-structured throughout?						
	1	2	3	4	5	μ
Example 4P7	2	8	3	1	0	2.21
Example 4P6	0	1	4	8	1	3.28
Example 4P1	0	1	1	2	10	4.50
Example 4P9	2	6	4	1	1	2.69

Table 8.10: *Example 4P6 receives a consistently high score, whereas example 4P7 is again judged to be of poor quality.*

Is the piece similar in style to a Bach chorale?						
	1	2	3	4	5	μ
Example 4P6	0	2	7	3	2	3.35
Example 4P1	0	0	3	3	8	4.35
Example 4P3	0	8	4	0	2	2.71
Example 4P2	2	5	4	2	2	2.71

Table 8.11: *Example 4P3 receives a surprisingly low score which is not in keeping with previous results for this examples. Example 4P6 is seen to be the machine generated example that is closest in style to an original Bach chorale.*

more varied than soprano parts (see chapter 7.3), using a larger number of different possible events. They are therefore slightly more difficult to represent and learn with a machine than soprano lines. In the case of a B-S network, the bass line is matched against the metrical structure, conditioning the possible bass events to a relative place in a bar. The number of metrical events is very small, thus matching the bass structure to the metrical structure successfully narrows the event space for the bass. However, if the bass is added after generating the soprano (and is therefore directly dependent on the soprano), the number of possible bass events in every generative step is larger, thus possibly leading to the generation of structures that are of poorer quality.

How would you assess the overall quality of the piece?						
	1	2	3	4	5	μ
Example 4P5	0	8	4	1	1	2.64
Example 4P8	4	6	3	1	0	2.07
Example 4P3	0	2	7	5	0	3.21
Example 4P4	0	0	2	2	10	4.57

Table 8.12: *Example 4P3 is back to receiving the highest result out of the machine generated examples. Similar to the earlier results, example 4P8 is assessed with the lowest score.*

8.2 Measurement of Learning

The subjective evaluation of the musical examples generated with a variety of different networks highlights the listeners' preference of certain musical results, and thus points towards the networks which have generated those results. Each model's and network's ability to learn a musical data set can be measured more objectively by calculating the cross-entropy values of the training data and the newly created results. The measure of cross-entropy, which was introduced in section 3.2, allows to put an absolute value (given the model) on the accuracy with which a model learns the training data. However, we are not using the cross-entropy values in the standard Machine Learning approach. Cross-entropy values calculated on the training data gives us confidence that the program produces reasonable results. Lower values indicate a closer approximation of the training data. The more complex the training material, the higher the cross-entropy value for that material and a given model or network.

The real probability distribution of the language that forms musical structures is not known, however it can be approximated using a corpus of examples that represent a part of the language. In this work we used cross-entropy as a measure of how successful the different models are in learning structures of training data, by comparing results on that data. An interesting extension to the evaluation would involve the more usual use of cross-entropy results for unseen data.

Smoothing is used to enable the program to introduce events that do not occur in a training data set during the generation of new material. The cross-entropy measurements could be performed without taking the smoothing models into account, because there are no unexpected events in the training data. However, given that we are also interested in generating new material using our program, we want to include smoothing so as to be able to put cross-entropy values on the new data.

Using this type of measurement, the cross-entropy values for models that use smoothing are expected to be higher than the cross-entropy values for equivalent models with no smoothing. The reason for this is that the transition and emission probabilities are distributed more widely across the entire model and the probabilities are lower overall because a given percentage of the probability space is reserved for the smoothing process. These lower probabilities lead to a higher value of cross-entropy for each smoothing model. The cross-entropy values for smoothing models are shown (e.g. in Table 8.15) to illustrate this point.

Tables 8.13 and 8.14 list the cross-entropy values calculated using the training data for the non-smoothing models and networks that generated musical examples for the two-part listening experiment. Eighty different chorales (5880 events in total) were used to train the models that generated material sampled at crotchet rate, whereas 50 chorales (6511 events) provided training data sampled at quaver rate. The cross-entropy values are calculated separately for each model, i.e. each step through the network, and are then added together to form a global entropy value for the entire network. This global value allows us to compare different types of networks, provided that they are built using the same types and numbers of models. The networks in Tables 8.13 and 8.14 for instance are all built from three separate models that represent metrical structure, metre/pitch structure and pitch/pitch structure.

In both Tables 8.13 and 8.14 the cross-entropy values are slightly higher for the larger training data set, i.e. the data set that samples the Bach chorales at quaver rate, which is an expected result. The larger the data set the models can learn from,

non-smoothing Bass-Soprano networks		
	80 examples	50 examples
metre	1.093	1.151
metre/bass	5.160	5.732
bass/soprano	5.406	5.816
total	11.659	12.699

Table 8.13: *Cross-entropy values of the training data for a non-smoothing Bass-Soprano network. The left-hand column shows the cross-entropy results for 80 training examples sampled at crotchet rate (5880 events), the right-hand column shows those same values for 50 examples sampled at quaver rate (6511 events).*

non-smoothing Soprano-Bass networks		
	80 examples	50 examples
metre	1.093	1.151
metre/soprano	4.871	5.312
soprano/bass	6.622	7.140
total	12.586	13.603

Table 8.14: *Cross-entropy values of the training data for a non-smoothing Soprano-Bass network.*

the more diverse the distribution of probabilities that reflect this data set.

The cross-entropy results for the different stages of the networks also reflect the levels of complexity of the structures that are represented by the models. According to the results, metrical structure can be represented a lot more accurately by a model than pitch structure for instance. Compared to pitch structure, metre has a very low cross-entropy: metrical structure is regular and predictable, and can easily be represented by a model. The metre/pitch and pitch/pitch models have to represent structures with an increased complexity, thus the cross-entropy of the training data is higher. The cross-entropy values indicate that a network that models

soprano depending on bass is slightly more successful in capturing the structural probabilities of the training material. For instance, the global cross-entropies of the training data as represented by a Soprano-Bass network are consistently higher than those represented by a Bass-Soprano network. The metre/soprano model receives slightly better values than the corresponding model in the Bass-Soprano network, however the entropy values for the final step through the network are considerably higher and result in comparatively worse global cross-entropies. The main explanation for this effect are the different levels of complexity of bass and soprano lines respectively. Bass lines tend to be more diverse than soprano lines, which often prefer a uniform pitch distribution. Thus, for a probabilistic model, a soprano part is simpler to learn through the extraction of probabilistic structures. This is reflected in the cross-entropy values of the metre/soprano and metre/bass models. In the consecutive step through the network however it becomes apparent that the simpler structure of a soprano part is not beneficial when acting as the input to a model. The lack of diversity of events in a soprano part reduces the probability of each bass event, thus the cross-entropy values for the soprano/bass models are higher than the values for the bass/soprano models.

Tables 8.15 and 8.16 show the cross-entropy values for the same training material, represented by smoothing networks. The metrical structure was modelled without smoothing, therefore the cross-entropy remains the same as previously. The values are again slightly increased compared to the non-smoothing networks. This can easily be explained: because smoothing models take a part of the probability space and assign it to events that have not occurred in the training material, the probabilities of every event that *does* occur in the training data is slightly lower than if there were no smoothing process.

Comparing the cross-entropy values for the networks that use hierarchical models with those that use standard flat HMMs (see Table 8.17), the results support the assumption that the hierarchical structure of the models facilitates and enhances the accuracy of the learning process. For the same training material, the cross-entropy values are notably higher. With only one level of description that represents all the phrases in a piece, the transitions between the states are assigned lower probabil-

smoothing Bass-Soprano networks		
	80 examples	50 examples
metre	1.093	1.151
metre/bass	6.841	6.992
bass/soprano	7.012	7.327
total	13.946	14.470

Table 8.15: *Cross-entropy values of the training data for a smoothing Bass-Soprano network. (The metrical structure was learned using a non-smoothing model, thus the values are identical to those for non-smoothing networks.)*

smoothing Soprano-Bass networks		
	80 examples	50 examples
metre	1.093	1.151
metre/soprano	6.767	6.832
soprano/bass	7.311	7.614
total	15.171	15.597

Table 8.16: *Cross-entropy values of the training data for a smoothing Soprano-Bass network.*

non-smoothing Bass-Soprano networks with flat HMMs		
	80 examples	50 examples
metre	1.597	1.611
metre/bass	7.124	7.398
bass/soprano	7.540	7.781
total	16.261	16.790

Table 8.17: *Cross-entropy values of the training data for a non-smoothing Bass-Soprano network of flat HMMs.*

ities because they are independent of their timing. For example, a transition `G - Bb` that only occurs in the first phrase of a piece is assigned zero probability (or a very small probability if the model uses smoothing) by a hierarchical model in every phrase except the first one; a flat model however will allow the transition in every phrase and thus reduces transition probabilities overall.

Given the global cross-entropy values, the non-smoothing Bass-Soprano network should be the most accurate in modelling, and thus predicting, the training material. Assuming that successful modelling of the training data will lead to the successful generation of believable results, the network should have generated those results that scored the best results in the two-part listening experiment. Indeed, the subjects evaluate the two pieces, 2P2 and 2P3 (see Figures 8.1 and 8.2), that were generated by this type of network as being of high quality, especially example 2P2 in Figure 8.2. This example has a cross-entropy of 15.138, which is considerably higher than the cross-entropy of the training data at 12.699. However, the goal is to model the training material as closely as possible and to reuse the probabilities extracted during training during the generating process. The cross-entropy of the generated material thus shows how far the new material is removed from the training data, it does not directly reflect the quality of the training process. Newly generated examples whose cross-entropy is close to the cross-entropy of the training data however are likely to be of better quality than those examples with much higher cross-entropy values. This is especially true for examples generated using

smoothing models: if a new piece has a cross-entropy that is much higher than the cross-entropy of the training data, it is likely that a substantial part of the new example was built from events that did not occur in the training material.

