Imperial College of Science, Technology and Medicine
Department of Computing

Context-Free Composition: A Novel Approach to Algorithmic Musical Composition

by

Nicholas Gubbins

Submitted in part fulfilment of the requirements for the degree of
Master of Science in Computing Science at Imperial College London, September 2014

Abstract

I present an investigation into context-free algorithmic analysis of music and its use in the generation of new compositions which imitate the style of the analysed pieces. The two ways which are investigated and implemented are Markov Processes and Generative Grammars. For Markov processes, I propose my own novel multi-faceted way in which to analyse music, including separating the transition matrices per voice, and for each voice mapping rhythmic and pitch transitions separately. Similarly, a transition matrix is established for overall harmony. Finally, this same overall process is carried out both forwards and backwards temporally. For Generative Grammars, I propose and implement the ADIOS algorithm, an unsupervised, stochastic grammar-inducer, to derive a harmonic and melodic grammar which can then be used to generate new music. I also present my final program, written in C++, which creates new pieces of polyphonic music, having been trained in the aforementioned ways on 250 Bach Chorales.

Acknowledgements

Special thanks to Dr Iain Phillips for his invaluable advice and attention, and who helped make this project a reality.

I would also like to thank my family, without whom I would never have learned to sing out of tune.

‘[The Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...

Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.’

Ada Lovelace

Contents

List of Figures

Chapter 1

Introduction

In this chapter I introduce the project, including the reasons for choosing to explore Markov processes and Generative Grammars, my own personal motivation for the project, as well as helpful musical definitions for those reading this report without a musical background.

1.1 Objectives

The aim of this project was to explore the ways in which to create a program capable of analysing music and creating new musical material in the style of the input with no parameters or contextual rules applied at all. Whilst it would be easy to create monophonic music, that is music with only one voice, the type of music that I wanted to create was complex polyphonic music which abided by certain rules of construction or music theoretical parameters. As such, having discovered that Markov processes have long been utilised in the creation of novel musical material, but with a certain limitation and naivité, I have devised a new approach to applying Markov processes to musical composition by separately mapping the multiple dimensions of music in both directions temporally, improving their ability in not just creating melodic material, which they have historically been resigned to, but also in generating harmonically succinct polyphonic compositions. Similarly, in order to further explore context-free composition, I have applied Zach Solan's context-free grammar inductor, the ADIOS Algorithm [30], to harmonic and melodic structure of music in order to create new "grammatically correct" pieces. Therefore, without instilling a program with a specific set of musical rules of construction or music theoretical parameters, I set out to create a program which can read in any corpus of works in any style and create an entirely novel composition in the same style.

The training data for this project was a set of 250 Bach Chorales in MusicXML format. Bach Chorales have long served as a foundation for the training of music theory, as well as the training of past algorithmic composers due to their short length, simple four-part texture, and most importantly, their strict adherence to the rules of music theory. As a proof of concept for the ability

of the new approach to Markovian composition, I shall create new Bach Chorales, with the hope that the fundamental context-free approach to creating an algorithmic composer may be applied to pieces of more complex texture in the future.

1.2 Motivation

In what ways can a computer program be creative? Of course, there is a degree of creativity at the hand of the programmer who finds a particularly nifty way in which to efficiently translate an algorithm into a given language. However, with respect to the programs process resulting in an output, given that the computation must be algorithmic in some sense, can we define the output as creative?

While the notion of computer-based creativity might be a sour pill to swallow for some, for philosophical reasons regarding the nature of art being a strictly human endeavour, computer made creativity has already proven popular in certain fields. In the realm of visual art, Benoit Mandelbrot has developed fractals that have proven so popular that [Mandelbrot and his gallery] are paid more money for these pictures than some people get for their art. However, previous attempts at computer composed music has already been met with some animosity, particularly from musicologists. On the debut presentation of his Experiments with Musical Intelligence (EMI) program, David Copes audience responded with utter silence. Most of the participants left without speaking to [him].

In fact, beyond David Cope, there have been many instances of attempts to make a program that can compose new creative music, some of which have been more successful than others. The majority employ at their very foundation a Markovian system, which I shall discuss in the next chapter, for the generation of new material. However, as I have discovered, Markov Processes until now have been fairly limited in their application, largely being used to generate melodies.

As with any program that is designed to create a creative output, there are always rules of construction for the creative content. If designing a program that creates novel text, a vocabulary and grammatical rules of the output language are necessary. Within music, particularly the music of the Western Classical tradition, there are strict rules of grammar as defined under the auspices of *Music Theory.*

Having studied music theory for many years, and learning methods of composition, I believe that the process that I undertake should be easily replicable by a computer. The steps that I undertake in composition are:

1. Decide a key
2. Decide a form or structure and number of voices
3. Come up with a melody
4. Come up with a music theoretically correct harmony that fits the melody
5. Combine the harmony and a melody

With these steps in mind, I believe that the simple decisions of which key, how many voices and a form are arbitrary, and as such can simply be simulated via randomisation. Similarly, a computer can and have been easily programmed in a rule based program how to compose pieces of music in a rigid style. However, I wanted to improve the pre-existing techniques such as Markov Processes and Generative Grammars, two simple ideas, and create a context-free composition program which does not need to be supplied with any rules of construction, and which can emulate new pieces in the style of the input. In essence, every time the program is run, the training that I had, learning possible sequences of notes, rhythms and chords, is emulated, and the output reflects the gleaned rules.

1.3 Musical Definitions

Unlike other art forms or languages, music is a multi-dimensional entity which intertwines multiple frequencies, rhythms and sonorities in order to create a piece of music. As such, the composition of music is a complex process in which there is a lot of variability with respect to the decisions that can be made. There are certain fundamental elements to music which it would help to understand:

- Pitch - a frequency split into twelve classes: A, A#, B, C, C#, D, D#, E, F, F#, G, G#. The pitches repeat in this order, with each iteration representing an octave.
- Rhythm - a duration of time defined relative to a number of beats per minute and the time signature of the piece.
- Note - a unit of music comprising a pitch value and a rhythmic value, thus defining a frequency and temporal location.
- Time Signature - a signifier of how many rhythmic units appear per bar.

- Mode - a sequence of pitches defined by the number of steps between each pitch. The most common modes used in western classical music are the major and minor modes. For example, C major is defined as C, D, E, F, G, A, B (, C). As such, one can infer that a major mode is defined as the following sequence of pitches: $X, X+2, X+4, X+5, X+7, X+9, X+11$. Most modes in the western classical tradition have seven unique pitches.
- Key Signature - the relative tonal home of a piece, where the foundation of the piece revolves around a mode starting on a certain pitch. On a score, the key signature is indicated by a sequence of sharp or flat signs preceding the first pitch.
- Melody - A sequence of notes across time associated with one voice. On a score, melody can be read horizontally on one voice's stave.
- Chord - A combination of two or more notes co-located in time across multiple voices. On a score, a chord can be read vertically at a given rhythmic point in a measure across multiple voices' staves
- Harmony - A sequence of chords.
- Cadence - a predefined series of chords that harmonically signify the end of a musical section or subsection. The most widely used cadence in the Western Classical Tradition is the terminal Perfect Cadence (V-I).

These elements are combined in certain ways to create pieces of music. Music theory lays out parameters in terms of what sequences of notes can appear melodically, as well as how voices should interact harmonically, and which chord sequences are permitted.

1.4 Outline of the Report

In chapter 2 I explain some of the different ways in which music is stored electronically, since it was an important consideration in the development of this project. In chapter 3, I discuss the history of algorithmic composition and musical analysis. In chapters 4 and 5, I explain the high level concept behind the composition platform, and demonstrate the system that I have developed, including the new approach to bidirectional Markovian analysis of music. After that in chapter 6, I explore some of the outcomes of the three different compositional techniques that are incorporated within the system, including the benefits and capabilities of each of the three, as well as some ways in which the system can be improved.

1.5 Creativity Defined

Since this project deals with automatic generation of what is often considered to be *creative* material, it is interesting to consider what it means to be "creative". In order to ascertain whether

or not a computer program is capable of producing a *creative* work, it is important to define what exactly is meant by creativity, beyond the self evident notion that it is something simply being created. Merriam-Webster defines the adjective creative as "the ability to make new things or think of new ideas" [19], while the Oxford English Dictionary defines it as "Inventive, imaginative; of, relating to, displaying, using, or involving imagination or original ideas as well as routine skill or intellect" [29]. Interestingly, the commonality between the two definitions is the notion of innovation through "new" or "original" content. This innovation, however, doesn't require that the entire piece is entirely extra-paradigmatic, that is to say that it is entirely new in every sense. Pop songs that are *new* still use the same I-V-vi-IV chord progression that has been used for decades, while the painters since the fifteenth century have depicted the same biblical scenes. In fact, I would argue that the majority of creative work is nothing more than a re-appropriation and re-contextualisation of what has come before. The composer learns to compose through studying counterpoint, four-part writing, chromatic harmony, modal and even a-tonal harmony from examples throughout history. Very few artists spark an entirely new artistic language, but when they do, it is never an overnight shift, but rather a gradual evolution, and each shift will have discernible causes, rather than being divinely-inspired or an entirely spontaneous creation. As such, I believe that a computer can equally be trained armed with the language of previously composed work, and the validity of the output be as *creative* as a human.

Chapter 2

Musical Codification

Since it is necessary to analyse multi-part pieces of music as data, it is necessary to contemplate the many formats in which music has historically been stored electronically. The ideal format should fulfil certain criteria for the development of an autonomous composer. As such, the notation system should:

1. Represent different pitches and note lengths
2. Accommodate pieces consisting of multiple parts
3. Suitable representation of meta-data of a piece of music including key and time signature
4. Easy to manipulate i.e. transposition

It would also be desirable for the notation system to allow for easy translation to sheet music or audio files once output has been generated by a program.

2.1 MIDI

MIDI, an acronym for Musical Instrument Digital Interface, was introduced in 1983 and has largely remained unchanged since its inception [14]. MIDI is probably the best known electronic music format given its vast influence in the realm of synthesisers as well as digital recording and even mobile ringtones. However, MIDI is not actually a musical language per sé, rather, it is a communication protocol agreed among manufacturers of music equipment, software and computers which "describes a means for music systems and related equipment to exchange information and control signals" [25].

MIDI, allows for any electronic musical device to communicate with one another via messages. For example, the messages from a MIDI guitar will be sent in real time to a computer and command a sound module to play the notes encoded within the message, which correspond to the frequencies

and volumes of what the guitarist is playing. The two most common MIDI messages ae Note On and Note Off. When a device wants the sound nodule to play an A, a Note On message will be sent, along with the data about pitch, velocity and the appropriate channel information. This note will now play until the Note Off message is received from the same instrument [16].

Ultimately, MIDI is much better suited to performance, such as the use of MIDI keyboards to create the sound of a violin or synthesiser in a live setting with a real-time response. In fact, to elaborate, it might be useful to distinguish between two states of a piece of notated music. The first state is the score, the abstract representation of a piece of music. The score does not represent the physical sound of the performance of a piece, rather it is a map and representation to how a piece ought to sound, which may be very detailed in terms of ornamentation and dynamics, or may not be. The second state of music is the performed piece, whether at a live performance or on a recording. This second state takes up a given amount of time, and is unique in its harmonic qualities, both of the instrument and the space in which the piece is performed. The latter state can be better understood as an interpretation of the former, and it is this second state that more appropriately aligns with MIDI, since MIDI consists of the signals of each attack and release of each note in time. The MIDI recording can be manipulated and edited, however, the end result will still be an interpretation of a piece that can be played back. Of course, the former state can be inferred from the latter, however, it will lack a lot of the meta-data that one associates with a score, and will be a transcription of the performance and performer's interpretation of the piece rather than the original piece itself.

The drawback to MIDI in the case of an algorithmic composer, is that while there are readily accessible MIDI files containing many works that I would use for my project, the fact that it represents a performance rather than a score means that there is more overhead in translating the MIDI file into a desirable data format which can be effectively analysed. For example, to translate the MIDI file into a meaningful data set, the file will have to be parsed, dealing with Note On and Note Off separated by a certain amount of quantized time depending on the tempo to distinguish a note, rather than simply being given a codified minim at the second beat of the bar at pitch A.

2.2 ABC Notation

ABC Notation was developed by Chris Walshaw in the 1980s as a replacement for writing MusicTeX. Originally a front-end for generating TeX, ABC was standardised and is now commonly used for writing out folk melodies and traditional tunes [33]. The format uses ASCII characters to represent note values and lengths, as well as time signature and key signature declarations. An example of an ABC format is seen in Figure 2.1. The resulting score after compilation via one of many free softwares would result in the score seen in Figure 2.2.

```
X:1
T:Speed the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA|
  GABc dedB|dedB dedB|c2ec B2dB|A2F2 G4:|
|:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df|
  g2gf g2Bd|g2f2 e2d2|c2ec B2dB|A2F2 G4:|
```

Figure 2.1: An example of ABC notation taken from [34]

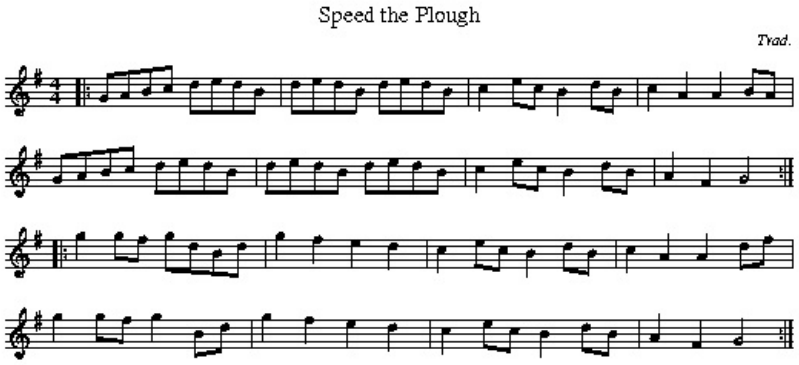


Figure 2.2: The compiled result of Figure 2.2 taken from [34]

Unlike MIDI, ABC Notation represents the score as opposed to a performance, which is preferable for the data supplied to an algorithmic composer. However, ABC Notation is a very small package, only capable of representing a single melody line. While it has impressive features, for example allowing the representation of any complex metrical subdivision, it does not appear to be well suited to the complex analysis of polyphonic music. Similarly, there is no provision for modulation. It is possible, however, that one could parse multiple ABC files, each of which represents an individual voice, however, given the alternative format MusicXML, this seems unnecessary and contrived. There are features of ABC Notation that are desirable, however, such as the ability to create a graphical score and the ability to create a MIDI file for the graphical score for playback, as well the human readability of the notation. However, on the whole it seems an unsuitable candidate for this project.