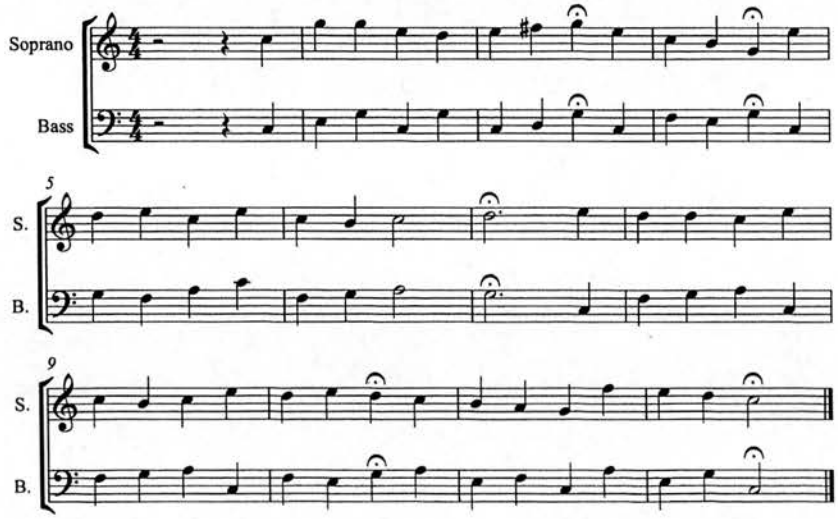


Figure 8.1: *Non-smoothing Bass-Soprano network, sampled at crotchet rate. Cross-entropy values per model: metre = 1.075, metre/bass = 5.139, bass/soprano = 4.920. Global cross-entropy value: 11.134.*

The example in Figure 8.2 received a higher global cross-entropy value because it includes a small number of lower probability events. The local cross-entropy for the metre/bass structure is especially high compared to the training material, which suggests that most of the lower probability events occur in the bass line. The fact that this example was the subjects' preferred piece (aside from the original Bach chorales) in the listening experiment supports that salient, low probability events can increase the quality of a piece of music. Too many rare and low probability events in the music however may have the opposite effect and break the flow of the musical structures. This can be problematic for examples generated by smoothing networks which give too much probability space to previously unseen events.

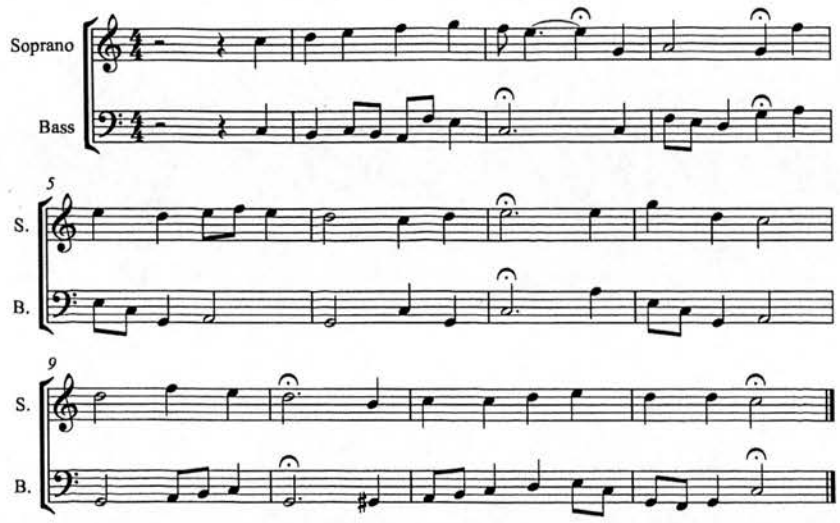


Figure 8.2: *Non-smoothing Bass-Soprano network, sampled at quaver rate. Cross-entropy values per model: metre = 1.075, metre/bass = 7.738, bass/soprano = 6.325. Global cross-entropy value: 15.138.*

The cross-entropies of the four-part training material are similar in value: Tables 8.18 and 8.19 show the degrees of accuracy with which the two different types of networks represent each training corpus. The global entropies are very similar for both networks, with a slight preference given to the network that represents alto and tenor being dependent on the soprano.

8.3 Analysis of Generated Music

The cross-entropy measurement puts a value on a composite network's ability to capture and represent the data in a training corpus. However, it can only give a limited insight into the quality and correctness of the musical examples that are generated using these networks. This section gives a musical analysis of some of the four-part examples used in the listening experiment, focussing mainly on the structural aspects of the music.

non-smoothing B-S-A-T networks		
	80 examples	50 examples
metre	1.093	1.151
metre/bass	5.160	5.732
bass/soprano	5.406	5.816
soprano/alto	5.374	5.832
alto/tenor	5.243	5.768
total	22.276	24.299

Table 8.18: *Cross-entropy values of the training data for a non-smoothing B-S-A-T network.*

non-smoothing B-S-A-S-T networks		
	80 examples	50 examples
metre	1.093	1.151
metre/bass	5.160	5.732
bass/soprano	5.406	5.816
soprano/alto	5.374	5.832
soprano/tenor	5.012	5.615
total	22.045	24.146

Table 8.19: *Cross-entropy values of the training data for a non-smoothing B-S-A-S-T network.*

Ideally, each piece of music that is generated based on probabilities extracted from a set of Bach chorales should have a style-specific structure. A main characteristic, which is addressed in two questions in the listening experiment, is that each phrase ends in a cadence, with the final phrase ending in a perfect cadence. The listening experiment evidently shows that the lack of cadences in the musical examples results in a very poor quality assessment by the subjects. Example 4P3 (see Appendix B, Figure B.3), which received the best results, does indeed have clearly defined cadences. The piece consists of five phrases, all of which end on either the dominant or the tonic. The perfect cadence at the end of the piece is a $V - II^6 - V^7 - I$ progression. The phrases of example 4P6 (see Appendix B, Figure B.6), which is also judged to be of acceptable, if slightly poorer, quality in the listening experiment, progress towards cadences (either dominant or tonic). The perfect cadence at the end of the piece is less clear ($I^6 - II_5^6 - I$), if still present. As opposed to these two pieces example 4P7, generated by a network of standard flat models (see Appendix B, Figure B.7), does not show any progression towards cadences at the end of its phrases, or indeed the end of the piece. The lack of a flat model's ability to learn the relative time of an event in a piece, which is supported by the hierarchical models, becomes evident. Similarly, the example taken from (Allan, 2002) (example 4P8, see Figure B.8), which was also generated using standard HMMs, does not show any progression of the harmonic structure towards cadences at the end of the phrases. The piece does find a perfect cadence at the end with a $I - V^7 - I$ progression, however the example is a harmonisation of the original melody which thus strongly leads the models towards the perfect cadence.

Given that these four examples represent various levels of quality as assessed by the subjects of the listening experiment, it is worth having a closer look at all the examples individually to highlight and understand the possible reasons behind those assessments. Example 4P3 was mistaken to be an original Bach chorale by 42.9% of the subjects, and received above average results for all questions except one, which targets the style of the piece. Aside from the presence of cadences, which was discussed earlier and has an obvious influence on the subjects' judgements, the overall structuring of the phrases and the reuse of certain patterns inside the phrases are important to the perception of the entire piece. The use of small patterns and the

repetition of these patterns is evident in example 4P3. On the metrical/rhythmical level, there seems to be a preference for the quaver-quaver-crotchet pattern, which occurs throughout the piece and in all four voices. The use of this pattern becomes especially clear for instance in the second phrase, where it is alternated between alto, tenor and bass. This pattern is used similarly in the final phrase, adding the notion of contrary movement between the voices. Another example of repeated structures is the rhythms at the end of phrases three and four, which are almost identical to each other. This reuse of small patterns in a piece of music is a clear indication of parallelism, i.e. the occurrence of identical or similar fragments of events. The phenomenon of parallelism will be discussed further in Chapter 9. Additionally, the piece seems to have an acceptable large-scale structure. For instance, the second-last phrase progresses towards a cadence on the dominant, then the final phrase modulates briefly into A minor before returning to C major and ending in a perfect cadence. The sum of these points can probably explain why the subjects rated the example to be of an above average quality during the listening experiment. However, the piece is far from being mistake-free: the occurrence of parallel fifths or octaves is not uncommon, especially between voices that are not directly dependent upon each other in the network. Thus for example the alto and bass parts are entirely parallel in bar nine, a clear mistake according to the rules associated with Bach chorales.

Example 4P6 was judged to be of similar quality to the previous example, however it was seen to be of a style that is closer to a Bach chorale than example 4P3. It is difficult to ascertain why example 4P6 received a much higher rating with regard to its style. The piece makes use of cadences and parallelism to a similar degree as the previous example. The main difference between the two pieces probably lies in the different number and lengths of their phrases: example 4P3 has five phrases, the second phrase being very short at four beats; example 4P6 on the other hand consists of four phrases which are all eight beats or more. The single very short phrase in example 4P3 is quite unnatural for a chorale, which might have influenced the subjects' decisions. The voice leading of both pieces is in a style that can be expected from a chorale: the soprano in both examples is more uniform and prefers stepwise motion, leaps of an interval bigger than a third are rare; the bass

however is more varied and leaps of fourths and fifths are not uncommon.

Whereas the previous two pieces were well-received during the listening experiment, examples 4P7 and 4P8 were assessed to be of rather poor quality. Both pieces were generated using standard flat HMMs, with example 4P8 being a harmonisation of Bach chorale K211, BWV 367, taken from (Allan, 2002). The main weaknesses of both pieces are their lack of cadences, as discussed earlier. Example 4P7 however also suffers from poor phrase structuring, particularly on the pitch level. As the flat models cannot distinguish the relative time of an event, the phrases of each voice are generated with a number of small mistakes, especially towards the end of a phrase where the cadence should be prepared. The mistakes are carried through to the next step in the network and accumulate (see for instance phrases three and four of example 4P7). The reuse of patterns is still detectable, however the lack of structure in the phrases and the entire piece outweigh any traces of parallelism. Finally, although example 4P8 is the harmonisation of an original Bach chorale melody, and thus has the advantage of an existing high quality structure to build the remaining parts on, it is of poor quality. The piece ends on a perfect cadence, however the endings of the phrases are not respected at all and there is virtually no progression towards a cadence at any point through the piece.

8.4 Summary

This chapter tries to deliver an evaluation of a number of musical pieces that were generated using different types of composite networks. Firstly, the results of a listening experiment for both the two-part and four-part pieces are presented and interpreted. Then, as a more objective measure of the networks' and models' ability to capture the structural probabilities of the Bach chorales, an evaluation using cross-entropy values is provided. Finally, a small number of the four-part pieces that were presented to the subjects of the listening experiment are analysed in greater detail, trying to determine the reasons behind their results in the experiment. The following chapter will discuss the overall results further, trying to highlight the importance of parallelism and network structure.

Chapter 9

Discussion

The previous chapters have defined the novel concept of hIOHMMs and their inference algorithms, introduced the idea of composite networks and evaluated the musical results generated with these networks. This section will discuss, with regard to the generated results, the methods and techniques, and will try to single out their advantages and drawbacks.

9.1 Methodology

In order to learn and generate believable large-scale musical structures, the complex problem space which these musical structures represent has to be divided into smaller and simpler subtasks. A natural method of dividing an entire piece of music is the separation of the different layers, or levels of representation, that make up the entire musical structure. A simple melody can be regarded for instance as being a composite structure built with metre, rhythm and pitch structures. The entire melody is thus a composite object that is defined by the combination of those three structures, i.e. their interdependency, as well as their individual internal structures, i.e. the dependencies between their internal events. This research uses the notion of building entire pieces of music from separate structural levels and introduces the concept of composite probabilistic networks that use hierarchical extensions of HMMs. The models that build the networks extract and learn the relevant internal structural probabilities and interdependencies from a given training corpus. The composite nature of the networks allows for great flexibility regarding the config-

uration of the networks and makes them dynamic and expandable. HHMMs and hIOHMMs act as the building blocks of the networks.

The flexibility in the configuration of the networks proved to be a major advantage in this research. Each model inside a network can be trained individually and independent of any other models in the network. Thus a trained model can also be reused in different network structures. The obvious restriction to reusing trained models lies with the training data: in order to receive consistent results, a workable network should only use models that were trained on the same set of training data. Even with the improved training algorithms, the procedure of estimating the new model parameters and probabilities to reflect the training material can be computationally expensive. Therefore the opportunity to use already trained models and plug them into new networks greatly reduces the computing cost. The reusability of trained models and the easy reconfiguration of the network made it possible to test a large number of different network types (see Chapters 7 and 8) in order to find the optimal layout of models for a given problem. If a network has to represent music with multiple voices, for instance two-part and four-part music, it is important to decide the hierarchy of the interdependencies between the different parts so as to be able to construct the most efficient configuration of models. In two-part music for example, it is necessary to decide whether to represent the soprano dependent on the bass or vice versa. A simple solution is to build both types of networks, i.e. one network that models the soprano dependent on the bass and one that models the bass followed by the soprano, and to then measure the accuracy of the networks using cross-entropy. This is only possible if it is cheap to build and run the different network structures by reusing the trained models.

The networks' building blocks consist mainly of HHMMs, which provide a basis layer, and hIOHMMs. The input-output models learn the connections between a given input layer and an observation sequence, and thus stand for the interconnection of the different levels of representation of musical structures. For instance, a hIOHMM that models metre and bass extracts the structural probabilities of a bass line when depending on metre. This interconnection then allows for the direct link between a model that emits a metrical structure and the model that uses

metrical structure as an input. During the generation process, a new structure is created and emitted from a model and then passed on to the following model in the network to serve as input. When this method of feeding structures forward through the network is used with models that do not smoothen their parameters after training, it is possible to overfit the generated structures. The effects and consequences of overfitting are discussed later in this chapter (see Section 9.3.1).

9.2 Learning Musical Structures

Cross-entropy measures the degree of accuracy with which a network can model and reflect training data, and thus is an indication of the quality of the learning process. Chapter 8 shows that not all types of networks are equally powerful. The cross-entropies, which measure the networks' abilities to represent a data set, vary slightly for different network structures. The cross-entropies for the networks that use hierarchical models however demonstrate clearly that these models are more suitable for learning and representing musical structures than the simpler flat HMMs (see Tables 8.13 and 8.17). The hierarchical models' superior power lies in their ability to naturally learn the hierarchical structures in music and the relative timing of the musical events, in addition to the events' frequencies of occurrence and most likely contexts.

One aim of this research was to devise a system that can extract the knowledge that is necessary to create new pieces of music from a training corpus of Bach chorales, while only a minimum of expert knowledge is hard-coded into the system. The main areas where domain specific knowledge would have influenced the decisions made about the system are the representation of the music inside the system, the design of the basic models and their structures as well as the configurations of the networks. Regarding the music representation, an effort was made to keep the representation simple and minimalist. It was thus decided that it is not necessary to explicitly define rhythmical or harmonic structure, as both these structures can be implied from pitch structures and metrical structures. The separate symbols that represent the start and end of a phrase, as well as the end of a piece, can be seen as giving more than the minimum information, i.e. pitch or rhythm, to the sys-

tem, however this information was seen to be essential: if a system is supposed to learn the structure of a phrase it needs to have enough knowledge to understand the notion of 'phrase'. Using that same reasoning, in order for the system to learn the large-scale dependencies of an entire piece, it needs to know the concept of 'piece'. Therefore the representation includes a symbol that acts as a delimiter between the different chorales, defining the length of the pieces.

Designing the probabilistic models, i.e. deciding the configurations and structures, requires the use of a certain level of domain specific knowledge. The knowledge that influences the design stage is thus also incorporated in the final model. When deciding on a structure for a HHMM it is possible to set up a model with an arbitrary number of levels and states, however it is unlikely that such a model will ever work to the full potential of a properly structured HHMM. In this research, the main information that is hard-coded into the models via their structures, i.e. their numbers of levels and the distribution of states on those levels, is the concept of phrasing. The models are defined such that the entire piece is subdivided hierarchically into a varying number of distinct phrases, which in return are built from small events such as pitch or metre symbols. Additional information is given by defining that a piece has to consist of at least two phrases, that the start and end phrases are distinct, i.e. the system knows that each piece must have exactly one start and one end phrase, and that a piece has to end with an 'end' symbol. When devising the structures for the networks, the decision to use metrical structure as a basis was influenced by the simplicity of the structure rather than being based on any expert knowledge.

The amount of explicit expert knowledge that is hard-coded into this system is very small, yet it is crucial to the performance of the system. The individual models need to be provided with basic domain knowledge of the data they will represent, otherwise their performance is likely to be suboptimal. As the training procedure involves updating a model's parameters, i.e. its transition and emission probabilities, to maximise the overall probability of the training data, a poor choice of model structure can result in model parameters that only represent a local maximum. A well-designed model however can find the global maximum, i.e. an ideal set of

parameters, for the training data.

The major part of the knowledge that is used to generate new large-scale musical structures is implicit knowledge that is extracted from the training data and stored as a model's transition and emission probabilities. Each model thus builds its own set of rules according to the probabilities with which a process can move from one state to the next and emit certain observations. Implicit knowledge holds a major advantage to hard-coded, explicit knowledge: when the system needs to learn a new set of rules, it can extract new probabilities from an alternative training data set, for example music of a style different to Bach, and then update its parameters. Unless the basic structure of the training data is entirely different, for instance if the music in the new corpus does not use phrases, the training algorithm updates the system automatically to represent the new data. It is not necessary to change the basic design of the system. A system that relies on hard-coded knowledge on the other hand is likely to have to be rewritten and expanded in order to fit new requirements.

9.3 Generation of Musical Structures

The generative process used with this system uses the random-walk method which has the advantage of producing a large number of results at very low computational cost. The random element that is part of the procedure influences the quality levels of the generated results, which can thus vary greatly. However, this is not seen as a drawback of the process: rather than wanting to generate only the best examples that a system can produce, it is more desirable to produce and then analyse a larger number of results of varying quality. These results can then be evaluated with methods similar to the ones described in Chapter 8. Cross-entropy put an absolute value to a network's ability of representing a data set. A network that can accurately learn musical structures is also likely to generate new results that are of acceptable quality. However cross-entropy value alone does not necessarily indicate the perceived quality of the new music. The cross-entropy measures the amount of information contained in an event sequence and puts a value onto the predictability of that sequence. If a new piece of music has a cross-entropy that

is close to the cross-entropy of the training material, this only indicates that the new musical structure is close to the training corpus regarding its level of information or predictability. It does not however put an explicit value to the quality of the new piece of music. This is underlined by the fact that the two-part example 2P3 (see Figure A.3), which was preferred by the subjects during the listening experiment has a cross-entropy of 15.138, compared to an average of 12.699 for the training corpus. This leads to the conclusion that the most probable generated musical structure is not necessarily the listener's preferred musical structure. A sequence of musical events that has an overall high probability, and therefore low cross-entropy and information content, is unlikely to contain many salient, low-probability events. Thus a listener will perceive such a sequence as a rather uniform structure with few outstanding events. The presence of a small number of low-probability events can benefit the perceived quality of a musical structure, as is evident through example 2P3.

9.3.1 Overfitting and Smoothing

The random element in the generation process gives the system a certain degree of freedom when making choices. Instead of using the overall most likely paths through models, which restricts the system into generating the overall most probable solutions, the random element allows to potentially choose paths and events that have low probabilities. All the possible results are made available and events with high probabilities are preferred, however the system does not force the generating procedure to choose events according to their likelihood.

Although the generation is not dictated by an overall probability threshold, it can be restricted through the use of the non-smoothing models in the networks. The input-output models in the networks use the observation sequence emitted from the previous model as their input sequences. When using non-smoothing models, it is important that these input sequences are legal sequences, i.e. they have to be representable in the current hIOHMM. All the progressions that are in an input sequence have to be legal progressions with non-zero probabilities. If an input sequence does include progressions between events that have zero-probabilities in the current hIOHMM, the input sequence will be rejected. For example, given a model

that emits a sequence which contains the pitch progression `C#-E`. The sequence is then used as input to the next model in the network. In that model however, the input progression `C#-E` has zero probability because it was not encountered in the training data. The model will therefore reject the input sequence as illegal. The effect of the distinction between legal and illegal input sequences is *overfitting*.

The main problem of overfitting is the fact that a system prefers to regenerate material that is close to the training data, i.e. the system will have the tendency to simply recreate the existing training data. This problem can be resolved by using smoothing procedures after the re-estimation of each models parameters to redistribute probabilities and assign part of the probability space to unseen events. However, because this system uses a generative approach, with an incorporated random element, which builds the final results from several layers of musical structures, the potential of severe overfitting, i.e. recreating the training data, is seen to be very low and has currently never been traced during a generation process. Rather than decreasing the diversity and quality of the generated musical examples, slight overfitting can be seen to be beneficial: the random-walk method can create a large number of structures, whose quality can vary and is not always easily assessed. By using non-smoothing models the input sequences that are of very poor quality, such as sequences that have a very high number of low probability events, are rejected by the system, whereas the sequences that are higher in quality are accepted. Slight overfitting can thus lead to a beneficial selectiveness of the system, as long as the selectiveness does not become too restrictive.

Models that use smoothing procedures will not show signs of overfitting, though they might suffer from the opposite effect and become too flexible. Even if non-observed events are only assigned a minor fraction of the probability space, the effects of using too many events from that space can have considerable consequences to the quality of the generated material.

9.3.2 Parallelism

A side-effect of the random-walk generative procedure, which makes solutions available rather than enforcing them, is the occurrence of parallel structures in

the generated material (see Section 8.3). The problem of parallelism, or the identity/similarity problem, has had plenty of attention in a number of research projects (Lerdahl & Jackendoff, 1983; Cambouropoulos, 1998) and focusses on identifying and defining patterns in musical structures that show traces of similarity. The main difficulty with parallelism is that, although it is simple to define the identity of two objects or patterns, describing their similarity is a lot more complicated and dependent on their definitions. A popular solution to the problem is to attach weights to each of the parameters of the objects, and to define a threshold at which two objects are seen to be different. By calculating the sum of the differences in weight of the parameters, the objects are classed as being different if this sum exceeds the threshold. The main drawback of this method is that the weights of the parameters and the threshold are often chosen arbitrarily, and the parameters are different for each pair of objects. The problem of parallelism is a clear example of where a rule-based approach for instance struggles to find a solution. The composite networks system learns parallel structures from the training set and stores the probabilities that define them into the models' parameters. Two patterns are seen to be parallel if they are similar to a certain degree (or identical). Parallelism can appear in any structure of a piece of music, for instance in the rhythm or pitch structures. The different models that make up a composite network, and which break a musical piece down into several layers of structures, learn those parallel patterns through their recurrence inside the training data. The more often a certain pattern occurs in the training data, the higher the probabilities that will be assigned to the transitions, which then lead to the construction of the pattern. The layered configuration of the networks allows for parallelism to spread through all of the musical structures, but does not necessarily constrain towards parallelism, e.g. a bass line that contains parallel patterns may lead to a similarly parallel soprano line, but not necessarily so. In the generation process, parallel patterns will be favoured due to higher probability of occurrence of the parallel events. As shown in the evaluation of the musical examples in Chapter 8, most of the generated musical material shows some traces of parallelism. Although the models and networks do not learn a definition of parallel structures, they are made available and are favoured in the solution space.