2.3 MusicXML

Just as Extensible Markup Language (XML) has become an easy to use standard used ubiquitously throughout the Internet, MusicXML too has become the standard for the representation of scores electronically. Not only is it human-readable, it is incredibly flexible, capable of producing all facets of a score, including text, ornaments, full orchestral staves, and dynamics, but also other notation formats such as a Jazz lead sheet.

MusicXML, standardised by IEEE as The IEEE 1599 Standard, was developed by music software

company Recordare LLC in 2002 so that they had a universal translator between music notation programs [2]. The standard uses XML clauses to encode all data logically in a file. For example,

$$< clef\_type = G\ staff\_step = 2\ event\_ref = c1/ >$$

will lead to a treble clef appearing on a score. These XML clauses are then layered, starting with the general layer, exposes all the meta-data of a work, including title and author. Next, there is the logic layer, which describes the core format, including key and number of staves if a stave format is being used, as well as assigning staves to instruments. Then, and most importantly is the Logically Organised Symbol layer, for which there is one entry per voice, and this consists of a sorted list of musical events, the notes themselves. The layers reference one another, for example the LOS layer references the stave variable assigned in the logic layer. Similarly, the referencing of time locations allow for different entries on the LOS layer to synchronise events, thus allowing for polyphonic scores. Beyond this, there is a Structural layer, capable of representing the overall form of a piece of music, such as a ternary form piece consisting of three sections ABA. There is also a Performance layer, which can incorporate MIDI files of the piece [1].

The following images show the translation of musical objects “p0e0” to “p0e11” and how they form a melodic phrase, with timing and hpos both representing the length of the note:

```
<ieee1599>
  <logic>
    <spine>
      <event id="p0e0" timing="0" hpos="0"/>
      <event id="p0e1" timing="1" hpos="1"/>
      <event id="p0e2" timing="1" hpos="1"/>
      <event id="p0e3" timing="2" hpos="2"/>
      <event id="p0e4" timing="1" hpos="1"/>
      <event id="p0e5" timing="1" hpos="1"/>
      <event id="p0e6" timing="2" hpos="2"/>
      <event id="p0e7" timing="1" hpos="1"/>
      <event id="p0e8" timing="1" hpos="1"/>
      <event id="p0e9" timing="2" hpos="2"/>
      <event id="p0e10" timing="1" hpos="1"/>
      <event id="p0e11" timing="1" hpos="1"/>
    </spine>
    <los>...</los>
  </logic>
</ieee1599>
```

Figure 2.3: An example of MusicXML taken from [1] p30

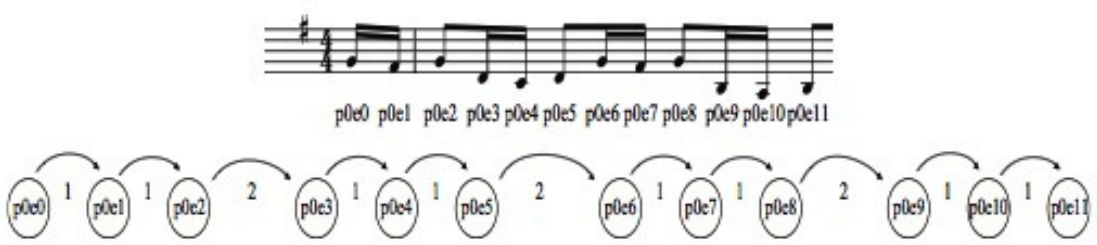


Figure 2.4: A graphical illustration of the result of Figure 2.3 taken from [1] p34

Given MusicXML's vast capability of different musical styles, and the detail which can be encoded, MusicXML now serves as the main interface between music software. If a composer has been using the composition software Sibelius, and they want to transfer it to Finale, MusicXML is now the standard go-between, as both softwares are capable of writing and interpreting MusicXML files. As such, if an algorithmic composer could read from and write to MusicXML files, the result could be easily ported to a score writing software and shown in its graphical form. Further, due to its popularity, MusicXML has a vast catalogue of compositions which could be used as a dataset for the algorithmic composer. Given the type data that is in encoded in the standard, it appear that MusicXML perfectly satisfies all the requirements laid out above for an electronic musical notation system.

2.4 Conclusion

There have been multiple developments in the formats in which to store music electronically, many of which could be appropriate for this project. However, due to the fact that this project is concerned with the composition of a piece of music in a more academic sense, as opposed to the performance of a piece, it appears that MusicXML is the best choice for the data format given the detailed musical information it can represent, the vast availability of scores in the format, the ability to interface with score editing software such as Finale or Sibelius, in case the user of the program were to want to make any hand edits to the output, and also due to its standardisation by IEEE, which means that the final program will have further use so long as MusicXML isn't abandoned.

Chapter 3

A History of Algorithmic Composition

In this section I shall explore some of the main algorithmic techniques that have been applied in the past to the creation of algorithmic composition. Each approach has its own benefits and drawbacks, and certain techniques are better suited to certain aspects of algorithmic composition than others. I shall focus on Markov processes and Generative Grammars, however, I shall also explain other developments in algorithmic composition such as Transition Networks and Neural Networks.

3.1 Overview

Algorithmic Composition has been attempted to varying degrees of success even before electricity, let alone the transistor, was discovered. In the eleventh century, Benedictine monk Guido of Arezzo wrote the work *Micrologus*, in which he exposed the first algorithmic way in which to automatically create melodies out of any text. Developed for sung mass, Guido assigned certain pitches to certain vowel sounds. From that point, he invented a system of neumes, multiple pitches grouped together to form phrases, and caesuras, pauses in the music initiated by a cadential figure. Given any text, the vowels could be sung at the designated pitches, and neumes formed from the textual phrase [11].

Centuries later, composers in the Classical period began to use randomisation as a form of algorithmic composition. Dice music, or Musikalisches Würfelspiel, became a popular way to compose, and even Mozart composed in this style through analysis of the manuscript of his piece *K. 516f* from 1787. In fact, dice music is often attributed to Mozart, however, there are earlier examples, the earliest of which is German composer Johann Philipp Kirnberger's *The Ever-Ready Minuet and Polonaise Composer* [22].

Dice music is a very simple algorithm, which requires a single die and a selection of pre-composed musical phrases. The phrases are split into groups of six, with each phrase in each group being assigned the numbers between one and six. Then, the composer rolls the dice, selecting the phrase numbered with the roll's outcome from the first group, and then continually rolling, selecting from

each consecutive group [20].

As I shall demonstrate in this section, with the dawn of computers, music has become intrinsically linked with processors. However, the revolution in music has largely affected the performance, storage and dissemination of music, with MIDI, mp3s and even recently with streaming services such as Spotify and iTunes. However, one area which is still in its infancy is the creation of novel music by a computer.

3.2 Markov Processes

Markov models are credited to Russian mathematician Andrey Markov (1856-1922) who first wrote about the probabilistic chains in his 1906 paper Extension of the law of large numbers to dependent quantities [18], in which Markov mapped the sequence of 20,000 letters in A. S. Pushkin's poem *Eugeny Onegin* and discovered that the probability of a vowel following a vowel was 0.128. It was actually later in 1926 when Russian mathematician Sergey Bernstein coined the interchangeable phrases "Markov chain" and "Markov model" referring to Markov's paper [3].

A Markov Model is a stochastic process, that is to say a process combining probability and statistics. The model provides a matrix of probabilities associating states to subsequent states, such that if at any point in time, if the current state is X, the likelihood of the state $X + 1$ is known, based on the matrix. The sum of the transition probabilities for each state must equal 1.

For example, the Markov chain for choosing Pick 'n' Mix at a sweet shop where there are three types of confectionary in the bins: chocolate pieces, gummy bears, and peanuts. After a certain item has been chosen, the matrix shows the probability of what the next choice will be.

Up until now, all Markov processes have been explained as first-order Markov chains, where only one state X is considered when designating the values into the probability matrix. However, this system can be expanded into $n - order$ Markov processes, where the order indicates the number of previous states that are relevant in designating the transition probabilities.

Markov chains have had a variety of uses in computing, notably in the PageRank algorithm used by Google's search engine to order the results for a search request [4]. However, it has also been used in text generation, such as Bell Lab computer scientist Bruce Ellis' Mark V Shaney, a bot which commented on news websites with the text deriving from a word-based Markov process using all other comments as the basis of its probabilities. The result was convincing, yet often absurd given the incongruity of the sentences in each post. For example, in November 1984, Mark commented:

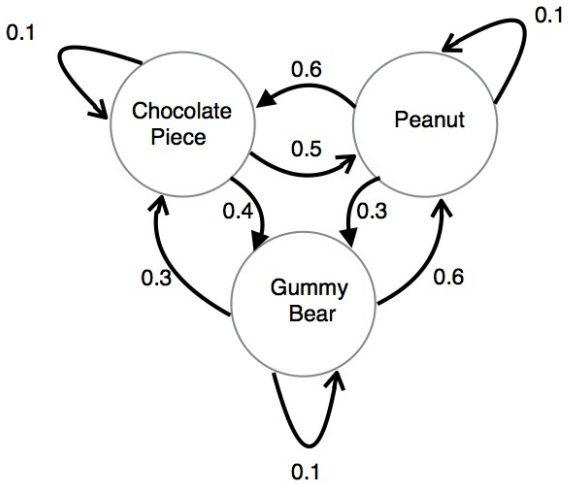


Figure 3.1: A diagram illustrating the states and transition probabilities in a Pic 'n' Mix scenario

Previous Choice \ Next Choice	Chocolate Piece	Peanut	Gummy Bear
Chocolate Piece	0.1	0.5	0.4
Peanut	0.6	0.1	0.3
Gummy Bear	0.3	0.6	0.1

Figure 3.2: The transition matrix associated with Figure 3.1

> "It looks like Reagan is going to say? Ummm... Oh yes, I was looking for. I'm so glad I remembered it. Yeah, what I have wondered if I had committed a crime. Don't eat with your assessment of Reagan and Mondale. Up your nose with a guy from a firm that specifically researches the teen-age market. As a friend of mine would say, "It really doesn't matter"... It looks like Reagan is holding back the arms of the American eating public have changed dramatically, and it got pretty boring after about 300 games." [27]

In this case of PageRank and Mark V Shaney, the transition probabilities weren't arbitrarily decided by the programmers, rather they used a dataset to calculate the probabilities. Especially with its ability to generate grammatically novel text, regardless of the semantic meaning of the individual words, it becomes immediately apparent that Markov processes are clearly well adaptable to music generation, where there are no semantics, only grammar in the form of music theory.

Markov models can be used by an algorithmic composer by analysing a corpus of works with any order of n, of course limited by the length of the longest piece in the corpus. At this point, using pseudo-randomised choices weighted by the probabilities, the composer would be able to create

novel music. The states could be a variety of things, from either a single note for one voice, to a harmonic or rhythmic motif, to a multi-bar phrase, or even a multi-voice chord. If using multiple voices, essentially the outcome would be a harmony generator. Further, the corpus of work that is analysed will be weighted towards the stylistic traits of the composer or composers. If we consider a word-based Markov process generating text whose input was the works of Shakespeare, both the vocabulary and high probability word couplings will be Shakespearean. As such, the output will be Shakespearean. This is a useful property if the algorithmic composition program is attempting to emulate the style of a given composer.

Markov processes have already been used thoroughly in the generation of melodic material. Noted modernist composer Iannis Xenakis used Markov models to generate some of the melodic material that contributed to his final compositions, such as the 1958 piece *Analogique A* [35]. MIT Professors F.P. Brooks, A.L. Hopkings, P.G. Neumann and W.V. Wright created a program which analysed thirty-seven Bach chorale melodies, and analysed them up to the eighth order [5]. The pieces were transposed to a common key, had a common length of 8 bars and one beat, were all in the 4/4 time signature and were all within a range of four octaves. However, in this case, there were also certain parameters that were enforced, such as not allowing for any rests, demanding there be melodic motion on certain beats of the bar, and melodies had to end on the tonic. The results can be seen in Figure 3.3.

Another way in which Markov models have been used to create new material is instead of analysing a corpus, simply defining transition probabilities for precomposed segments of melody, and then using weighted randomisation in order to recombine these segments in a novel order. This was carried out by Kevin Jones and the result is seen in Figure 3.4.

There are certain problems that arise with pure Markovian composition. Firstly, the lower the order of the process, the more *random* the output is likely to be. Considering a note-based Markov model which can create melody lines, a melody line is not simply a series of notes that never ends, rather it has phrases, rests and a degree of symmetry. One could imagine a first-order Markov produced melody line meandering with no motivic structure. Of course, the order can be increased, but as the order increases, the likelihood of a given sequence of notes having ever occurred decreases. However, in music, there tend to be short 4-5 note harmonic or rhythmic motifs which are regularly used. $n - order$ is increased, the repeated patterns will be more heavily weighted, and thus the composer becomes more motif oriented. This phenomenon was also explored by Brooks et al in the generation of the music seen in 3.3. Further, the notion of motivic development is immediately incongruous with Markov produced music, since the composer only considers the immediate past, not phrases that have appeared much earlier in a piece.

Markov processes are very useful for melodic phrase generation, or a generating a harmonic skeleton, but it becomes increasingly complex interleaving the dimensions of a Markov produced melody, a Markov produced harmonic sequence, and a Markov produced accompaniment. In fact, as far as melodic generation is concerned, if the states are only considered as being a pitch, then the output will only be a sequence of pitches, devoid of any note lengths, which is not a melody line but a

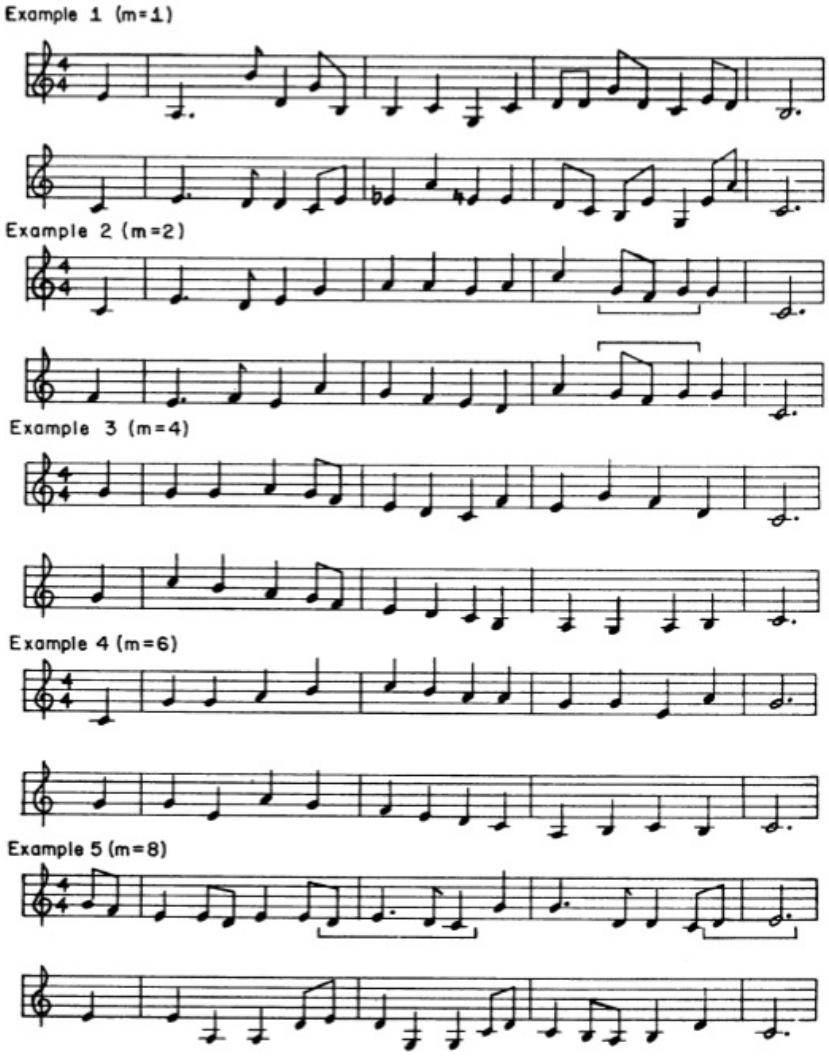


Figure 3.3: Results of Brooks, Hopkings, Neumann and Wright's Markovian melodic generator [5]

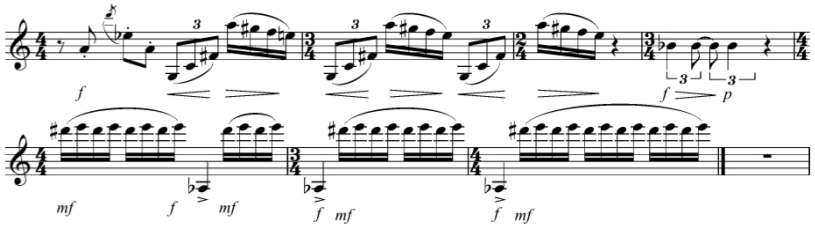


Figure 3.4: Results of Kevin Jones' melodic generator taken from [22], p76

melodic contour. Of course, one could analyse the corpus using pitch-note value states, however, his makes the rhythmic and pitch states dependent on one another, and then relationships between like-pitched but separate-rhythmic values will be missed.

Finally, even at the stage of analysis, if one attempts to create a harmonic Markov chain, the majority of music is not simply four part, homorhythmic choral music. Rather, voices are polyphonic and interleave, and there are many points of ambiguity such as suspensions of notes which offset the resolution for a chord.

Therefore, despite the appropriateness of Markov processes for creating new musical material, in their most naive form the output is likely to be mechanical and to lack any musical form. Simply put, Markov processes are ideal for mapping one dimension of any data, and as a result, have largely been resigned to melodic generation, the simplest type of composition. However, since Markov processes serve as the fundamental platform for many algorithmic compositional techniques, and because of its rooting in probability, they serve as the perfect foundation for a context-free composition program. As such, I have developed a way to synthesise multiple transition matrices representing not only the melodic, rhythmic and harmonic dimensions of music per voice, but also in two directions temporally such that self-contained pieces can be composed using just simple Markov processes. Unlike Brooks, Hopkings, Neumann and Wright, the program will need no parameterisation or rules to be encoded, and as such the system is an entirely context-free compositional program.

3.3 Generative Grammars

Generative Grammars derive from Noam Chomsky's basic linguistic model and hierarchy as described in his 1956 work *Three models for the description of language* [6]. For any formal language, there exist certain principles and structures which dictate how to generate a well-formed sentence, where a sentence consists of a combination of units, and each unit may itself be made up of subunits recursively. With a given alphabet, it is therefore possible to create new well-formed sentences, regardless of semantic meaning, armed only with the alphabet and formal rules. As it happens, at the very basis of Generative Grammars is a finite-state Markov process.

For example, Gisbert Fanselow offers an example language hierarchical structure, as seen in Figure 3.5, consisting of the following elements: sentence (S), nominal phrase (NP), verbal phrase (VP), verb (V), prepositional phrase (PP), adjectival phrase (AP), adverb (Adv), adjective (A), preposition (P), determiner (D), and noun (N). Note that a determiner is equivalent to an article, and brackets indicate optionality [10]. Once armed with this structure, we can define our own alphabet as seen in Figure 3.6. We can now create well formed sentences, such as Figure 3.7. However, we can also create the well-formed, yet semantically nonsense sentence in Figure 3.8.

S	→	NP	VP		
VP	→	V	(NP)	(PP)	
AP	→	(Adv)	A	(PP)	
PP	→	P	NP		
NP	→	(DET)	(AP)	N	(PP)

Figure 3.5: Example grammatical structure taken from [10] p26

N = {dog, tree, Nick}
DET = {a, the}
V = {ate, saw}
A = {tall, black}
Adv = {quite, very}
P = {under, in}

Figure 3.6: An example alphabet to be used with the structure in Figure 3.5

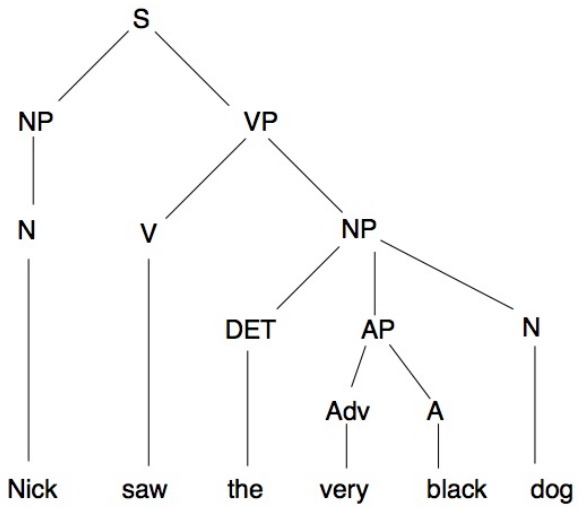


Figure 3.7: Example sentence and hierarchical derivation using Figures 3.5 and 3.6

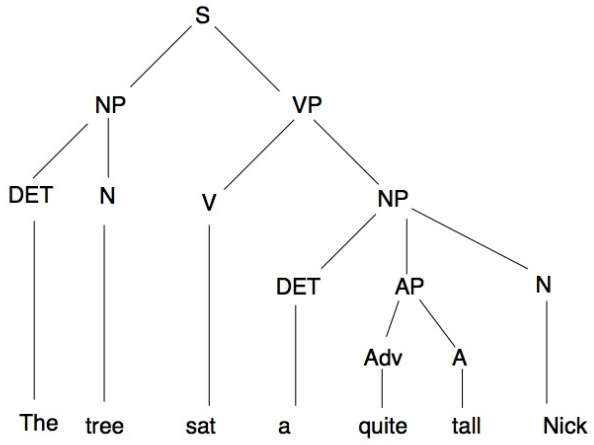


Figure 3.8: Example sentence and hierarchical derivation using Figures 3.5 and 3.6

Generative Grammars are also capable of grammatical transformation, in which the grammatical integrity of a sentence is maintained, but the order of units is changed. A simple example would be the translation of a sentence from the active to the passive voice: “The dog saw the cat” translates to “The cat was seen by the dog”. From a simple translation, a new sentence, and as such new

material, has been generated.

In terms of musical generation, generative grammar algorithms have proved particularly effective in the generation of harmonic structures. Mark Steedman created an algorithm which created Jazz progressions armed with the grammatical rules of Jazz theory, as well as the possible substitution rules [31]. An example of the chord sequence analysis performed by Steedman's program is in Figure 3.9.

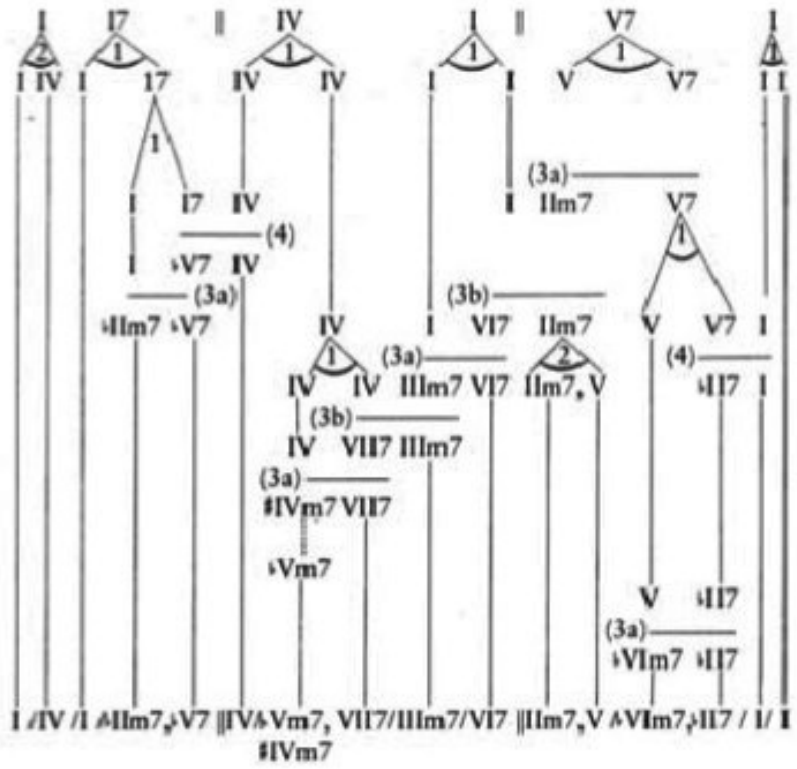


Figure 3.9: The result of a chord analysis by Mark Steedman's program described in [31]

Again in the realm of Jazz, Philip Johnson-Laird created an algorithm, which, armed with certain rules, could create both a chord sequence as well as a convincing walking bass line [15].

Generative Grammars have largely proved useful in musical analysis, serving as the basis of 20th century music theorist Heinrich Schenker's Schenkerian Analysis, in which a score is recursively harmonically reduced until it has been decomposed into its fundamental structure. In fact, programs have already been written which use generative grammar to analyse scores in the Schenkerian style, such as Bryan Pardo and William Birmingham's "HarmAn" [23].

Whereas many programs have provided their systems with grammars such as Pardo and Birmingham's system, grammars can also be inferred from a dataset. This stochastic process is known as a context-free grammar in the Chomsky hierarchy. If a system can find regularities in a data set, as well as a terminal alphabet, it can infer rules of construction and can then recreate novel

music in the style of the original data. Unlike simple Markov processes, the units can represent higher level elements of a piece of music at different levels of the hierarchy of the structure, from chords, to overall form, to repeated melodic fragments, to individual voice's characteristics. An example of an algorithm which is capable of inferring grammar is Craig Nevill-Manning and Ian Witten's "Sequitur" algorithm. The algorithm successfully found repeated motifs, as well as correctly identifying final cadences in J. S. Bach's chorales, which it used to define grammatical rules [21]. However, there has been little research into the use of context-free grammars in algorithmic composition.

Ultimately, in terms of harmonic and structural generation, Generative Grammars are extremely capable for algorithmic composition. However, they have historically been used for Jazz, and have mainly been used as a rule based system in which a grammar is supplied to the system, such as Mark Steedman's program [31]. For algorithms which can infer grammatical rules, the algorithmic composer also has the advantage of adapting to a certain style that it might analyse, as opposed to simply being limited to predefined stylistic traits. However, it is always possible that an inferred grammar might miss certain rules, and in the generation of material, break rules known only to the listener. Further, as with Markov processes, a single generative grammar is only capable of generating one dimension of a new piece of music, such as the harmonic sequence, or melodic phrase structure. As such, more complex tactics must be employed to interleave the outputs of different grammatical algorithms if a piece of complex structural, melodic and harmonic character is to be composed. Therefore, I have devised a way in which to extract an alphabet from a piece of music which represents the harmonic and melodic structure of a piece of music, such that a grammar parser can attempt to infer some rules of construction of a piece of music, and recreate novel pieces based on this grammar.

3.4 Other Techniques

Now that I have explained the forms of algorithmic composition that I shall develop in this project, I will demonstrate other developments and approaches to algorithmic composition.

3.4.1 Transition Networks

As an extension of generative grammar, transition networks have been very successful in algorithmic composition. Transition networks consist of a set of finite automata. Represented as a graph, the network consists of nodes representing a state, and edges between nodes representing a possible transition. Using the theory behind Generative Grammars, a transition network could be set up consisting of transitions which represent other automata, which themselves represent lower levels of the grammatical hierarchy, as seen in Figure 3.10.

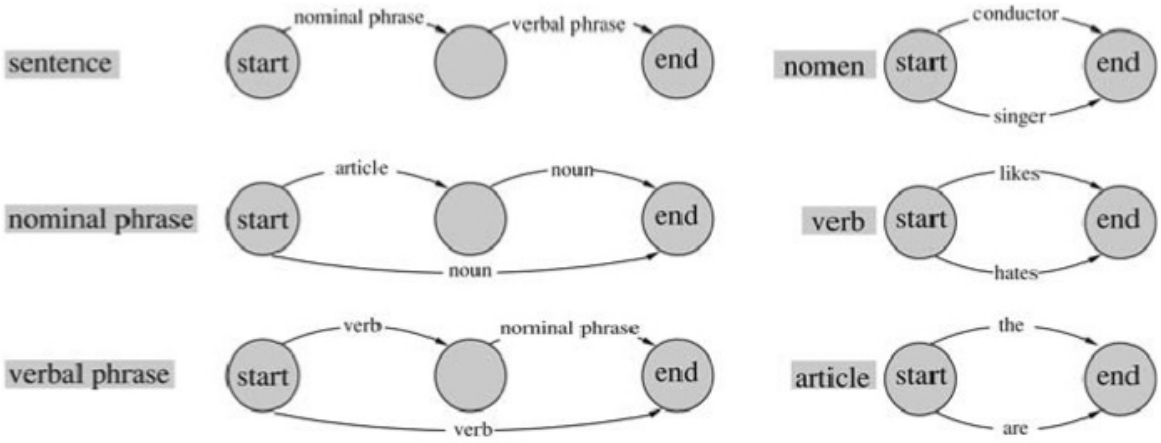


Figure 3.10: An example of a transition network graph from [22] p121

Augmented transition networks extend transition networks, allowing for transitions to contain specific instructions, such as a jump, and whole other sub-programs. Therefore, edges do not have to simply represent other finite automata, but rather could be labelled with extra information. Augmented transition networks serve as the basis of David Cope's Experiments in Musical Intelligence (EMI) software [8], probably the most capable and impressive algorithmic composer, capable of generating compositions in almost any musical style supplied only with the data of previously composed works.