9.4 Summary

This chapter discusses the methods and techniques that the system of composite networks is built on, and attempts to explain arising issues such as overfitting and parallelism. It also highlights the difference between the implicit and explicit knowledge that was used when designing the representations, models and networks. The next and final chapter will draw the conclusions of this research and establish whether the initial goals have been met.

Chapter 10

Conclusion

This research set out to devise a system that, with the use of a minimum of explicit expert knowledge, can learn and generate believable large-scale musical structures. Based on the experiences from related research works, the complex problem space of musical structures was divided into simpler, more manageable domains. The notion of composite networks, whose individual building blocks represent the different domains of the problem space, was introduced. It was decided that probabilistic networks would be ideal building blocks for the composite networks, because they are proven to have the ability to extract important structural probabilities from large data sets. The concept of HHMMs was extended using the idea of input-output models, and the novel technique of hIOHMMs was defined. Both HHMMs and hIOHMMs were then used in conjunction to create composite networks that can represent the different layers of musical structures.

At the beginning of this thesis, the project scope and its major goals were outlined. The research aimed to develop a system that should be able to create new pieces of music, which have a clear large-scale structure, from scratch, while making use of as little expert knowledge as possible and learning the necessary structural probabilities and information from a corpus of training examples. These aims address some major issues that were highlighted from related research projects (see Chapter 2) and on which this research attempted to improve. Moving away from rule-bases and knowledge intensive systems to an approach that largely relies on the automatic extraction of knowledge from training material, the main challenge

was in devising up a system that could create new music entirely from scratch. Only one of the related works in Chapter 2[1] addresses this problem, solving it using an expert system, thus restricting the generation of music heavily to a specific style and type of music. The second challenge was the representation, and generation, of large-scale musical structure. The projects presented in Chapter 2 are all successful in modelling and representing small-scale and local structures, however there are virtually no attempts to try and capture the large-scale structures in music. The hierarchical structures of the models used for the composite networks did provide a powerful way of representing those large-scale structures.

All of the aims outlined for the project have been achieved during the course of this project: composite networks, and the models that build them, were shown to be capable of successfully learning the local dependencies of the events inside each musical structure, as well as the interdependencies between the different structures. Additionally to extracting and learning the structural information, the approach achieved the generation of new pieces of music that have a believable large-scale structure and are of acceptable quality, as was shown by the listening experiment. Finally, the project scope put a restriction on the amount of domain specific knowledge that could be encoded into the system. The approach uses an essential minimum of expert knowledge to enable the configuration of a workable system.

The newly introduced concepts, i.e. the hIOHHMs and the composite networks, as well as the notions of hierarchy and structural layers in music, have proven to be fundamental to the success of the project. The hierarchical models were crucial in the learning process, as they provide the possibility to extract the probabilities of recurring events relative to their time in a sequence. They made it possible to, for instance, learn the specific makeup of different phrases in a piece, i.e. they could extract specific structural information depending on the timing of a phrase. The timing specific knowledge was then used in the generation process, thus a model would be able to generate large-scale structures that progress towards cadences. The novel concept of hIOHMMs was developed to provide the opportunity to use the power of hierarchical models together with the added constraint of an input

[1] (Löthe, 1999)

sequence. Combining both HHMMs and hIOHMMs in composite networks thus delivered a very powerful tool to represent and learn each of the different layers of musical structures and their interdependencies. The major advantage of using the composite networks was their flexibility and the reusability of the building blocks, i.e. the trained models. The advantage of being able to reuse trained models is largely caused by the approach's most severe drawback: executing the training algorithm on large sets of training data is computationally expensive. A system that can reuse previously trained models can thus avoid a computation overhead and change the configurations of its networks more easily.

As a conclusion it can be stated that systems of probabilistic models combined in flexible, composite networks are highly adaptable and powerful tools that can learn the structural information of highly complex data sets. The training of the separate models can be time consuming and computationally expensive, however this can be avoided by the intelligent reuse of trained models. The system that was devised during this research fulfilled all the goals of the project scope and generated acceptable pieces of music from scratch, extracting the vast majority of its knowledge from a given set of Bach chorales.

Future Work

Using the existing model and network structures, a future step in the research would be to train the models on a distinctly different set of musical material and reassess its performance. Although the system is expected to generate results of a comparable quality if the new material uses mainly short examples of music, it would be interesting to measure its performance on longer pieces of music. In order to obtain similarly good results, it is likely that both model and network structures would need some adjustments to accommodate the characteristics and properties of longer pieces of music.

A major improvement of the system would be an increase of the efficiency of the code that implements the models. Although the revised definitions and algorithms have considerably reduced the runtime of the training process, the computational expense remains high for large training sets. This problem could be resolved by

transforming the existing code into a parallel program, i.e. by rewriting the code so that different tasks and procedures can be distributed over several processors and run in parallel. A successful parallelisation, and therefore increased performance, of the code would allow to train the system on much larger data sets, which could, for instance, either contain a larger number of training examples or use higher precision and sampling rates. Additionally, a parallel system of generative networks would be able to work with models that have an increased number of internal and production states, thus adding to the accuracy which with musical structures can be represented.

Appendix A

Listening Experiment - Two-Part Examples

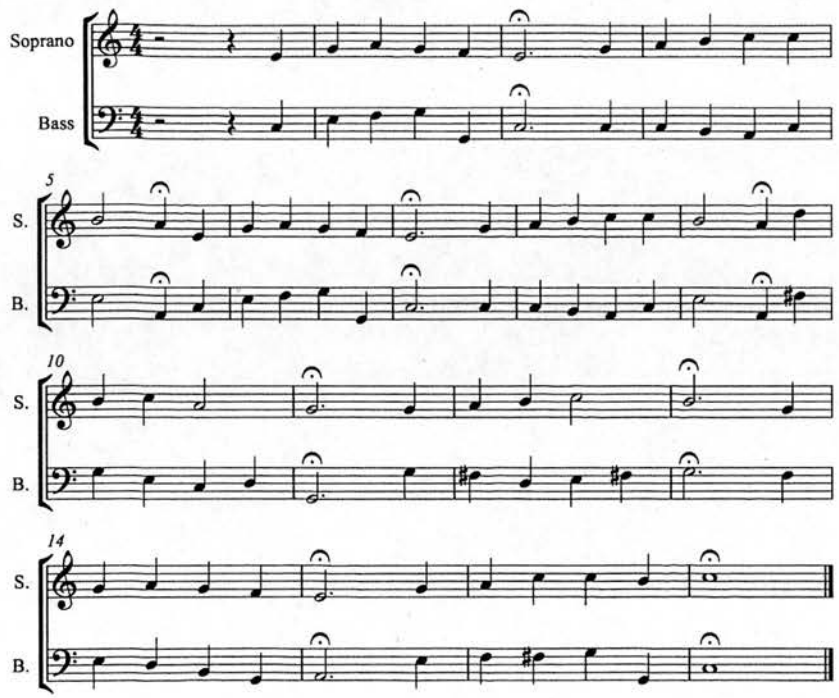


Figure A.1: ***(2P1)*** *Bach chorale K181, BWV 133. Soprano and bass lines sampled at crochet rate.*

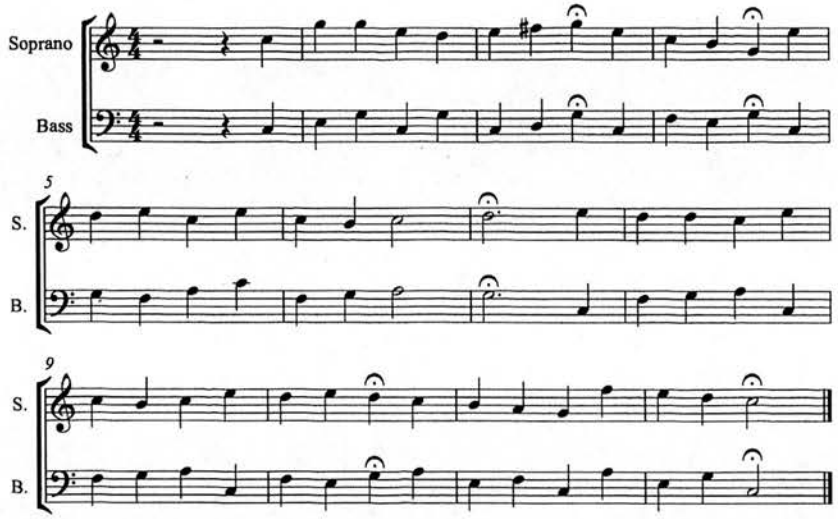


Figure A.2: ***(2P2)*** *Soprano built on top of bass with non-smoothing models (sampled at crotchet rate).*

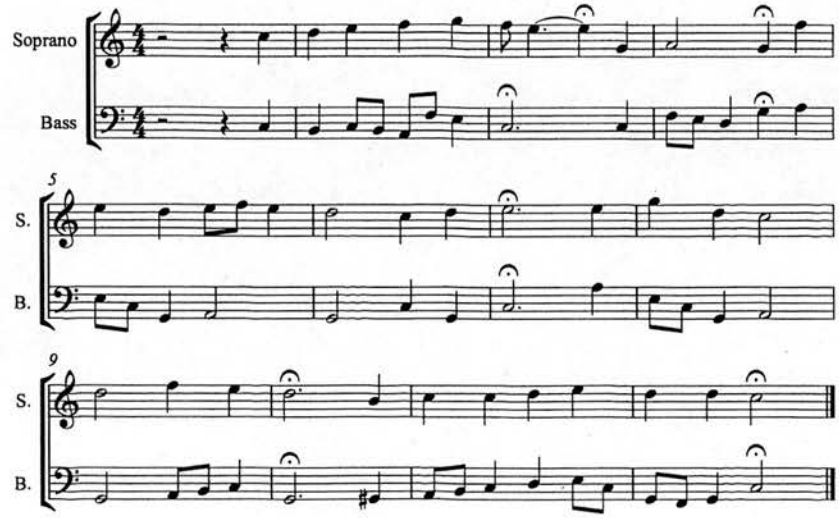


Figure A.3: ***(2P3)*** *Soprano built on top of bass with non-smoothing models (sampled at quaver rate).*