There are two steps to the EMI composition process. Firstly, it parses the data, inferring grammatical rules of form, harmonic structure, melodic structure, and phrase structure. Secondly, it extracts musical phrases into a database through pattern matching, which are then used in the process Cope calls "recombinancy" [8], the transformation and restitching of pre-existing musical components. For example, two motivic units gleaned from the data could be altered if necessary to appease voice leading issues between the two, and recombined to generate a new musical phrase. This musical phrase may be combined with another musical phrase to create a section. However, "recombinancy" happens at multiple hierarchical levels and dimensions of the music, not just in simple horizontal melodic construction.

Therefore, Cope's program intelligently uses primitive building blocks and recombines them, checking they abide by any relevant rules, and then uses these new units in a hierarchical manner, combining these units with others until eventually a whole piece has been created. These hierarchical levels are multi-dimensional, representing formal, melodic and harmonic structures and rules.

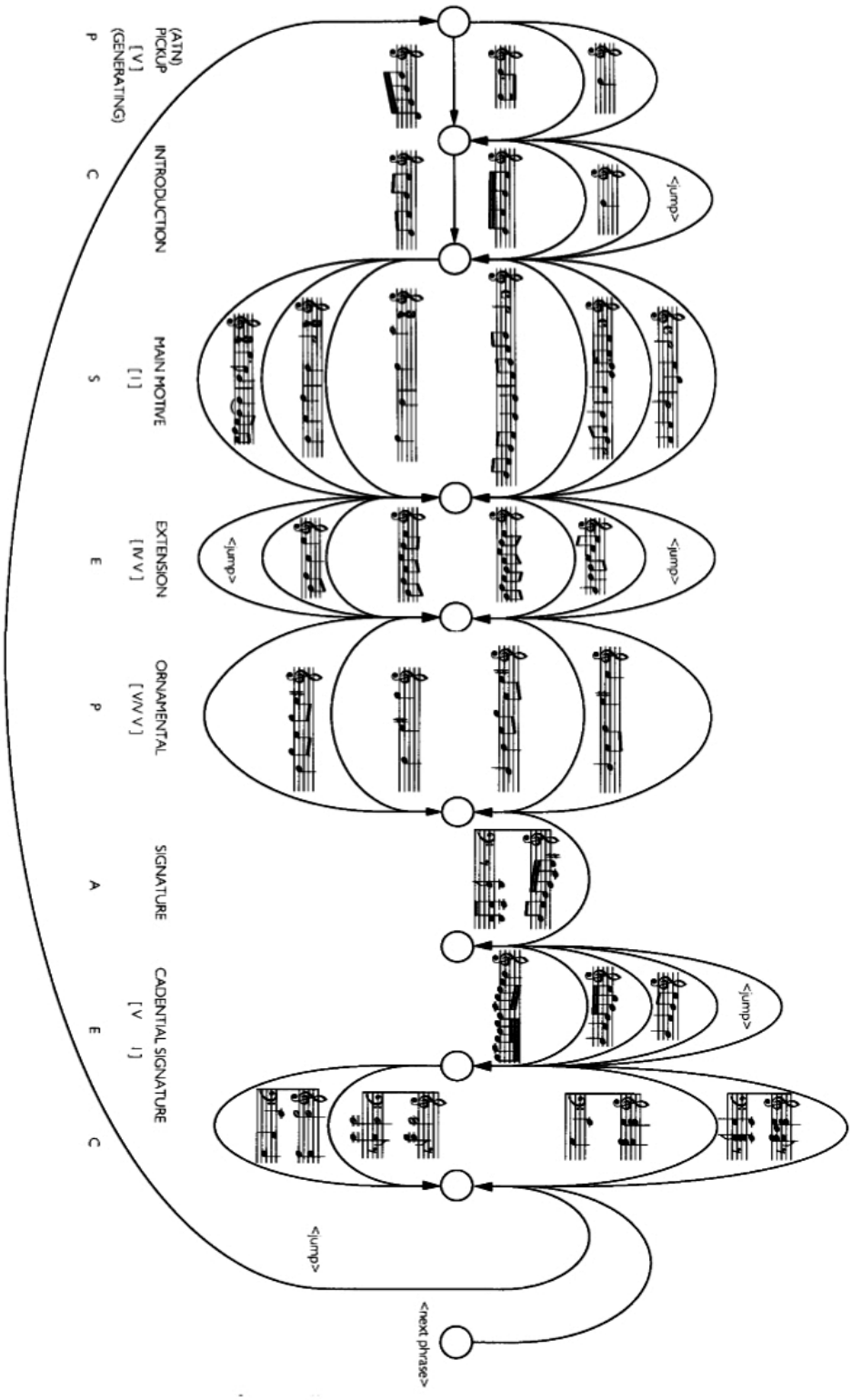


Figure 3.11: Musical illustration of an ATN taken from [7] p65

The drawback to Cope's system is that it requires very carefully selected and fairly minimal input in order to create a desired output, because in order for structural traits to be inferred, there cannot be too much difference between the pieces shown. Similarly, the program would become considerably larger in memory and slower as data it has to analyse increases given the vast amount of types of information that are checked, from melodic string-matching algorithms to structural analysis and harmonic analysis. Also, Cope's system does rely heavily on pre-composed music, as every motif can trace its ancestry to a previous incarnation in the supplied data, which certainly depends heavily on the notion of creativity being nothing more than the recycling and re-contextualisation of previous ideas. However, if a very specific type of piece such as Chopin waltz, a Bach chorale, or a Beethovenian Sonata is desired, EMI would certainly be the best piece of software to use.

3.4.2 Neural Networks

Neural Networks are machine learning algorithms which simulate the neural process of the human brain. Biologically, the brain is a series of interconnected neurons. A neuron consists of a cell body called a dendrite, and an axon which carries a signal to the dendrite. When a neuron is fired, it releases a substance via a synapse, which sparks an electrical impulse in a neighbouring neuron's axon, thus increasing the electric potential of this neighbouring neuron's dendrite. Each neuron has an electrical threshold, which when met, causes the neuron to fire. As such, a vast network of these interconnected neurons create a biological neural network.

Artificial neural networks function in much the same way, however, groups of neurons are abstracted into levels, where each level is processed consecutively. Initially, a dataset will enter the network and cause certain neurons in the first layer to fire. This sets off the chain reaction, with the firing of neurons propagating through the layers, both forwards and backwards depending on the structure of the network. Ultimately, the output layer will be reached, and depending on the individual neurons that are still firing, a certain output will be produced. As such, to the user, only the input layer and output layer are visible, the inner layers and inner workings of the neural network are unknown.

With the architecture in place, a neural network can be trained on a dataset in order to change the weightings between and thresholds within neurons. As such, any future input that is similar to the data the network was trained on will have its input and output compared with the relationships between input and output of the training data. Through the training, the network has learned patterns and stylistic traits, which are then weighted more heavily, and in turn influence the output of new data flushed through the network.

As far as algorithmic composition is concerned, artificial neural networks have been used both harmonically and melodically. Hermann Hild, Johannes Feulner and Wolfram Menzel created the program HARMONET [12], which is capable of harmonising melodies in the style of J.S. Bach. The system combines both rule based decisions as well as neural networks which were trained on Bach

chorales. Multiple networks were used, one for harmonic analysis and generation, which considered the current and previous three harmonic and melodic states much like a Markov model would and generates a future state. Until the very last step, the creation is homorhythmic, with each note representing a crotchet chord, however at this step, another network trained on the same data set but focussing on ornamentation is used, embellishing each voice. An example of the progression of the HARMONET algorithm is seen in Figure 3.12.

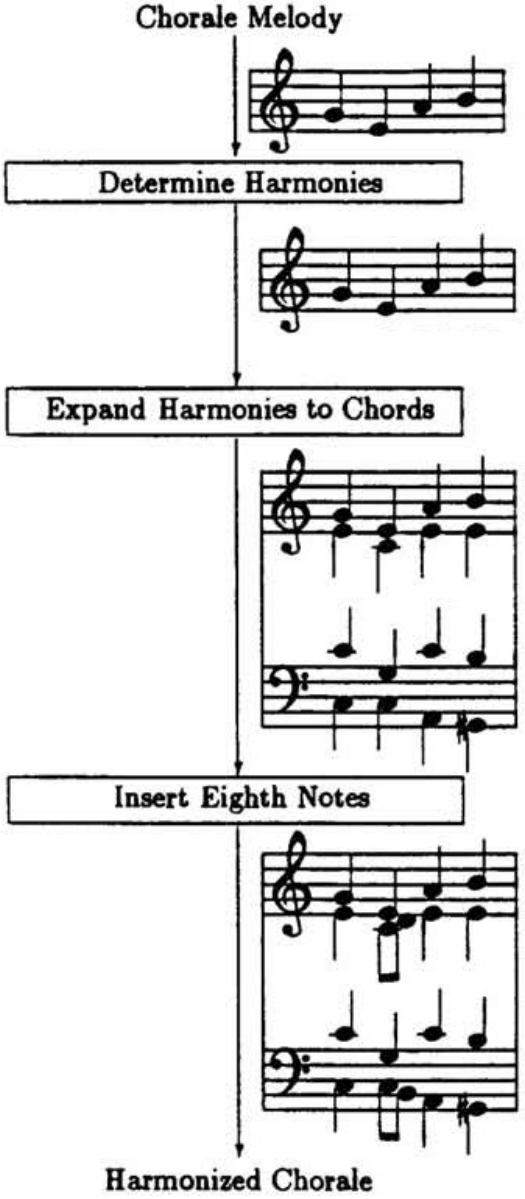


Figure 3.12: An illustration of the multiple steps in the generation of a harmonisation by HARMONET taken from [12]

Recurrent neural networks have been used by Peter Todd in the creation of complete melody lines as seen in Figure 3.13. In his neural network, there are three types of neuron. Plan units represent the current point in the melody, the context units maintain the state of the melody so far, and the output units generate new notes. Notes are stored within each of the units and contain a tone pitch and rhythmic length. The network is set up such that a time slice of a basic metric unit

continually increases, and the network loops through the different layers recursively creating one note per time-slice. If a pitch is shared with its immediate predecessor, they can be combined to create a larger metric unit. After a specified amount of time, the result will be output [32].

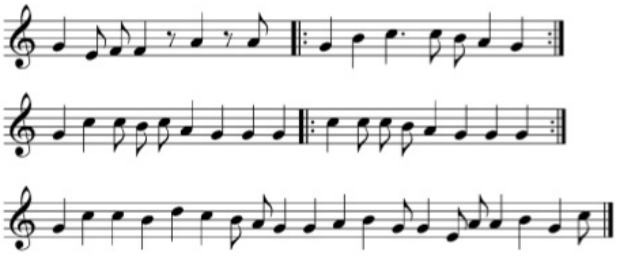


Figure 3.13: An example of Peter Todd's neural network made melodies from [32]

Nierhaus states in reviewing Neural Networks relative to Markov processes that:

> "A great difference from a Markov process lies in the treatment of the context of a note value, be that related to the tone pitch or duration. In a Markov models, the probability of the production of a certain note value depends on transition probabilities of note sequences in a corpus. The context depth given by the order of the Markov process, however, follows a sequential consideration of past events. So, the statistical prediction of a note value in a Markov process of $n - order$ is only possible if the already produced n note values also occur in the same order in the underlying corpus. An ANN, however, is not bound to this strictly sequential view. Even though the consideration of a particular although generally not very large context is possible, note values may still be generated that do not occur in the same sequential order in the corpus" [22].

Overall, artificial neural networks are very competent when it comes to harmonic analysis and melodic creation. In fact, as Nierhaus stated, the results of a neural network can often be surprisingly original, since unlike Markov models, they can create new melodic material that hasn't necessarily appeared previously in the training data. However, in the creation of melodic and harmonic phrases, there is a vast computational overhead when using neural networks, and some networks even require the use of a GPU. Neural Networks also lack a general notion of hierarchical structure as seen in Generative Grammars, and as such, larger outputs become stationary and repetitive. However, as far as creation of short melodic motifs, a neural network is more than capable, and in conjunction with augmented transition networks could prove a very powerful backbone to an algorithmic composer.

3.5 Conclusion

This is by no means an exhaustive list of techniques that can be employed in the quest for algorithmic composition. For example, genetic algorithms have had a proven application in melodic and harmonic creation. Genetic algorithms are theoretically underpinned by Charles Darwin's theory of evolution, with the core principles of the theory, including "survival of the fittest" and "mutation", being central to the algorithms. The program serves as the living space of a population, and the individuals have certain characteristics that can be examined through a fitness function, which may be rule based, mathematical or a comparison to an ideal. Individuals are initialised, and then they begin to randomly crossover and mutate, creating new individuals which inherit characteristics of their parents as well as mutated characteristics. Each new individual can be tested against the fitness function and through the principle of survival of the fittest, the strongest individuals will then be more heavily weighted.

As far as this can be applied to an algorithmic composer, one implementation by Andrew Horner and David Goldberg has proven successful. Starting with an initial melodic phrase and a reference pattern, the melodic phrase is a randomly mutated from a set predefined mutations algorithms, including deletion of a note, swapping notes and transposition of notes. After a predetermined number of mutations, a fitness function through comparison to the reference pattern can be applied, and the resulting pattern be adopted in the composition if deemed fit enough [13].

Despite the capability of genetic algorithms and other methods, I chose to focus on Markov processes and Generative Grammars in my approach to developing an algorithmic composer. It appears that each method has certain drawbacks, and certain desirable features. Markov processes appear to be well suited to the generation of motivic phrases for an individual voice, however, when it comes to creating harmonically correct polyphonic music with an interesting texture, Markov processes have not been successful. Given their ease of use, ubiquity in the realm of algorithmic composition, and simplicity, I wanted to come up with a new approach to make Markovian analysis and generation more powerful and appropriate to the generation of complex polyphonic music in a context-free style.

Further, Generative Grammars have been proven useful in the analysis of style and harmonisation of pieces. However, using context-free grammars, I wanted to explore their capability as the foundation of a complete compositional process, that is to say solely using Generative Grammars to create an entire piece of music in a context-free, unparameterised manner.

As such, I chose to develop a program that can convincingly analyse a set of works, from which a grammar may be inferred or dictated, from which many Markovian models representing the different dimensions of music can be generated, including harmony, and which is capable of creating a novel piece of music. That is to say that the user could generate a piece whose form represents a Bach chorale through the data set analysing a set of Bach chorales, inferring the structure and

grammar, and analysing the piece melodically to generate a piece in the same style. I will use MusicXML as the main data type given the availability of works online and the overall complexity and appropriateness of the information it carries.

Chapter 4

Methodology: Overview

In this chapter I shall describe the high-level design of the system, including the algorithms I shall use, as well as the general theory behind the composition program.

4.1 Design of the System

The overall system for any creative content generator consists of two phases, analysis, and generation. In this project the analysis consists of two steps: creating Markovian transition matrices and extracting musical sentences that appropriately represent the input music, and using this corpus of sentences to infer a grammar and alphabet. With this analysis in place, the grammatical rules inferred and data collected can be used to create new pieces of music.

The overall aim for this project is to develop an entirely context-free composition program. As mentioned previously, music is a multi-dimensional, complex art form. As such, the many facets of the input files must be analysed such that the output can appropriately reconstruct the many facets of a piece of music. However, superficially, the system must be built in such a way that any type of notated score can be read, regardless of the number of parts, key signature, modality or combination of voices.

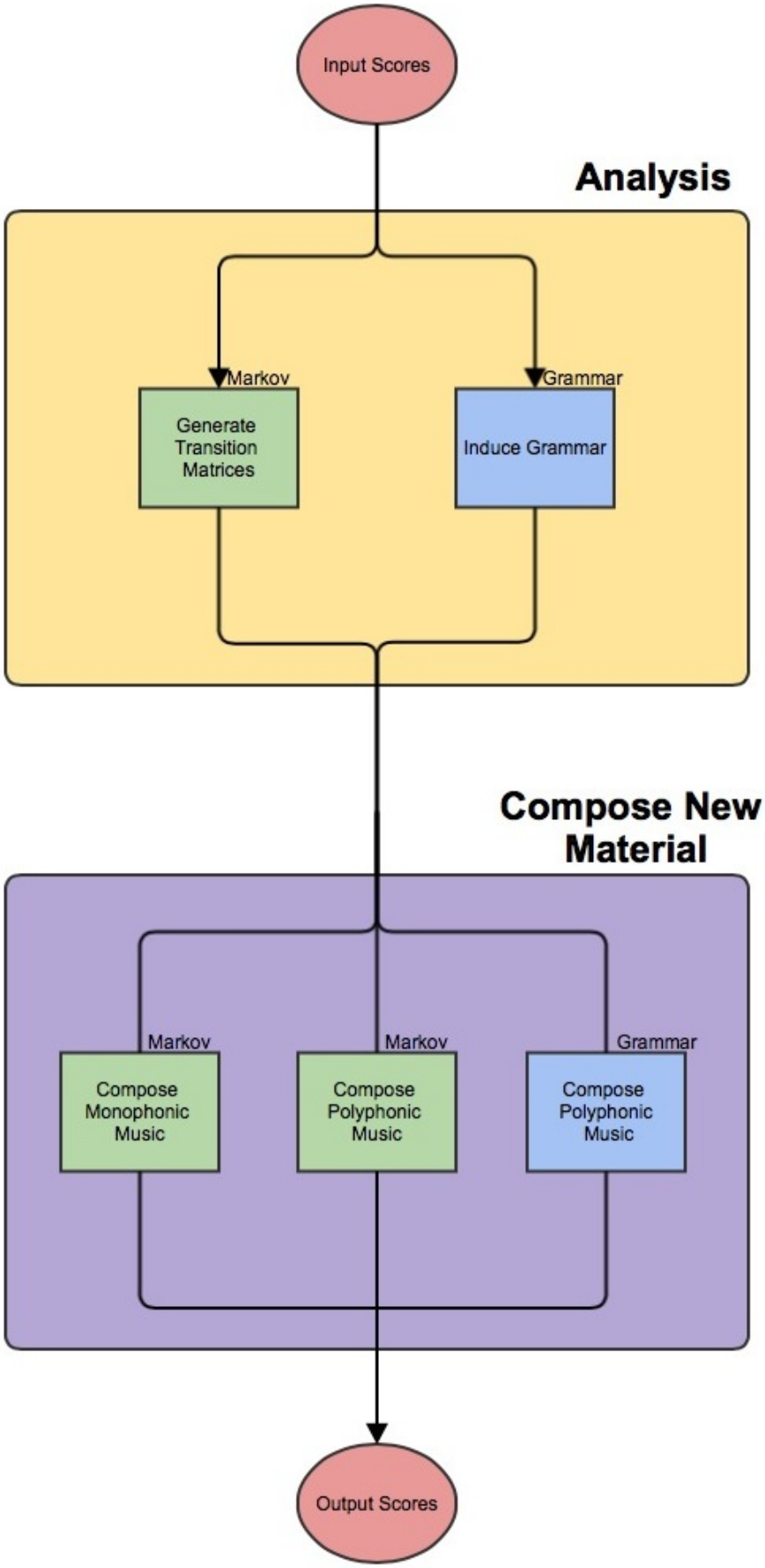


Figure 4.1: Design of the System

4.2 Markovian Analysis and Composition

The first manner in which the input scores will be analysed is Markovian analysis. As discussed in chapter 3, Markovian analysis parses through data, deriving a transition matrix representing the probability of any future state occurring based on the current state. This is basic Markovian system can be expanded to any $n - order$, where n represents the number of states preceding and including the current state.

In order to analyse the data in such a way that new and convincing music might be produced in an entirely context-free manner, the appropriate data members must be analysed. There are two main dimensions to music writing central to any piece of music regardless of style, melody and harmony. Melody concerns itself with an individual voice moving across time. A single voice consists of a series of notes, each of which consists of a pitch and rhythmic value. As such, these are the most fundamental data members by which each individual voice shall be analysed. Note that it would be possible to combine the two, and analyse a pitch-rhythm tuple, however, for the sake of diluting music into its most fundamental parts, I have chosen not to. With separate transition matrices for the pitches and rhythms per each voice type, the range of a voice type, as well as its rhythmic and harmonic characteristics will be maintained.

Harmony, on the other hand, consists of the interaction of multiple voices' pitches at any given time-slice. As such, for the harmonic analysis, each atomic unit will be a four-part chord. As per the most common style of composition, a melody line is composed and then a harmonic framework is established around the melody. As such, mapping harmonic rhythm will not be necessary, as at the stage of composition, rhythm will already be established within the melody.

For each type of matrix, each start state must also be mapped so that the starting rhythm, pitch and chord can also be generated probabilistically.

With the system in place thus far, an algorithmic composer could theoretically generate infinite music that would start by randomly selecting a start state, then utilising randomisation to select each next state from the transition matrix based on the previous n states, until the composer is made to stop. One issue that will arise, however, is that when the composer is made to stop, the terminal state will be random. Musical phrases and pieces, however, end in typically on the root chord, and the root is reached through certain chord progressions known as cadences. Therefore, in order to make sure that the composer starts and ends in the correct place, my system will simply carry out the normal Markovian analysis forwards, from beginning to end, as well as from end to beginning. With both forwards and backwards transition matrices, as well as a set of start and end states, the generation can start, continue for as long as the user of the system sees fit, compose the last X measures backwards in the same fashion, however using the reverse matrices, and then one common state that unifies both forwards and backwards composition can be found. As such, this novel approach to Markovian analysis and composition will allow for complete phrase construction,

which abides by the rules of the music theory of the input pieces, as well as the structure of phrases.

4.3 Grammatical Analysis, Composition and the ADIOS Algorithm

The algorithm that I shall use to derive a grammatical structure of the music is a context-free grammatical generator called the ADIOS Algorithm. The ADIOS Algorithm [30], which stands for Automatic DIstillation Of Structure, was developed by Zach Solan in 2005. It uses a statistical model in order to derive structure in an unsupervised manner.

The algorithm loads the entire set of sentences into a graph, in which each word is represented by a node, and each sentence is represented by a path through the nodes' edges in the appropriate order. There are also specially privileged nodes at the origin and terminus of each path, and these are known as the *begin node* and *end node*. Importantly, multiple edges are allowed between two nodes. At first, the number of paths in the graph is equal to the number of sentences provided. With this tree in place, the algorithm will look for hierarchical traits, segmenting the sentences and generalising patterns.

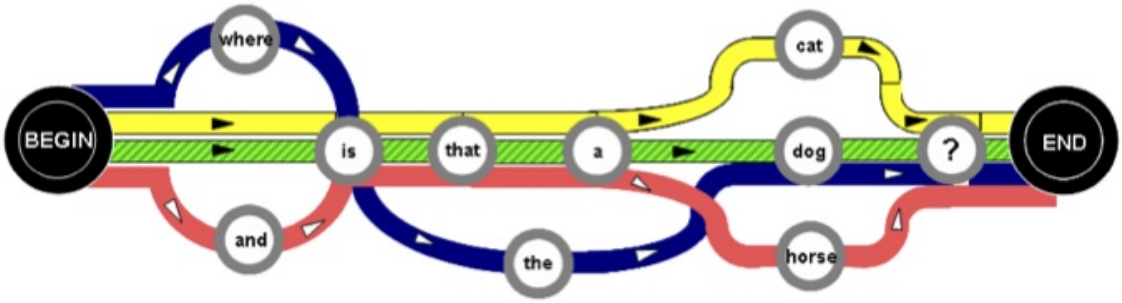


Figure 4.2: Example of the preliminary ADIOS graph structure consisting of three sentences taken from [30]

The segmentation process uses a subroutine called MEX by Solan, which stands for motif extraction. The process is statistical, and since multiple edges are allowed between two nodes, any location where there is a significant number of paths that traverse between two nodes implies a significant relationship or pattern. A new node P is created as an abstraction of the discovered pattern, and every path that once went through the pattern is rewired to go through P. As such, the graph is distilled until no new patterns can be found.

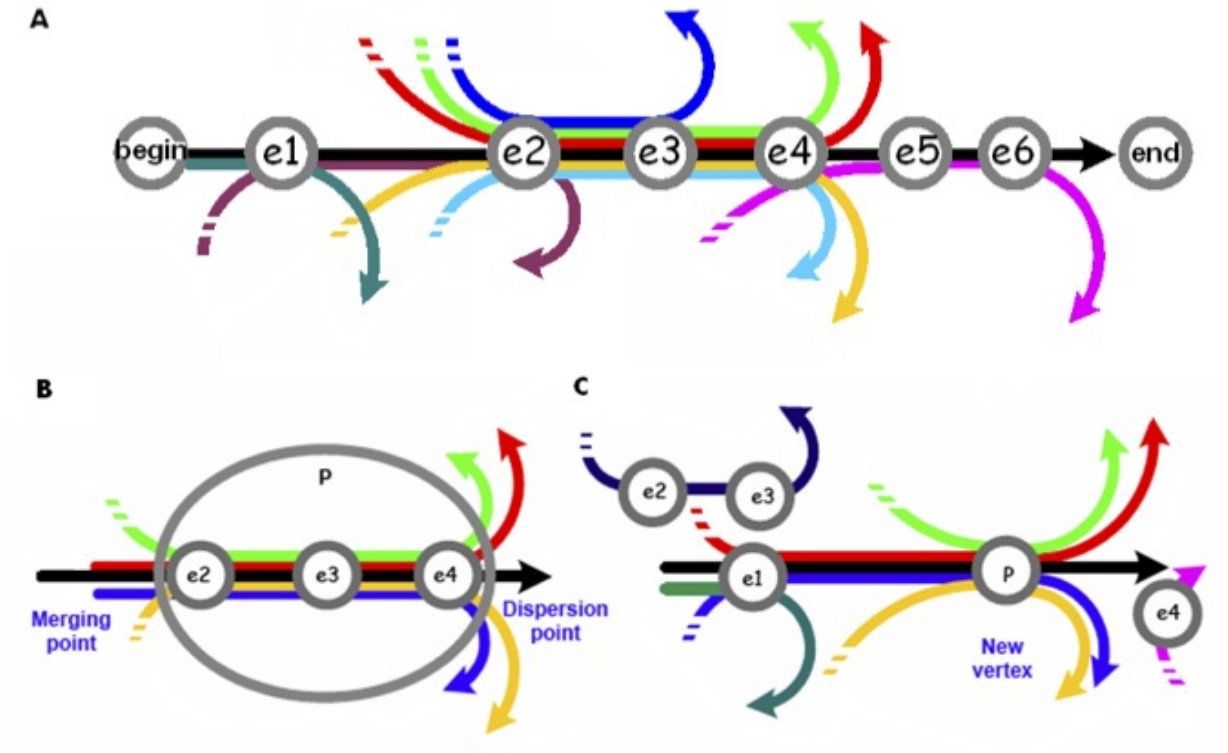


Figure 4.3: Example of the ADIOS segmentation process taken from [30]

The generalisation process also occurs during the MEX procedure of segmentation, but where segmentation looked for statistically significant bundles of edges, generalisation looks for similarly distributed words. This is carried out by traversing each path with a context window of a fixed size, looking for other paths that coincide with the window in all places except in one place. An *Equivalence Class* replaces this location. This is carried out for every location within the window and on every path. The segmentation process continues and thus Equivalence Classes too can be incorporated into the segmentation process.

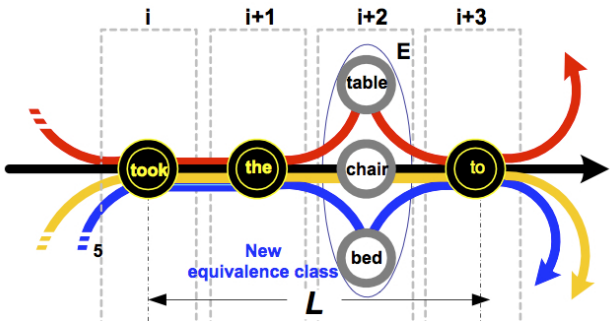


Figure 4.4: Example of the ADIOS generalisation process taken from [30]

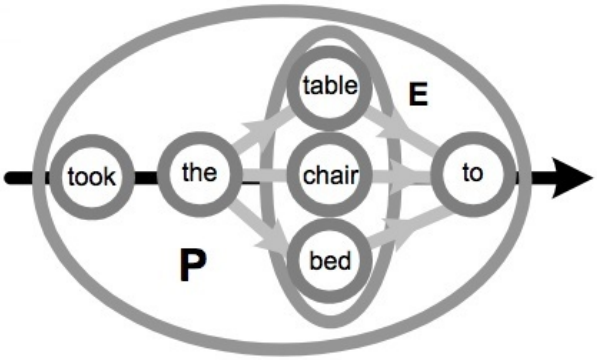


Figure 4.5: Example of the ADIOS segmentation process incorporating an Equivalence Class taken from [30]

Once all paths have been traversed and all equivalence classes extracted, the final graph represents the distilled grammar. New sentences can be generated by traversing the graph from any start node to any end node, and substituting any Equivalence Class or segmented path with a specific member word or words.