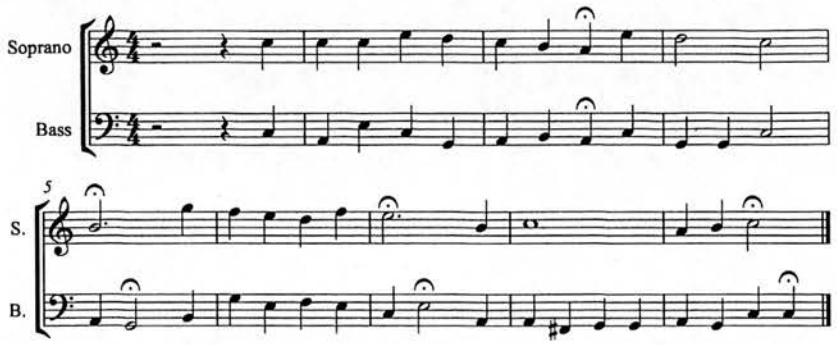


Figure A.4: ***(2P4)*** *Bass built under soprano with smoothing models (sampled at crotchet rate).*

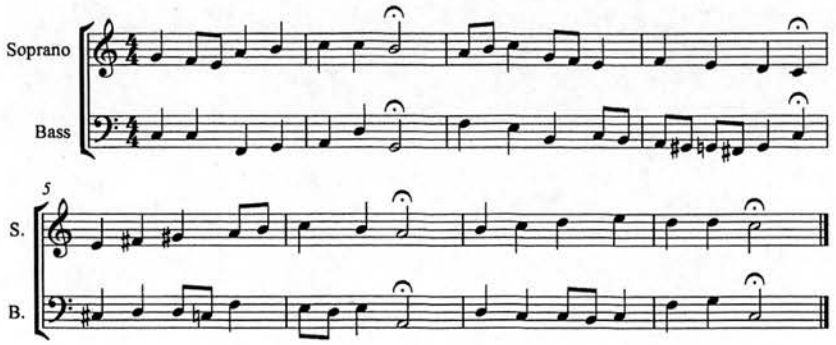


Figure A.5: ***(2P5)*** *Bach chorale K209, BWV 145. Soprano and bass sampled at quaver rate.*

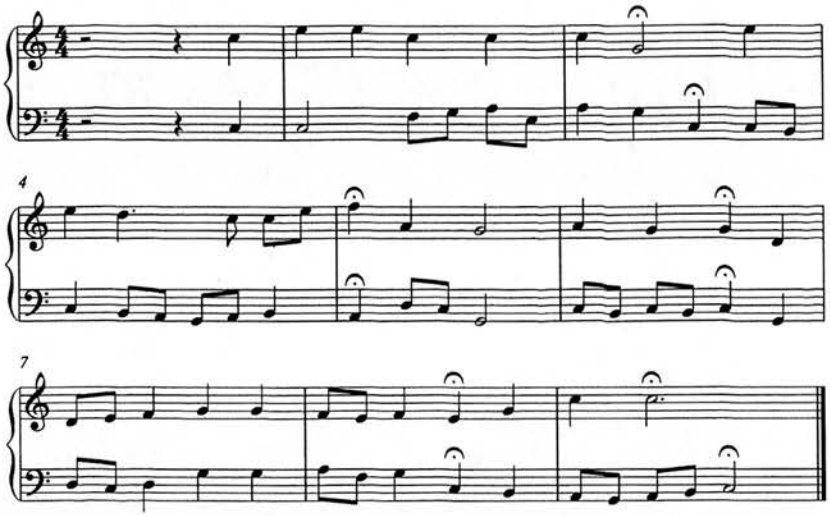


Figure A.6: ***(2P6)*** *Soprano built on top of bass with smoothing models (sampled at quaver rate).*

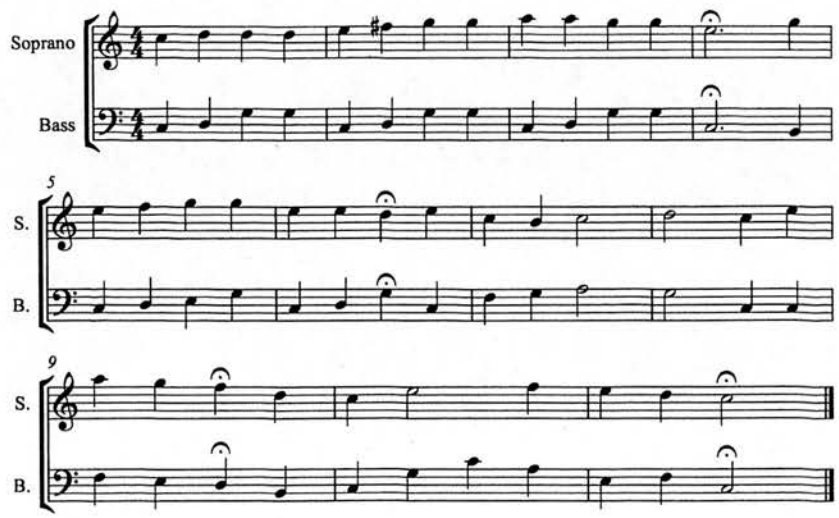


Figure A.7: ***(2P7)*** *Bass built on top of soprano with non-smoothing models (sampled at crotchet rate).*

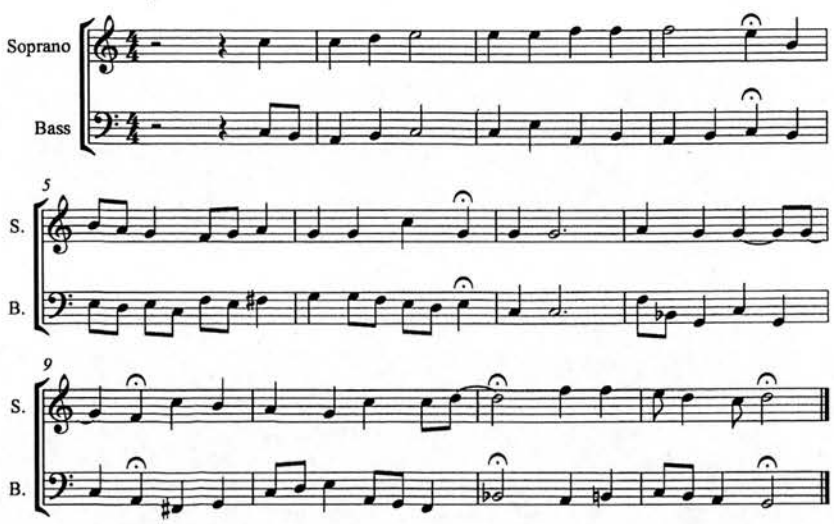


Figure A.8: ***(2P8)*** *Soprano built on top of bass with standard non-smoothing HMMs (sampled at quaver rate).*

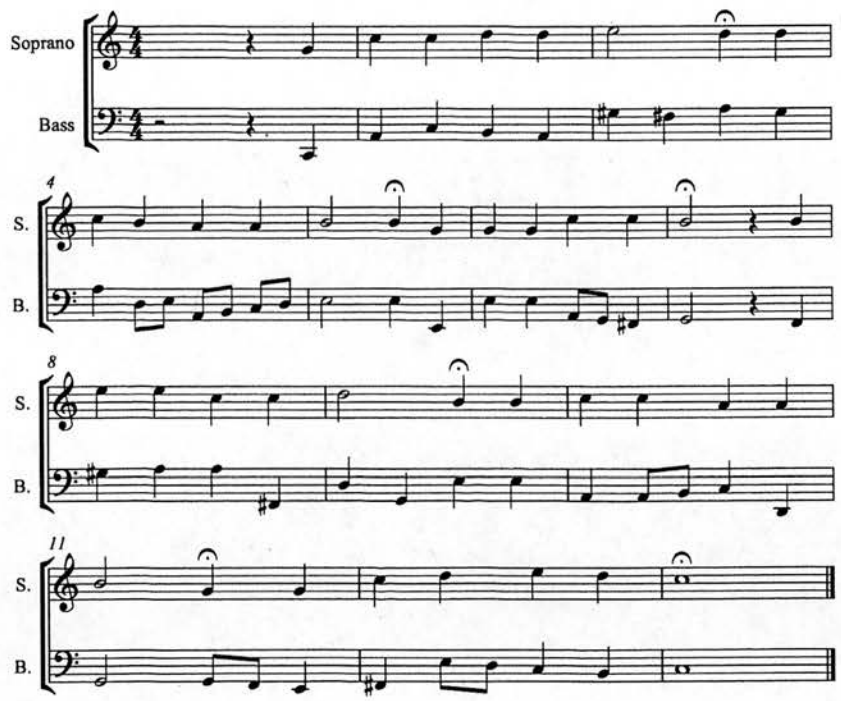


Figure A.9: ***(2P9)*** *Bass line added to the melody of Bach chorale K211, BWV 367, taken from (Allan, 2002).*

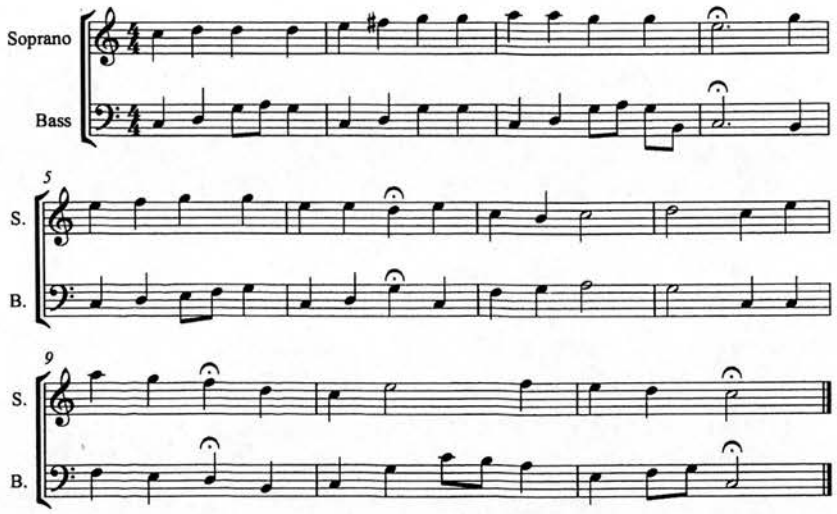


Figure A.10: ***(2P10)*** *Soprano built on top of bass with non-smoothing models. Passing notes added with separate model.*

Appendix B

Listening Experiment - Four-Part Examples

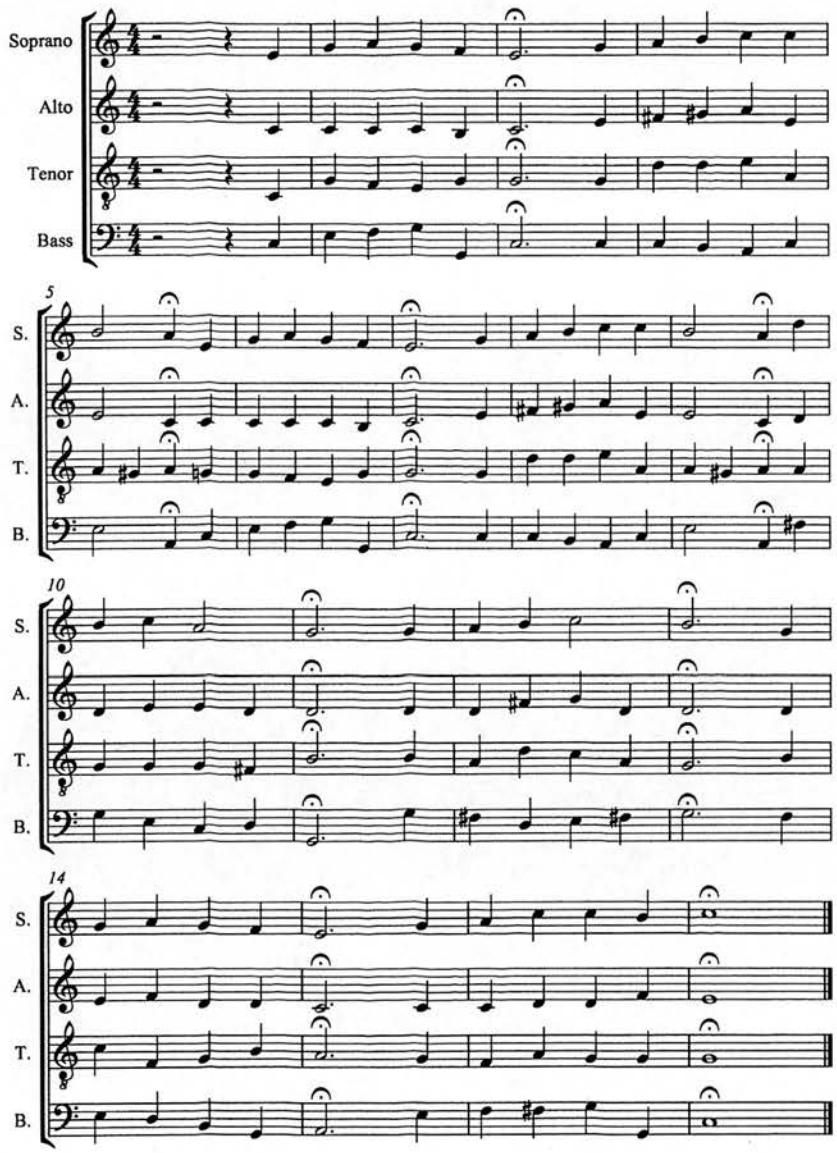


Figure B.1: ***(4P1)*** *Bach chorale K181, BWV 133, sampled at crotchet rate.*

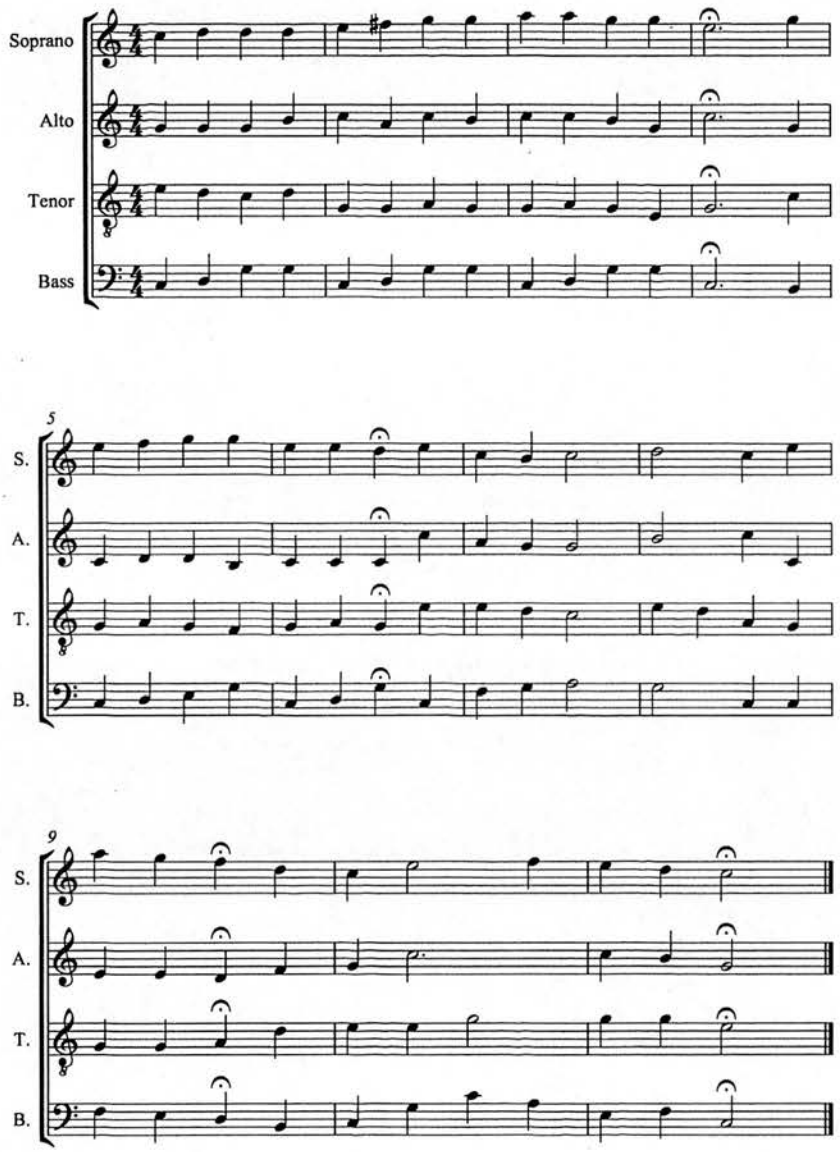


Figure B.2: ***(4P2)*** *Example generated from a B-S-A-T network with non-smoothing models (sampled at crotchet rate).*

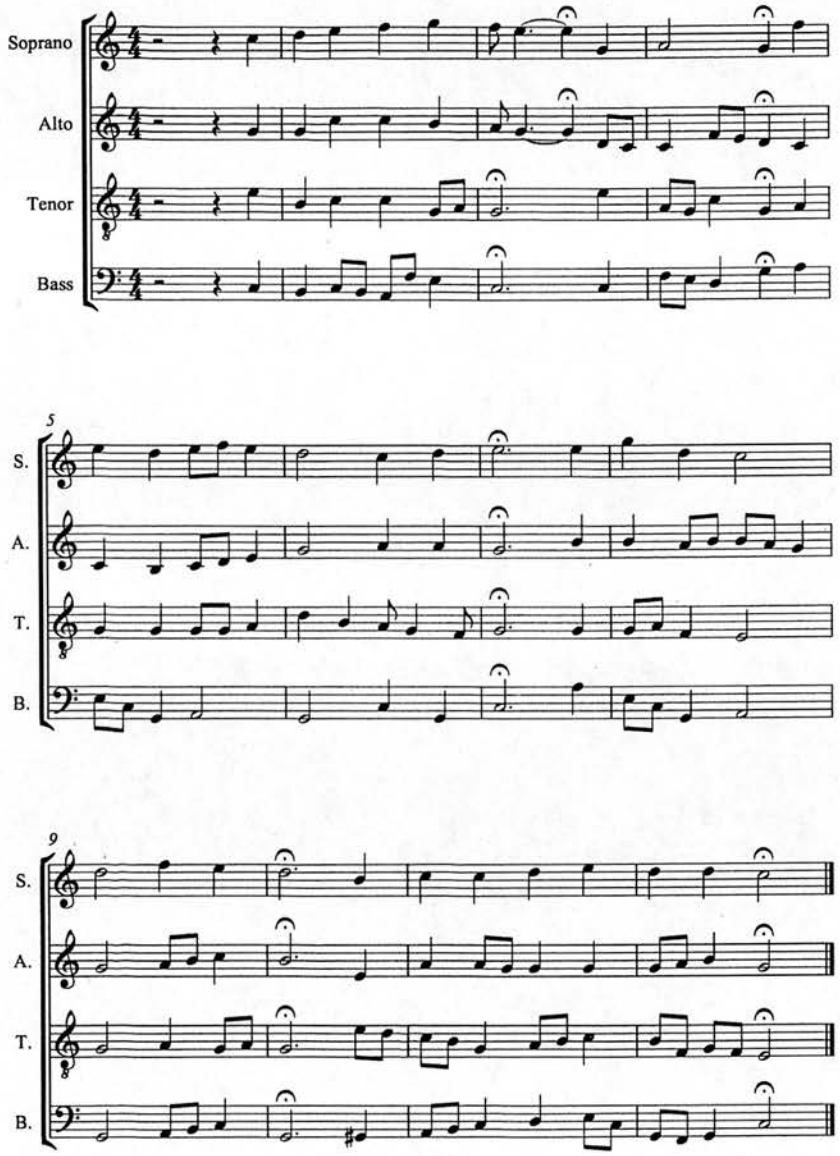


Figure B.3: ***(4P3)*** *Example generated from a B-S-A-S-T network with non-smoothing models (sampled at quaver rate).*