Therefore, to apply this to music, a system of encoding the data into words such that the generated words will be decipherable and create new music in the style of the composer is necessary. As such, words representing each harmonic time-slice will distill certain permitted transitions, as well as which four-part movements can lead to which other four-part movements, and the resulting output will represent all four voices which can then be extracted and each note assigned accordingly. Since the chords also contain the melodic information and transitions per voice, separate grammars aren't required for the generation of each voice melodically. As we want to define a grammar for how voices move relative to one another, if we included any other information, such as a rhythmic signifier, the grammatical parser would treat the word representing a C Major triad as a quaver and the word representing a C Major triad as a minim as different. As such, when it comes to composition, rhythm will simply have to be defined as one unit per harmonic word.

Chapter 5

Methodology: Implementation

The Algorithmic Composer is fully written in C++. The program is designed to take in any number of MusicXML files and to analyse them in such a way that no defined music theory or parameterisation is necessary, resulting in a context-free Markovian composer. The new approach to analysis incorporates the creation both forwards and backwards Markov matrices for rhythm, pitch and harmony, and unifying them in order to create a new harmonically accurate piece in the style of the input composer. Similarly, the Algorithmic Composer also uses context-free grammar distillation to create new pieces based on a grammar derived by the ADIOS algorithm.

5.1 Data Reading

As an input, the system is designed to take in MusicXML files. Using the boost filesystem library [9], the system is capable of taking in a directory, and all appropriate files will be analysed. Once an individual file is loaded into memory, the XML structure, which itself is a tree system, is transferred into a traversable tree in memory using the libmusicXML2 package [24], a C++ library developed by Recordare Inc.

Now that the XML tree is stored in memory, preliminary information can be gathered regarding the score. Firstly, the key signature is established, including whether or not the piece is in a major or minor mode, the number of parts and each of their names, and how many sharps/flats the key signature contains. Within MusicXML, a negative integer indicates that many flats within the key signature, and a positive integer indicates that many sharps. In order to compare each piece, each score must be transposed to a common key if necessary. For sake of simplicity, I chose a key signature of 0, which corresponds to C Major/A minor. As such, the tree is transposed using libmusicXMLs transposition method. The tree is now ready to be passed onto the **Score** class.

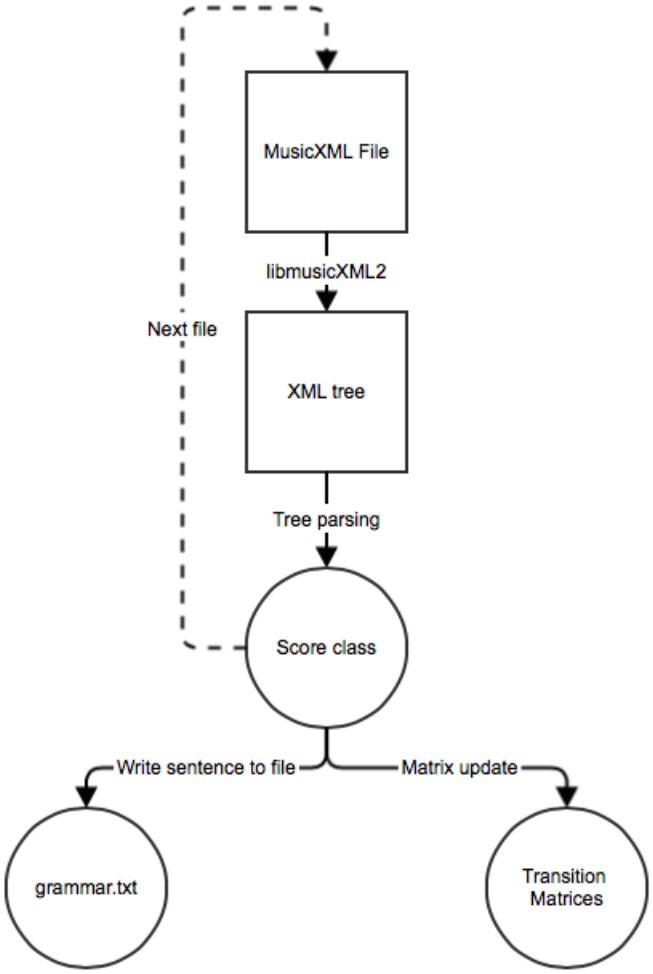


Figure 5.1: The Analytical Process

5.2 Score Data Structure

The **Score** class is initialised with the XML tree, as well as the basic information ascertained such as key signature and modality. From parsing the headers of the file, the **Score** initialises the list of part names, the key and the time signature.

Continuing through the iteration of the tree, the order in which parts are met matches the order of the names stored. In MusicXML, each part is written in its entirety, thus the format is partwise. When a part is met, a new **Part** object is created, and the current iterator in the tree is passed on for the **Part** to extract the pertinent information.

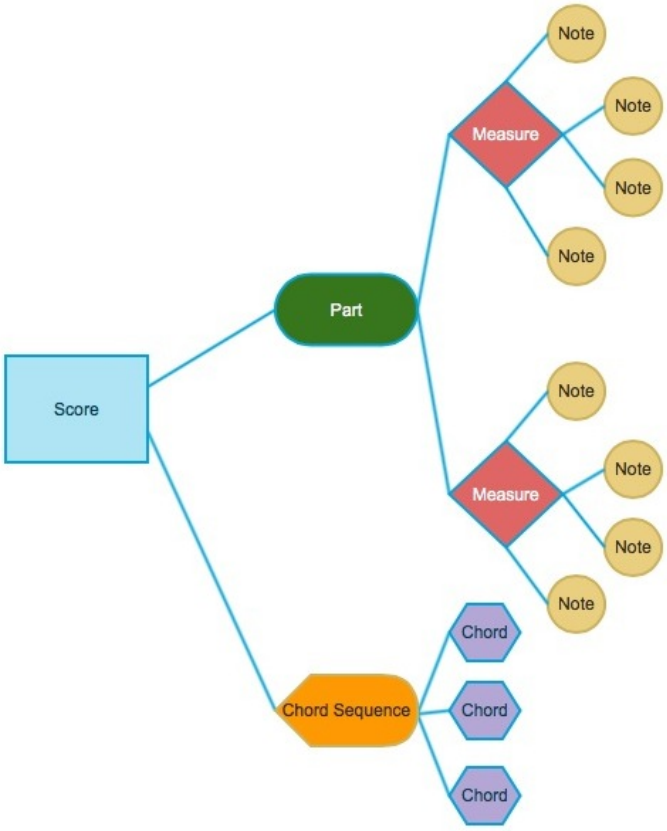


Figure 5.2: The Score Data Structure

Just as the iterator is passed on in the creation of a new **Part**, so too is the iterator passed on in within the **Part** in the creation of a **Measure**, and within the **Measure** it is passed on on the creation of a **Note**. Visualised as a hierarchical tree, the **Note** is a leaf, and the **Score** is the root, with **Part** and **Measure** in between. As you descend this tree, it is a one to many relationship, in that one **Score** consists of many **Parts**, each of which consists of many **Measures**, each of which consists of many **Notes**. At the level of the **Note**, the information that is extracted consists of:

1. Whether the note is a rest or not
2. Pitch
3. Octave
4. Alteration (sharp or flat)
5. Value

By counting through the bar and keeping track of the time signature, the **Note** also stores its relative location in the bar, as well as a pointer to the previous and next **Notes**.

Once each **Part** is fully incorporated into the data structure, the harmonic skeleton of the piece must also be established. In order to so, the **Chord_Sequence** class is initialised. The chord

sequence is established by creating a series of **Chords**. Each **Chords** is established by iterating through each **Part** simultaneously, and at the moment of any melodic change in any **Part**, the note values from each **Part** is recorded, as well as the duration that these four notes simultaneously appear. Note that if in a two part piece, part 1 has a C for a duration of 0.5 and part 2 has a E for a duration of 0.25 followed by a G for a duration of 0.25, the final result would be two chords $< C, E >$ and $< C, G >$ each with a value of 0.25.

Now that the **Score** represents each Part and the chord sequence, the Score structure can be parsed for data extraction.

5.3 Markovian Analysis

As discussed, music is a multi-dimensional entity, consisting of intertwined voices, each of which is comprised of a series of a combination of pitches and rhythms. There are multiple ways in which the score could be analysed, as certain parameters could be coupled in order to analyse units as a whole. For example, an entire note, including pitch and rhythmic duration, could be treated atomically in a Markov map. However, I chose to separate out all of the fundamental elements of music construction and map each one individually.

The **Matrix_Master** class contains all the transition matrices that are created throughout the analysis. For rhythm and pitch, there is a **Matrix** per voice type, Soprano, Alto, Tenor or Bass, and at the time of updating the **Matrix**, the voice is checked against a list of names stored in **All_Names.h** and assigned to the appropriate **Matrix**. Similarly, there is both a major and minor mode **Matrix** for pitch per voice, and as such there are eight pitch matrices, one major and one minor for each voice type. There are four rhythmic matrices, one for each voice type. Finally, there are two harmonic matrices, one for both major and minor mode. A **Score** is passed into the **Matrix_Master**, from which the **Matrix_Master** updates each of it's matrices appropriately based on modality and part name.

Each matrix derives from the abstract **Matrix** class. The subclasses all accept a **Part** pointer to update the matrix, and each subclass analyses different aspects of that **Part**. Each subclass contains:

1. A list of start states
2. A forward map in which the key is the n-gram, and the value stored is a list of possible transitions, including duplicates.
3. A list of end states

4. A backward map in which the key is the n-gram, and the value stored is a list of possible transitions, including duplicates.

Firstly, the pitches are analysed forwards and backwards in the **Pitch_Matrix** class. Based on the $n - order$ specified at the start of the program, the transitions between pitches are stored in a map. In this case, a pitch is stored as a combination of a pitch and an octave. It would be possible to store just pitch transitions, regardless of octave, however, this would treat an octave jump equally to a repeated note, which is a different transition. Therefore, each precise frequency is treated individually.

Secondly, the rhythmic transitions are stored in the **Rhythm_Matrix** class. Values are stored by the name of their value following the American notation system (half note, quarter note, eighth note etc.), because this is how they are presented in the MusicXML format. It would also be possible to store the rhythmic values as a numerical value.

Finally, the harmonic transitions are stored in the **Chord_Matrix** class. Each chord is stored as a list of pitch values, ordered by the voice they represent, *e.g.* $< Soprano, Alto, Tenor, Bass >$.

In order to avoid any problems with sparse data, the transition matrices also contain all appropriate transitions for n-grams where $0 < n < n - order$. Whilst this takes up more memory, it also allows for greater chance of finding a next state, particularly when it comes to harmonisation, as if the full n-gram cannot be found, the size can be continually reduced until a transition value is found within the **Matrix**.

5.4 Grammatical Inference

After each score is analysed, a sentence representing the harmonic structure of the piece is appended to a temporary file called grammar.txt. The sentence is preceded by a $*$ and terminated with a #. A sentence consists of words, which in this case represents a chord. The structure of a word includes the note values, as well as the duration, as follows:

$$< pitch_1, pitch_2..., pitch_n >$$

Once all the scores have been read through and every sentence deposited, the grammar can be distilled using the ADIOS Algorithm. I have used a C++ implementation of the ADIOS algorithm found on Github [28]. I created the class **Grammar_Parser**, which serves as an interface between the package and my program.

5.5 Score Creation

Now that all analysis is complete, there are multiple matrices and a distilled grammar, this data combined with randomisation can be utilised to create new compositions. There are three options offered in this program. Melodic Markov composition, Generative Grammar composition, and Melody First Markov composition. The **Score_Creator** class is responsible for all music generation.

5.5.1 Melodic Markov Composition

Melodic composition represents the most base form of composition where a single **Part** created only concerns itself with its own **Part_Matrix** and **Rhythm_Matrix**. The result is a monophonic composition.

Using the same **Score** class as before, a new instance is created. The user specifies a length and modality, and four parts are created. Each voice is created utilising randomisation and the Markov matrices. When a part is composed via the **create_blind_part** method, its name is checked such that the correct matrices will be used. From then on, measures are created in a forwards direction, and notes are added to them until they are full, at which point it is add to the part. The composer knows the n-order in which the analysis took place and when looking for the next note or rhythmic value, uses an n-gram of that order when querying the matrix. When the query takes place, if the n-gram does not appear in the matrix, the first value will be removed from the front and the n-gram now of length $n-1$ is queried until $n=0$ when a random start state will again be selected. When the key is found in the matrix, one of the possible transitions is randomly selected and returned. Pseudocode is presented in **Algorithms 1 and 2**.

This process continues until the last two measures, at which point, the same process takes place, except in the reverse using reverse matrices. This ensures that each part ends in a cadential manner, without having to teach the program how to cadence or what a cadence is.

Once the **Part** is created, it is added to the **Score**, which is sent to the **XML_Creator**.

Algorithm 1 Create Blind Part

```
1:  procedure PART MAKER
2:      p ← new Part
3:      while i < number of measures do
4:          m ← new Measure
5:          time left ← value from time signature
6:          while time left > 0 do
7:              n ← new Note
8:              repeat
9:                  n.rhythm ← getFromMatrix(n-gram)
10:             until n.rhythm <= time left
11:             n-gram ← n
12:             time left ← time left + n.rhythm
13:             m ← n
14:         p ← m
15:     Score ← p
```

Algorithm 2 Get Value From a Matrix

```
1: procedure GET FROM MATRIX(n-gram)
2: top:
3:     if n-gram is empty then return startValues[random]
4: loop:
5:     while n-gram is not empty do
6:         if n-gram exists in Matrix then return Matrix[n-gram][random]
7:         Remove value from front of n-gram
8:     goto top.
```

5.5.2 Generative Grammar Composition

The second way in which to compose is using the distilled grammar as opposed to using the Markov matrices. A feature of the ADIOS algorithm implementation used[28] is that a built in function can be used to create a new sentence by simply traversing the graph from any *start node* to *end node*. Therefore, the first step in creating a new piece with generative grammar is to simply request a new sentence from the grammar tree.

With the new sentence, not only is the melody and harmony in place, but we also assume each word to represent one rhythmic unit. As such, in this implementation we treat one word to represent one crotchet. In order to convert the sentence into a **Score**, each word must be extracted, and from each word, the list of pitch values. The pitch values were originally provided in part order, that is to say that they appear in the order $< Soprano, Alto, Tenor, Bass >$. As such, for each part, the appropriate note per word is simply found at the index of it's own name value as defined in *All_Parts.h*. Thus if a Part is a Soprano, it will extract the note at index 0 of each word.

With the sentence now split appropriately into full measures, the parts simply read through the sentence, extracting the appropriate note value. Since the sentences that the grammar was derived from split up longer note values if another part changes during the note (e.g. if in a two part piece, part 1 has a C for a duration of 0.5 and part 2 has a E for a duration of 0.25 followed by a G for a duration of 0.25, the final result would be two chords $< C, E >$ and $< C, G >$ each with a value of 0.25), a part checks the previous note it has added, and if they have the same pitch value, the rhythm of the previous note is simply updated, provided the value will result in a legitimate note value.