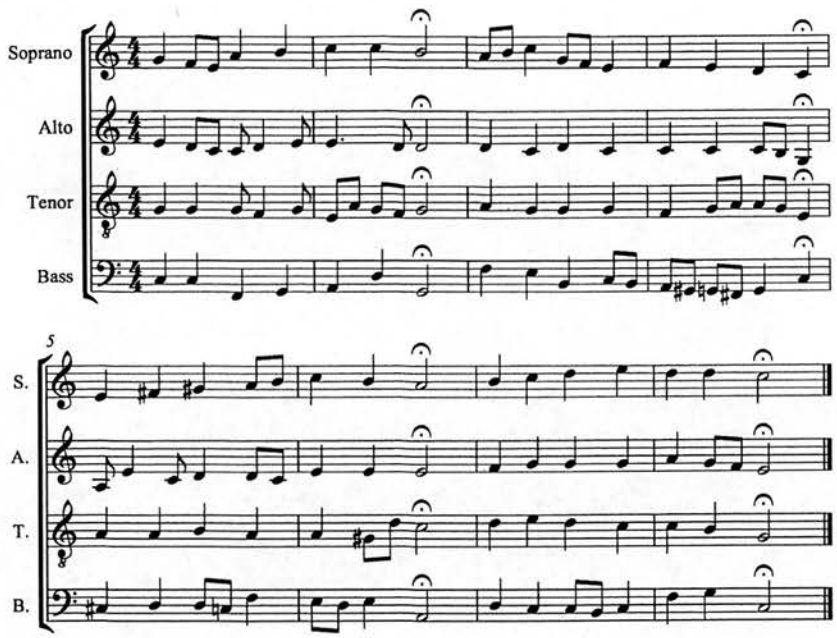


Figure B.4: ***(4P4)*** *Bach chorale K209, BWV 145, sampled at quaver rate.*

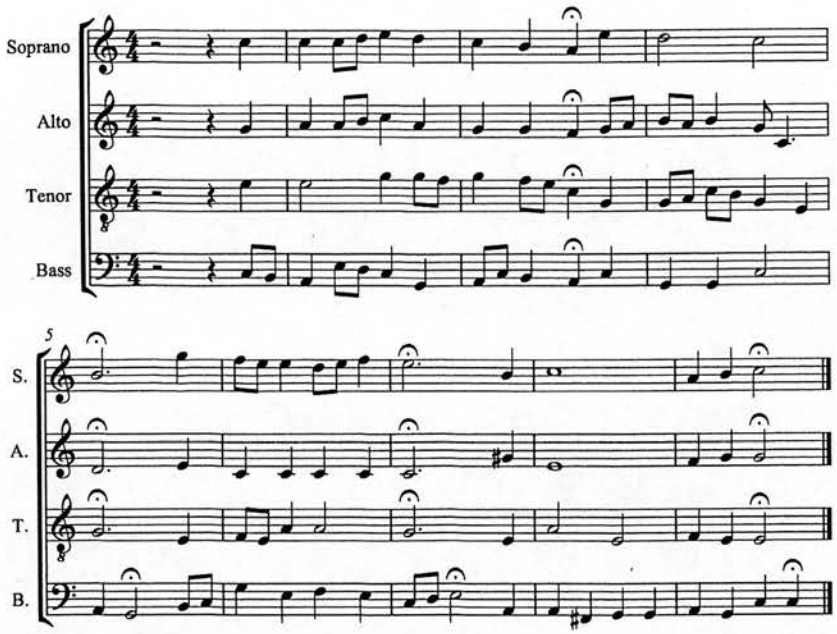


Figure B.5: ***(4P5)*** *Example generated from a B-S-A-S-T network with smoothing models (sampled at quaver rate).*

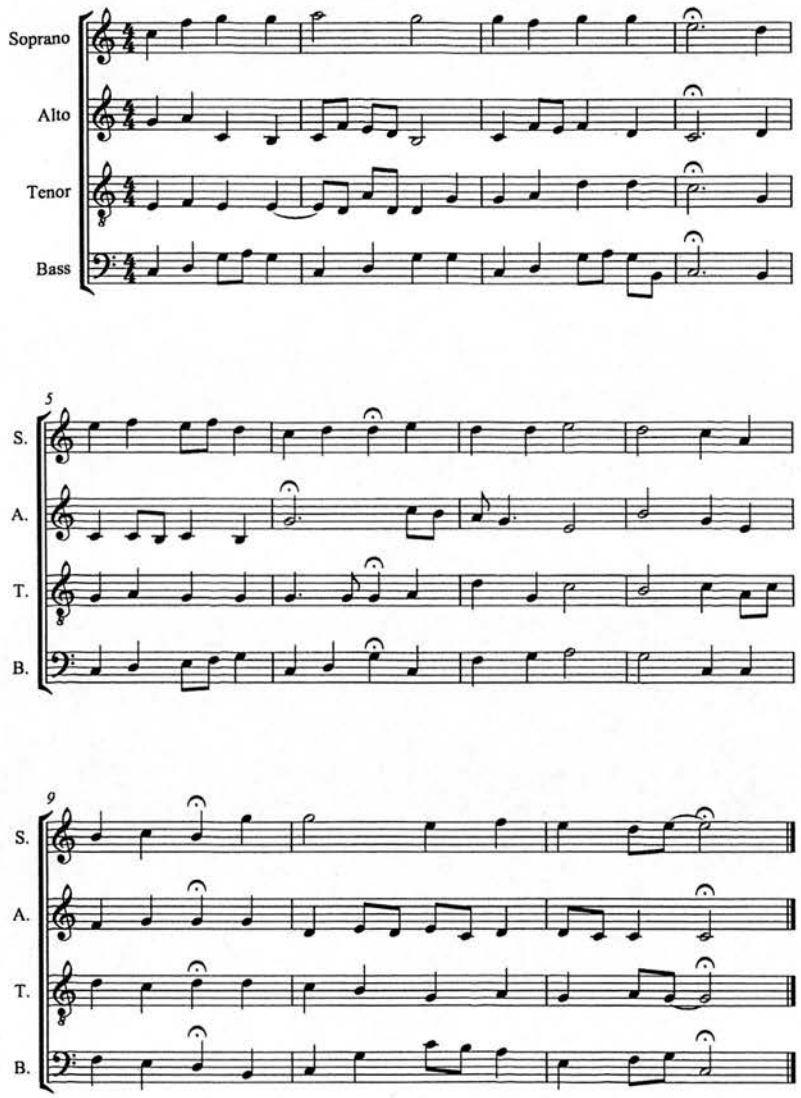


Figure B.6: ***(4P6)*** *Example generated from a B-S-A-S-T network with non-smoothing models (sampled at quaver rate).*

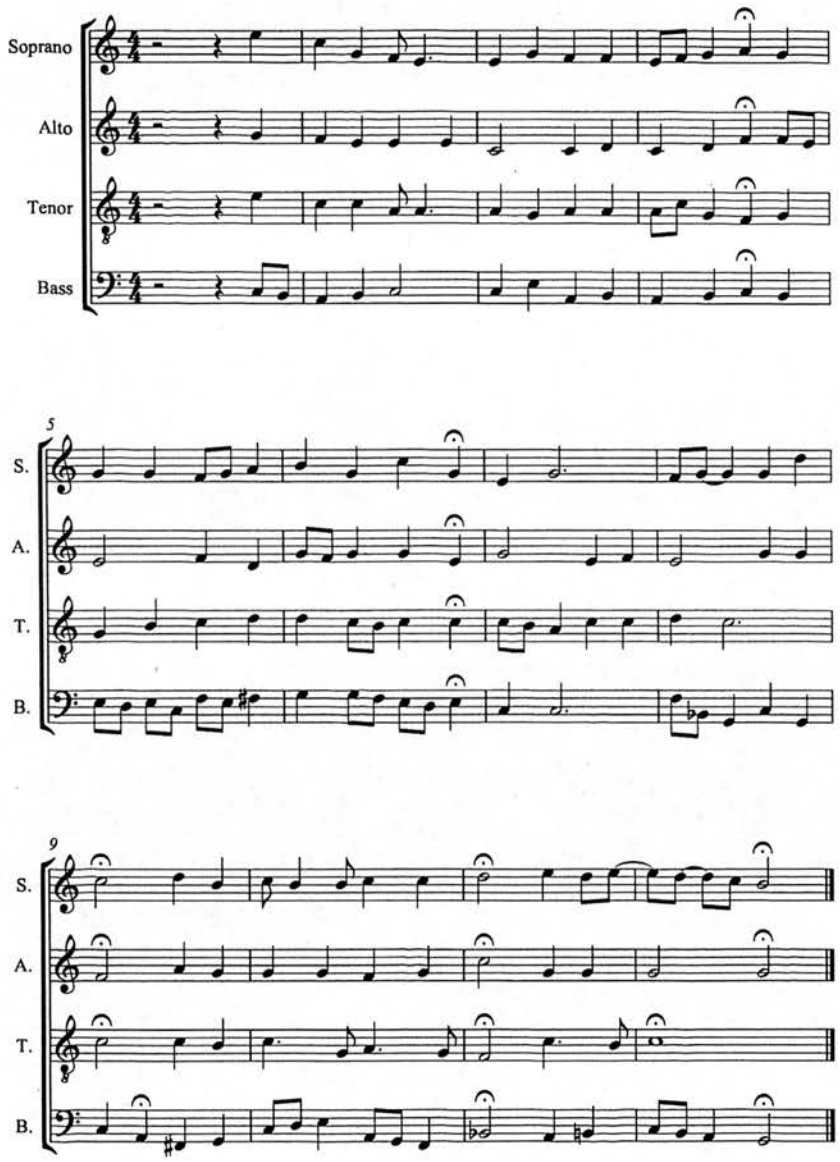


Figure B.7: ***(4P7)*** *Example generated from a B-S-A-S-T network with non-smoothing, flat HMMs (sampled at quaver rate).*

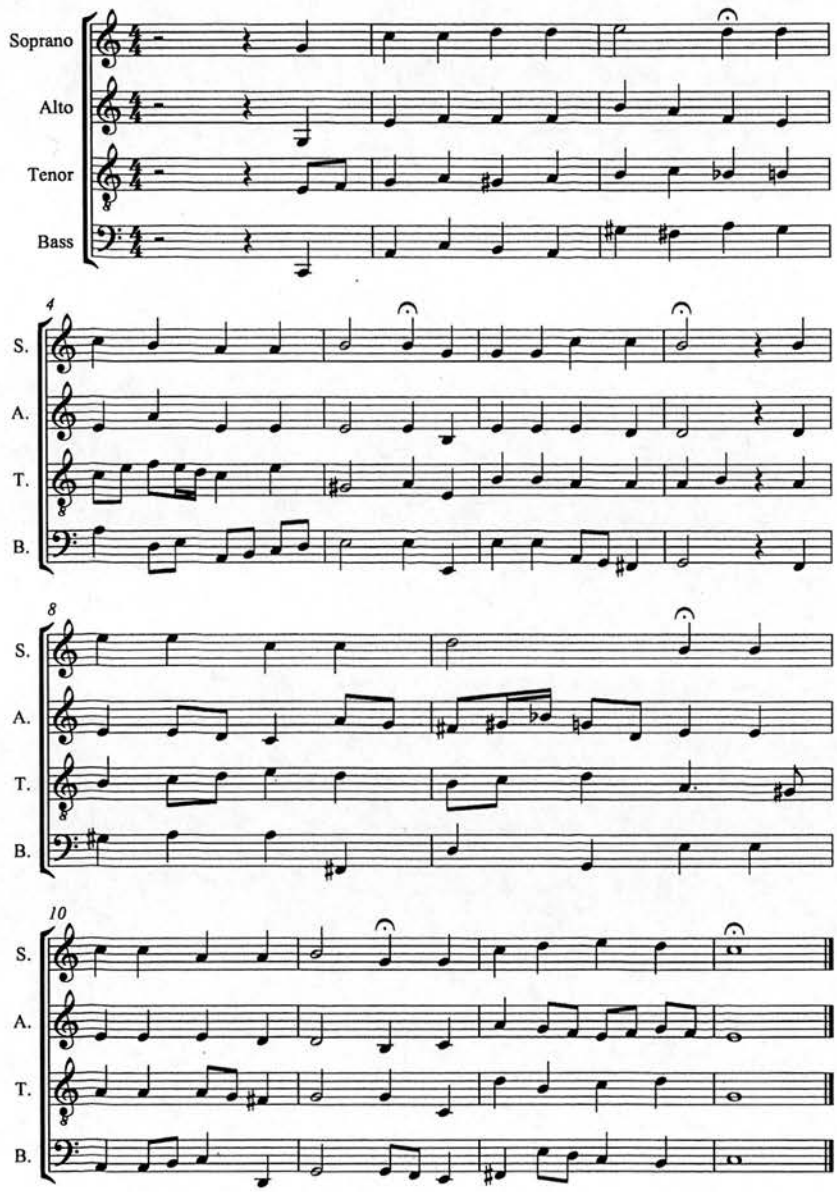


Figure B.8: ***(4P8)*** *Harmonisation of the melody of Bach chorale K211, BWV 367, taken from (Allan, 2002).*

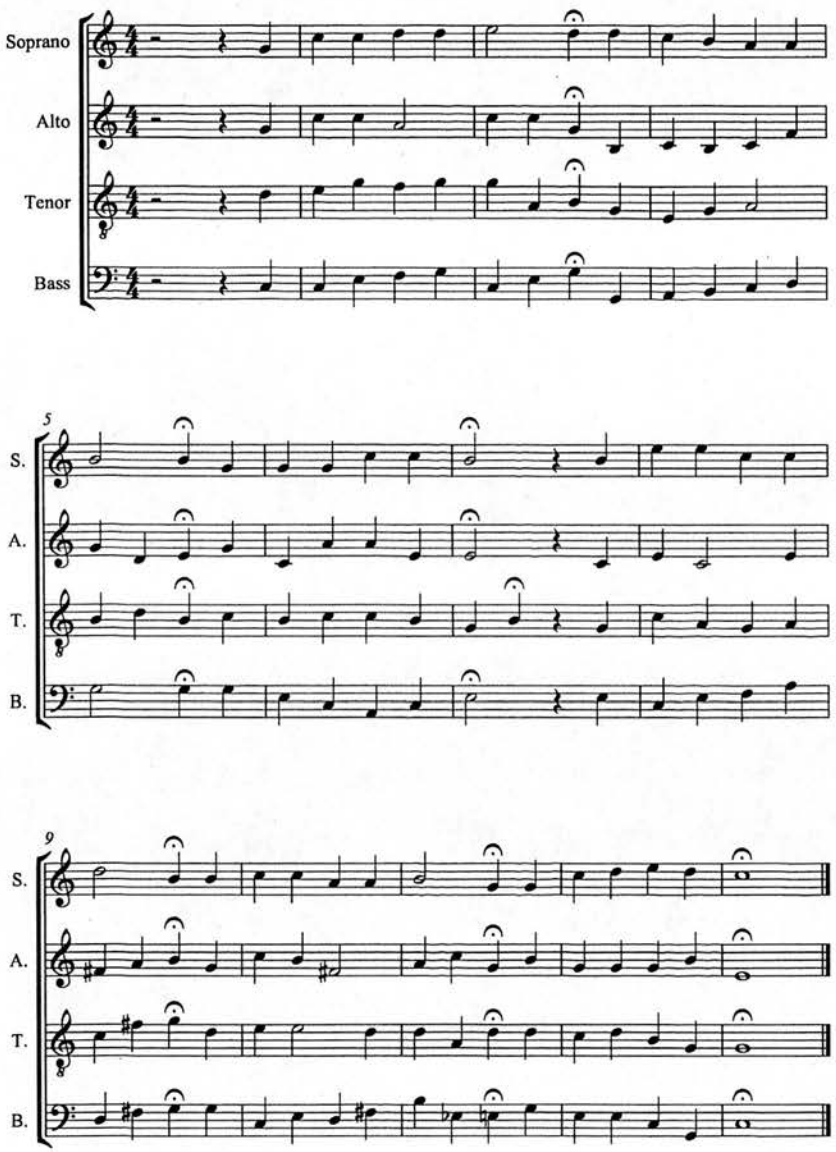


Figure B.9: ***(4P9)*** *Example generated from a BS-AT network with smoothing models (sampled at crochet rate).*

Bibliography

Allan, M. (2002). *Harmonising Chorales in the Style of Johann Sebastian Bach.* Master's Thesis, University of Edinburgh. (http://www.tardis.ed.ac.uk/~moray/papers/allan2002.pdf)

Allan, M., & Williams, C. K. I. (n.d.). Harmonising Chorales by Probabilistic Inference. In *Advances in Neural I]nformation Processing Systems, volume = 17, publisher = MIT Press, address = Cambridge, MA, USA, editor = L. K. Saul and Y. Weiss and L. Bottou, pages = 25-32, year = 2005,.*

Alpaydin, E. (2004). *Introduction to Machine Learning.* MIT Press.

Bengio, Y. (1999). Markovian Models for Sequential Data. *Neural Computing Surveys*, 2, 129-162.

Bengio, Y., & Frasconi, P. (1995). An Input Output HMM Architecture. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in Neural Information Processing Systems* (Vol. 7, p. 427-434). Cambridge, MA, USA: MIT Press.

Bod, R. (2002). A General Parsing Model for Music and Language. In C. Anagnostopoulou, M. Ferrand, & A. Smaill (Eds.), *ICMAI* (Vol. 2445, p. 5-17). Springer.

Burton, A. R., & Vladimirova, T. (1999). Generation of Musical Sequences with Genetic Techniques. *Computer Music Journal*, *23*(4), 59-73.

Cambouropoulos, E. (1998). *Towards a General Computational Theory of Musical Structure.* Phd Thesis, University of Edinburgh.

Conklin, D. (2003). Music Generation from Statistical Models. In *Proceedings of the AISB 2003 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences* (p. 30-35). SSAISB.

Conklin, D., & Witten, I. H. (1995). Multiple Viewpoint Systems for Music Prediction. *Journal of New Music Research, 24*, 51-73.

Cope, D. (1991). *Computers and Musical Style.* Oxford, UK: Oxford University Press.

Ebcioglu, K. (1993). An Expert System for Harmonizing Four-Part Chorales. In S. M. Schwanauer & D. A. Levitt (Eds.), *Machine Models of Music* (p. 385-401). Cambridge, MA, USA: MIT Press.

Farbood, M., & Schoner, B. (2001). Analysis and Synthesis of PalestrinaStyle Counterpoint using Markov Chains. In *Proceedings of the International Computer Music Conference* (p. 471-474). International Computer Music Association.

Fine, S., Singer, Y., & Tishby, N. (1998). The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning Journal, 32*(1), 41-62.

Gurney, K. (1997). *An Introduction to Neural Networks.* Bristol, PA, USA: Taylor & Francis, Inc.

Hild, H., Feulner, J., & Menzel, W. (1991). HARMONET: A Neural Net for Harmonizing Chorales in the Style of J S Bach. In J. E. Moody, S. J. Hanson, & R. Lippmann (Eds.), *Advances in Neural Information Processing Systems* (Vol. 4, p. 267-274). San Francisco, CA, USA: Morgan Kaufmann.

Jackson, P. (1990). *Introduction to Expert Systems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Jelinek, F. (1985). *Robust Part-Of-Speech Tagging using a Hidden Markov Model.* (Tech. Rep.). IBM TJ Watscon Research Centre.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. New Jersey, USA: Prentice Hall.

Kramer, O., Stein, B., & Wall, J. (2006). Ai and Music: towards a Taxonomy of Problem Classes. In G. Brewka, S. Coradeschi, A. Perini, & P. Traverso (Eds.), *Proceedings of 17th European Conference on Artificial Intelligence* (p. 695-696). Riva del Garda, Italy: ECAI.

Lerdahl, F., & Jackendoff, R. (1983). *A Generative Theory of Tonal Music*. Cambridge, MA, USA: MIT Press.

Löthe, M. (1999). Computerized Composition of Minuets in Early Classical Style. In *Proceedings of the AISB'99 Symposium on Musical Creativity.* Edinburgh, UK: AISB.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press.

Mitrani, I. (1998). *Probabilistic Modelling*. Cambridge, UK: Cambridge University Press.

Murphy, K. P., & Paskin, M. A. (2002). Linear Time Inference in Hierarchical HMMs. In T. G. Diettrich, S. Becker, & Z. Gharamani (Eds.), *Advances in Neural Information Processing Systems* (Vol. 14, p. 833-840). Cambridge, MA, USA: MIT Press.

Phon-Amnuaisuk, S., Tuson, A., & Wiggins, G. (1999). Evolving Musical Harmonisation. In *Proceedings of the Fourth International Conference on Artificial Neural Networks and Genetic Algorithms.*

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE, 77*(2), 257-285.

Russell, S., & Norvig, P. (1995). *Artificial Intelligence: a Modern Approach*. New

Jersey, USA: Prentice Hall.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, *27*, 379-423 and 623-656.

Skounakis, M., Craven, M., & Ray, S. (2003). Hierarchical Hidden Markov Models for Information Extraction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapoulco, Mexico.* San Francisco, CA, USA: Morgan Kaufman.

Spangler, R. R. (1999). *Rule-Based Analysis and Generation of Music*. Phd thesis, California Institute of Technology.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*. Cambridge, MA, USA: MIT Press.

Thywissen, K. (1999). Genotator: an Environment for Exploring the Application of Evolutionary Techniques in Computer-Assisted Composition. *Organised Sound*, *4*(2), 127-133.

Ueda, N., & Sato, T. (2001). Simplified Training Algorithms for Hierarchical Hidden Markov Models. In *DS '01: Proceedings of the 4th International Conference on Discovery Science* (p. 401-415). London, UK: Springer.

Weiland, M., Smaill, A., & Nelson, P. (2005). Learning Musical Pitch Structures with Hierarchical Hidden Markov Models. In A. Sedes & H. Vaggione (Eds.), *Proceedings of the 12th Journées d'Informatique Musicale* (p. 55-61). (http://jim2005.mshparisnord.net/download/8.Learning.pdf).

Wierstra, D. (2004). *A New Implementation of Hierarchical Hidden Markov Models*. Master's Thesis, Utrecht University.

Wiggins, G., Papadopoulos, G., Phon-Amnuaisuk, S., & Tuson, A. (1999). Evolutionary Methods for Musical Composition. *International Journal of Computing Anticipatory Systems*.