Again, once all parts are created, they are added to the **Score**, which is sent to the **XML_Creator**.

5.5.3 Melody First Markov Composition

The final method of composition is again uses to the Markov matrices, but unlike *Melodic Markov Composition*, the chord matrix is also employed in order to create complimentary voices, and thus a consonant harmonic sounding piece. The first step is to create a blind **Part** as demonstrated in *Melodic Markov Composition*. The default part to create is the Soprano, as typically this is the melodic voice, but the process would work with any voice being created first.

With this part in place, a harmonic skeleton is now found using the **Algorithm 2**, however, on the **Chord_Matrix**. To ensure that an appropriate chord is found that matches the melody, the chord returned must have the correct note in the correct location. There is an arbitrary number of attempts (set to 100 in this implementation), and if no appropriate chord has been found, the harmony is set to all rests apart from the melodic note. Note that because harmony is continuous, if a rest is applied because the Matrix cannot produce an appropriate chord value, the n-gram is not updated with the rest value. Rather, the rest is treated as a passing tone, so that the harmony will be continuous through each part.

The harmonisation occurs both forwards and backwards at the same time, and thus the bidirectional harmonisation meets towards the centre of the score. There is one transition that must be filled in, unifying both forwards and backwards directions. This is found by simply querying both forwards and backwards matrices with the appropriate n-grams, and looking for a common transition possibility.

Once harmony has been established, the voices are simply filled in in the same manner as the *Generative Grammar Composition* process, including updating the rhythmic value of any repeated pitches in a voice.

Again, once all parts are created, they are added to the **Score**, which is sent to the **XML_Creator**.

5.6 XML Creation

Now that the **Score** has been created, a MusicXML output file must be created such that the user of the program can open the composition in a score editor and either listen to the MIDI version or edit as they see fit. Using the libmusicXML2 package once again, there is an **XMLFactory**, which facilitates the process. The **XML_Creator** class serves as a wrapper for the supplied **XML-Factory**. In order to translate the score to MusicXML format, a new partwise MusicXML tree is created, and each part is added to the tree by simply parsing through the Score in a partwise manner and using the appropriate **factoryAddPart()**, **factoryMeasure()** and **factoryNote()** functions. The MusicXML file is written and saved to the directory specified at the outset of the program.

5.7 The Final Product/Installation

The final product is a command line tool called *Algorithmic Composer*. The source files can be downloaded from https://github.com/nicholasgubbins along with sample MusicXML files, as well as the makefile for installation. While the libmusicXML2 and ADIOS algorithm packages are included, the software requires the boost libraries [9] to be installed and the appropriate headers linked to the compiler. The program should be executed by providing both an input directory from which the XML files can be read, and an output directory into which the newly composed XML files can be written:

$$./AlgorithmicComposer < inputdirectory >< outputdirectory >$$

When the program is executed, the user will be prompted to provide an Markovian order by which the analysis will be performed. On completion of analysis and grammar distillation, **Score_Creator** is automatically called, and the user can choose to make as many compositions as desired in any of the three styles described above. For *Melodic Markov Composition* and *Melody First Markov Composition*, the user can choose the length of the piece as well as the modality. For *Melodic Markov Composition*, the user can also choose which voice should be composed. The score will be created, converted in MusicXML format and saved to the chosen output directory. The user also gets to input the composition name as well as the user's name which will appear under the Composer credit of the score. Once created, a rough harmonic analysis in terms of the Western Classical Tradition will appear on the command line.

In order to view and listen to the MusicXML file, score writing software is required. There are multiple free pieces of software that can read MusicXML files including Finale NotePad and Open-

Muse. This software can present the score as interactively as well as play back a MIDI version.

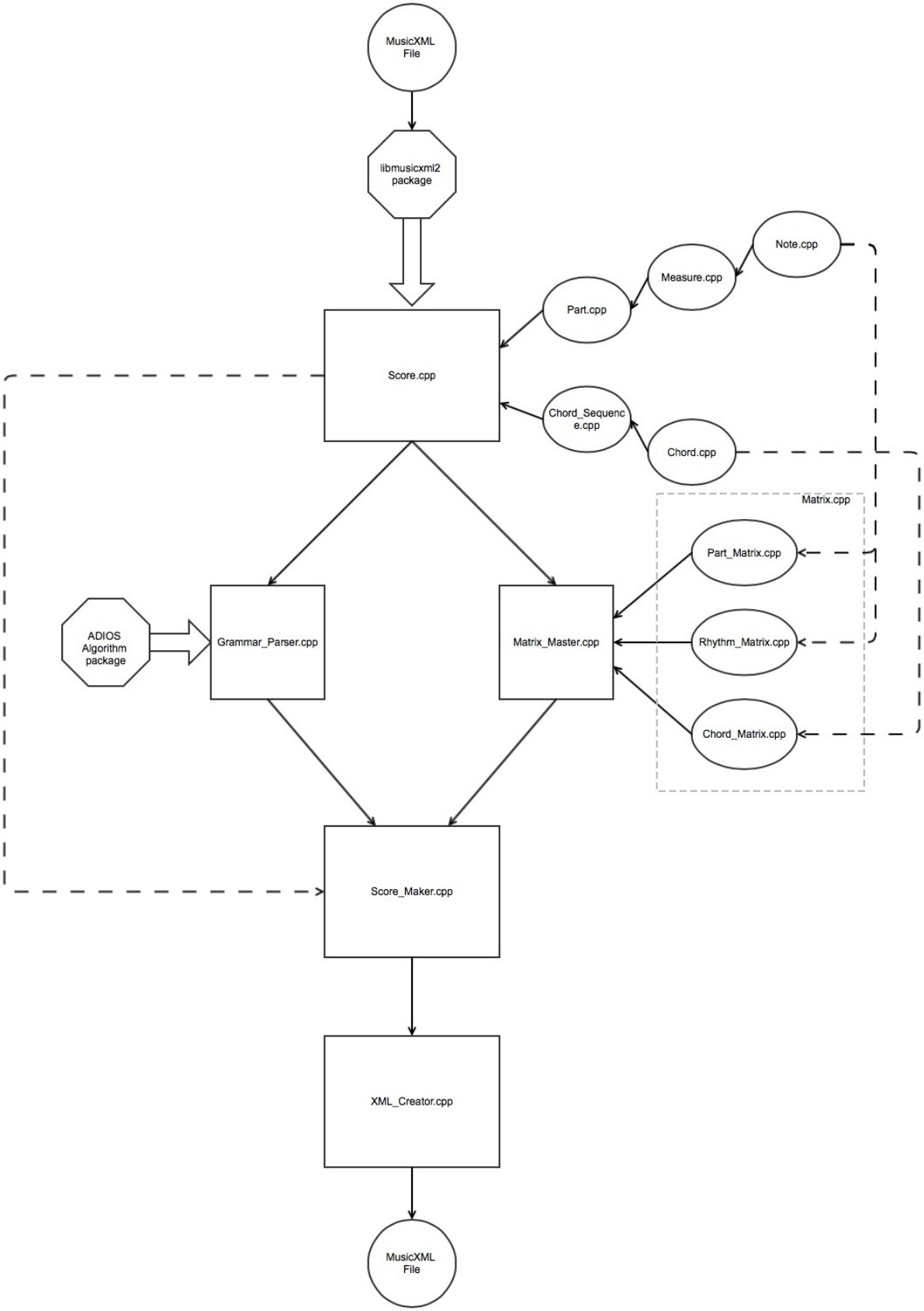


Figure 5.3: All the classes and their relations with one another in the Algorithmic Composer program

Chapter 6

Results and Discussion

The program works with varying degrees of success due to the effect of randomisation, but on the whole can successfully compose convincing pieces of music. I shall discuss the merits of each type of composition's output, as well as an analysis of the example output to demonstrate the ability of the composition. All compositions made by the program were trained on 250 Bach Chorales.

6.1 Melodic Markov Composition

Confirming what many others have executed before [5], the Melodic Markov Compositions are highly successful, as seen in **Appendix A.1**. Since each part type is mapped in its own set of matrices, the output of using a part's matrices will lead to style imitation for a part type. That is to say that the contour and range of each part's melodic line is very true to the input files. For example, looking at A.4, the Bass ends on a strong 1-5-1 which is very characteristic of a Bass as the voice's function tends to be anchoring the harmony, particularly at cadences. Similarly, this melody also implies a typical Bach harmony, particularly with the presence of an F# at bar 3, signalling a modulation to the dominant.

Comparing a Bach Chorale melody seen in Figure 6.1 with one of my own at Figure 6.2, even an untrained musician will be able to pick out similar features. Looking at the progression of melody, the intervals between each note are largely conjunct, that is to say they move stepwise up and down. Rhythmically, the note lengths are regular, using a combination of crotchets, minims, and occasionally quavers. When quavers do appear in both melodies, they come in pairs and are typically scaler. Another shared feature is the presence of natural breaks in the melody line. Whilst the Bach Chorale has pauses written into the score, they can be inferred within my own Chorale melody by the minims. Finally, there is a certain roundness to my own melody, thanks to my novel reverse Markovian composition. The melody clearly starts and ends in A Minor, the tonic of the melody. However, the overall shape of the melody is that of an arch. The melody starts on the fifth scale degree, moves upwards conjunctly until the middle of the fifth bar, at which point it climaxes

and resolves downwards until it finally rests on the tonic.

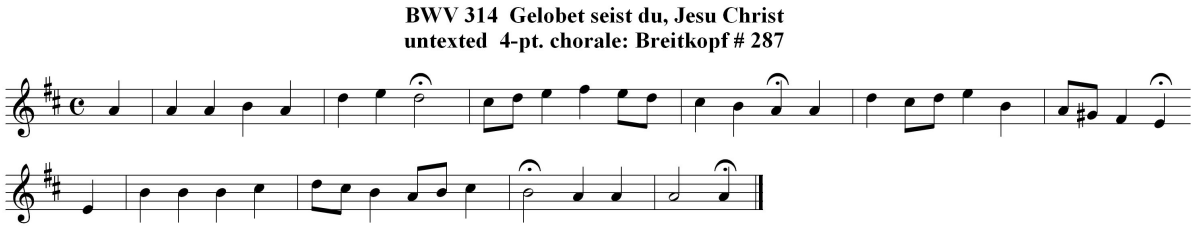


Figure 6.1: Bach chorale melody *Gelobet seist du, Jesu Christ* BWV 314

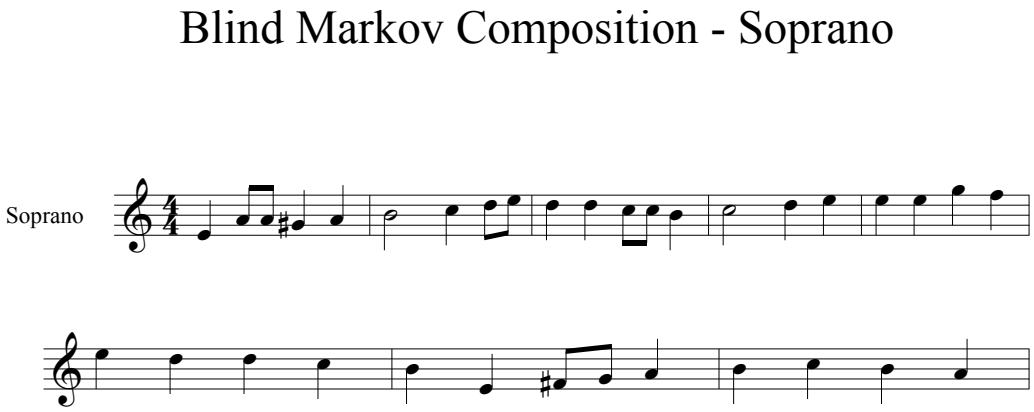


Figure 6.2: Markovian generated Soprano melody in A minor with order 4

Although not a feature in the final product, I decided to make a four part chorale consisting entirely of blind Markov melodies per part (see **Appendix A.5**). In this blind Markov Composition is that each part individually tends to be a convincing in its own right. Unsurprisingly, the blind Markov Composition generally produces a harmonically incorrect piece of music, due to each part being created individually without any influence from other parts. That is to say that each part is composed regardless of the rhythm, melody or implied harmony of other parts, and as such the output is polyphonic and dissonant. However, one surprising feature, and this is thanks to the bidirectional analysis and composition process, is that the pieces tend to start and end more consonantly. If you consider the rules of music theory, a piece tends to start on the tonic, or root, of the key. Therefore, it is unsurprising that the start states for all parts are consonant. Now applying the same logic, a piece almost always ends in the same key, unless it is in a minor key in which it can end on the Tonic major (i.e a piece in A minor might end on a chord of A major, known as a *tierce de Picarde*). Similarly, it is most common to precede the final chord with a dominant chord (V), otherwise known as a perfect cadence. As a result of these two factors, multi-voice pieces constructed in this blind style will often start and end correctly and with a few glimpses of random consonance in the middle, the rest of the piece will be discordant.

Therefore, without any paramerisation in the system at all, the melodies produced both imitate Bach's style, and adhere to the rules of construction of music theory. (Pitch transition matrices from an order 1 analysis of the Bach Chorales can be seen in **Appendices C.1 to C.4**).

6.2 Generative Grammar Composition

In the past, the ADIOS algorithm has only been applied to Israeli folk music by a team at Tel Aviv University [26], and as such the algorithm has never been used to map Western Classical music. Further, the previous application of the ADIOS algorithm took different approaches attempting to treat words as bars, melodic notes as well as notes with accompanying chords. I, however, am using the algorithm to represent complex polyphonic harmonies in order to compose new compositions.Examples of compositions generated by this program can be seen in **Appendix A.2**.

The result of the structure of word that I have used allows for complex yet stylistically correct voice leading, as I shall demonstrate that certain movements are allowed. If we consider again the way in which the words are formed, they consist of a chord and rhythmic length of the form:

$$< pitch_1, pitch_2..., pitch_n >$$

where the index of the pitch corresponds to which voice the note belongs to. There are certain voice leading movements that aren't allowed within the music theory that governed Bach's input, and the prime rule that cannot be broken is that of parallel fifths or parallel octaves, which are seen in Figure 6.3. As we assume that in our input there were no instances of parallel fifths occurring, it is then impossible for a parallel fifth to have occurred on the diluted grammatical graph, and as such parallel fifths will not be found in any generated sentence. However, note that if a different set of music theoretical rules governed a different set of input files in which parallel fifths were allowed but parallel fourths were not, the same would be true. Thus the musical grammar of the input is perfectly maintained.

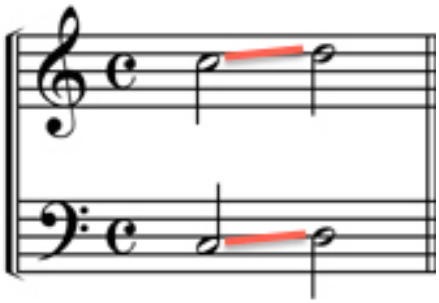


Figure 6.3: Example of parallel octaves

One problem, with this encoding of the words is that rhythm is not taken into account at all, and as such I had to artificially make all words 1 beat long when creating new sentences. Recall the

way that the chords were extracted from the score, if we had the following input:

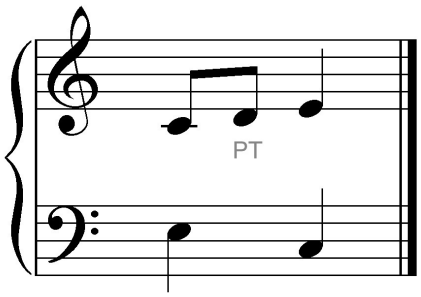


Figure 6.4: Example of a passing tone

the resulting chords would be:

$$< C, E > \ < D, E > \ < E, C >$$

If the rhythmic values could be taken into account somehow, the middle chord would only reappear as a swift motion between two other states, which is the nature of a passing tone - a *passing* dissonance. However, under this system, if the above sentence were to be recreated in a new sentence, each chord would have an equal weight, and as such the harmonic rhythm, which is characteristic of the input composer, is lost in translation. In order to begin to combat this, I implemented a system which retroactively updates note lengths if a part has two of the same pitches added consecutively. This at least allows for, in the passing tone example in Figure 6.4, the relative length of the Bass' E to be double the length of the Soprano's passing tone. Regardless, while the harmonic rhythm is largely lost, this encoding of words has maintained a context-free approach to harmonic grammar recognition and the creation of harmonically correct music. Without any concept of what constitutes dissonance or consonance, the output created is consonant and correct under the rules of Western Classical music. Similarly, were atonal music to be fed into the system, the output would again reproduce atonal music. In order to encode the words, there needed to be no analysis of the chord members, which allows for the program to be fully extensible to other musical styles.

There are problems, however, with using the ADIOS algorithm as a compositional tool. Firstly, given the vast number of possible notes and combinations of those notes, a vast data set is required in order to distill a meaningful graph and to find equivalences and patterns. Fortunately, with the 250 Bach Chorales the program was tested on, enough equivalences and patterns were recognised, as seen in **Appendix B.1**. Interestingly, the equivalences that were found match what are understood to be harmonic equivalences. For example, take equivalence class E4568:

```
5 P4567 --> <62,59,55,41> <60,
23 E4568 --> <64,55,48,36>
21 E4568 --> <64,55,48,48>
44 P4569 --> <62,55,47,43> E45
```

An equivalence has been found between all the same notes, apart from the bass note, for which there is a difference of 12. As such, since there are 12 chromatic notes, the bass in the second instance is an octave lower, but the same pitch. Therefore, in Western Classical terms, these are harmonically equivalent chords, and can generally be substituted in for one another in practise, and our system has discovered this equivalence.

Another large problem is that, as discussed earlier, the words in the sentences are treated atomically. In an earlier implementation of the program, I tried constructing words as a combination of both note values and length in the form $< pitch_1, pitch_2..., pitch_n > length$. However, immediate problems arose with this system, since for the parser there is no relationship between:

$$< C, E > quaver \text{ and } < C, E > crotchet$$

which seems problematic given that they are the same notes, simply elongated for a different amount of time, although in different time signatures, they might actually share the same proportion of a bar. The result was that the grammar found fewer equivalence classes and patterns, which as discussed above, led to less original content being reproduced.

In fact, in completing this project, I discovered the need for a modified version of grammar parsing algorithms which could serve the basis of a much more intelligent and effective music composition program. If we consider the process of grammar induction on natural language, the molecular units of any sentence is a word, which itself carries semantic meaning to a human. As semantic meaning is an extra dimension of a word that is irrelevant in the case of grammar, a grammar generator need not worry about the semantics of each individual word other than its signature with respect to letter order in a simple string matching fashion. However, if we consider music, as discussed in chapter 1, the atomic entities of music are multi-dimensional. If we consider a monophonic voice, the atomic entity is a note, which itself is two dimensional, incorporating rhythm and pitch. Now, if we consider the grammar of monophonic writing, there are certain transitions that we can assume will never appear in any corpus, for example a pitch transition of greater than a ninth. If this does not appear in the input corpus, then it cannot appear in any generated sentence. However, if we treat the atomic word representing a note to be in the form:

$$< pitch, rhythm > length$$

a simple string matching algorithm such as the ADIOS algorithm will not be able to see inside the brackets and, for sake of argument, infer that certain pitches may only transition to other pitches when $rhythm < X$. However, if we were able to introduce some sense of multi-dimensionality to the words, a parser might be able to infer some sense of meter by performing mathematical functions on the rhythmic length, and gleaning that adjacent sequences of notes always add up to a value of 1, which would represent a bar.

Further, if we consider the application of a more intelligent parser that can see within words, a parser might be able to realise that the following chords are in fact the same:

$$< A, C, E, A > \ < A, E, C, A >$$

Also the parser might similarly be able to extract common melodic movements, as if one voice moves from A to C to E in an arpeggaic fashion, the difference between the notes is the same as moving from D to F to A. As such, the melodic motif of an arpeggio might be inferred. Further, the passing tone example discussed earlier was an example of a $6-7$ passing tone, where the numbers indicate the interval between the top and bottoms notes. The current one-dimensional ADIOS algorithm can only learn instances of each exact passing tone, such as the $6-7$ passing tone above a C and the $6-7$ passing tone above an F. However, there is no deeper understanding that these movements are in fact similar, and the general movement of a $6-7$ passing tone is a music theoretical phenomenon.

Therefore I propose that a multi-dimensional grammar parser dedicated to music, which can perform mathematical calculations on note lengths and values, infer similarity of chords based on the member note values, as well as performing more complex and dedicated musical analyses might be an incredibly powerful tool in the realm of algorithmic composition, whilst still maintaining a context-free origin. Rather, certain harmonies, melodic movements, as well as structure of pieces may be inferred without any parameterisation required.

6.3 Melody First Markov Composition

While the *Generative Grammar Composition* is successful harmonically but rhythmically it is less intelligent, the *Melody First Markov Composition* creates the most convincing pieces of music, and I believe this is due to the way in which harmony was mapped and voices harmonised, as well as the novel approach of composing the voices both forwards and backwards.

As discussed in chapter 3.2, the higher the order of $n-gram$ upon which the text was analysed, the greater the capability of significant motif extraction of the system, and the lower the order, the less realistic the compositions. This is indeed true on comparing **Appendices A.8** and **A.9**. The latter is order $n = 1$, and we see that the melodic contour of the piece is much more static, there are unprepared dissonances, such as in the tenor line at bar 3, and harmonically it is less interesting. If we consider that there are likely more instances of notes and chords that revolve around the tonic, it makes sense that with only the ability to see one note previous, the matrix might regularly produce notes that revolve around the tonic without being able to expand into the more interesting harmonic progressions that escape the tonic and dominant and add flare and character to a piece of music.

For higher order pieces, the results are much more promising. As discussed earlier in this chapter, the melodies produced are representative of the input Bach Chorales. The melody is the foundation of this type of composition, as the Soprano melody is produced in the same way as in *Melodic Markov Composition*, and then given a harmonic skeleton. Harmonically, the pieces generated are convincing, though not as much as *Generative Grammar Compositions*. In every instance, the pieces start and end in the correct fashion, and each note is harmonised, thanks to the novel bidirectional approach. Take the last three bars of **Appendix A.10** as seen in Figure 6.5, for example. The piece, in A minor, ends with a the progression $i - V7 - i - iv - V7 - I$. This is a perfectly legal and harmonically consonant chord progression that is typical of Bach, even with the added flare of ending with a *tierce de Picarde*. Similarly, if we look at the construction of the melody, there is symmetry between the two bars. The final three notes of the Soprano's melody is a lower neighbour motion, moving down one tone and then immediately back up (A-G#-A). This is foreshadowed at the beginning of the previous bar with the quaver movement (C-B-C). Further, rhythmically these last two measures are symmetrical, with the phrase itself being split into two sub-phrases, one per bar. Each sub-phrase ends with a strong minim on the third beat of the bar. Therefore, even in this small segment of music, the result of the multi-faceted Markovian analysis in combination with randomisation has resulted in music rife for musical analysis.

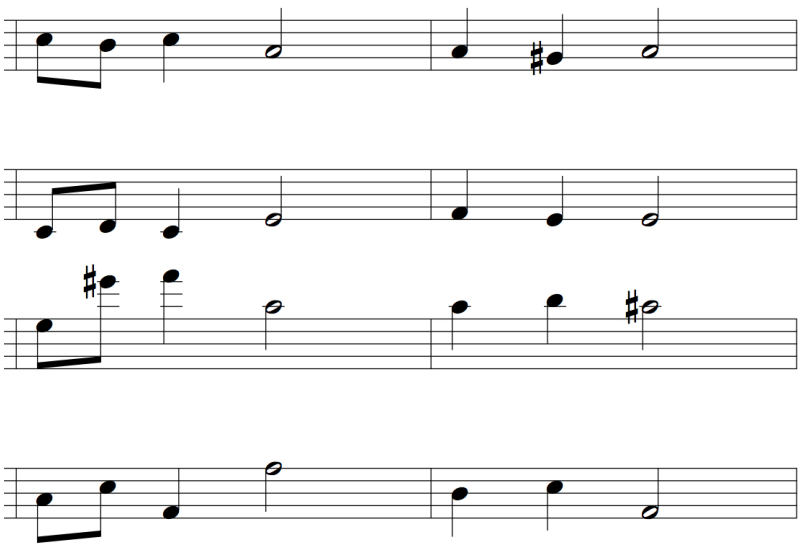


Figure 6.5: The last two bars of A.10

One drawback to the algorithm that I have used is that on creating the melody, each melodic note is harmonised as opposed to harmony resulting from the interweaving of polyphonic parts. While for longer melodic notes this one-to-one harmonisation is not a problem, when the melody consists of semi-quavers, and each note is harmonised separately, the piece becomes too busy, and the harmony lost on the listener. It is not immediately obvious to me how to rectify this issue without pre-programming certain music theoretical rules, defining what non-chord tones can appear in certain chords, or how quickly harmonic rhythm should move. This could be an interesting theme in the future development of the program.

The system is also prone to harmonising notes with severe dissonances. This is likely because the input scores have modulations within them. These modulations, or brief changes of keys, introduce the possibility for new chords, and also new chords with dissonant non-chord tones, such as passing tones and neighbours, that would have been read in from the score. Certain keys that might be modulated to will have certain similar chords with the overall piece's tonic key, and as such, transitions will appear in the matrix for those shared chords to both the chords in the modulated key, as well as the chords in the home key. For example, the key of C Major contains no sharps or flats, and the chord C-E-G is a valid chord within C Major. However, the key of G Major contains one sharp, F#, yet the chord C-E-G is also a valid within G Major. Now if we consider that a chord with any of the pitches moving one or two semi-tones in either direction likely appeared in the input corpus as a passing tone or neighbour tone, we can infer the chord -C-F#-G, where the key was G Major, and the second voice moved up a tone. Now there is a possible transition within the matrix, even if in C Major, to a very discordant chord. While this type of chord is more unlikely to appear without the correct context and preparation with a higher $n - gram$, reconsider the way in which chords are found. If a possible transition cannot be found, the $n - gram$ is shrunk by one removing a value from the front. As such, at the point that the system cannot find an appropriate transition and the $n - gram$ reaches size one, the context and preparation of the dissonant chord is lost, but the transition still a possibility.

Finally, as is always the way with Markovian composition, the compositions formed do not have any overall logical structure in terms of melody or harmony. In the realm of composition, melodies consist of motivic incipits, which are stated, developed and repeated. In this system, the composition algorithm is only influenced by the past up to the size of the $n - gram$. As such, if a motif occurs in the first bar, it will only be by pure chance that it is repeated or developed in another bar, and the two will not be connected logically.

Overall, the music is polyphonic as each voice has its own melodic and rhythmic character, and clearly in Bach Chorale style, despite not encoding any rules of music theory that Bach used to create his own pieces. Harmonically, the pieces are convincing also, both starting and ending with the appropriate tonalities, and with strong, mostly legal progressions within the realm of the Western Classical tradition. As such, the compositional program has correctly rendered convincing pieces of complex music in an entirely context-free environment.

6.4 Application to Other Works

To check whether or not the system truly represented a context-free compositional agent, I decided to test the program on another set of works using a different combination of voices, composed in a Classical style as opposed to the Baroque style of Bach. Therefore, I tested the program on Mozart's *Requiem*, which is representative of the Classical style. While the system worked perfectly, analysing the scores just as it had with the Bach Chorales and recreating new music, the output

was largely disappointing. Despite the quality of the compositions, the successful execution of the system confirms that I have made a program capable analysis, and as such, the produced scores will all at the very least be in the harmonic and melodic style of the input. Therefore, I have successfully created a context-free system.

As for *Generative Grammar Composition*, the grammar inferred was minimal with only four equivalences being found. Unfortunately, this meant that generated sentences largely resembled the input corpuses, and as such, these compositions were unsuccessful as they were not original enough, rather largely representing the input sentences extracted from the input scores.

Otherwise, Markovian Composition was better, however, it also was not as effective as the Bach Chorales. Firstly, there was relatively little data since the Mozart Requiem consists of only 12 movements. However, Mozartian music, along with most other music, places more emphasis on overall structure, with pieces consisting of sections which have thematic motifs, and other sections which develop those motifs. Further, while the chorale texture is largely homophonic four-part writing style, more complex music uses a variety of textures. For example, one groups of voices will sing a phrase, and that phrase will be responded to by another group of voices in an antiphonal manner. These textural devices cannot be picked up in the current system I have developed, and as such, does not lend itself well to the analysis of non-chorale type pieces. Another corpus of works that I imagine the system would work well on is hymn tunes, as they are very similar to Bach Chorales texturally. Further, the current Markovian analysis is perfect for generating short excerpts, such as a melodic incipit or main theme, however, it does not lend itself to creating larger pieces constructed of logical parts, and as such any output based on larger pieces with a concrete section-based form will only represent the compositional style in terms of harmony, melody and rhythm as opposed to structure or texture.

Figure 6.6 is an example of a *Melody First Markov Composition* using Mozart's *Requiem* as input data. It is immediately obvious that the textural differences of the input has led to sparsity in the voices. However, the piece is still harmonically sound, as well as maintaining melodically sound voice parts. For example, the Soprano line is still largely conjunct and is melodic in its own right.

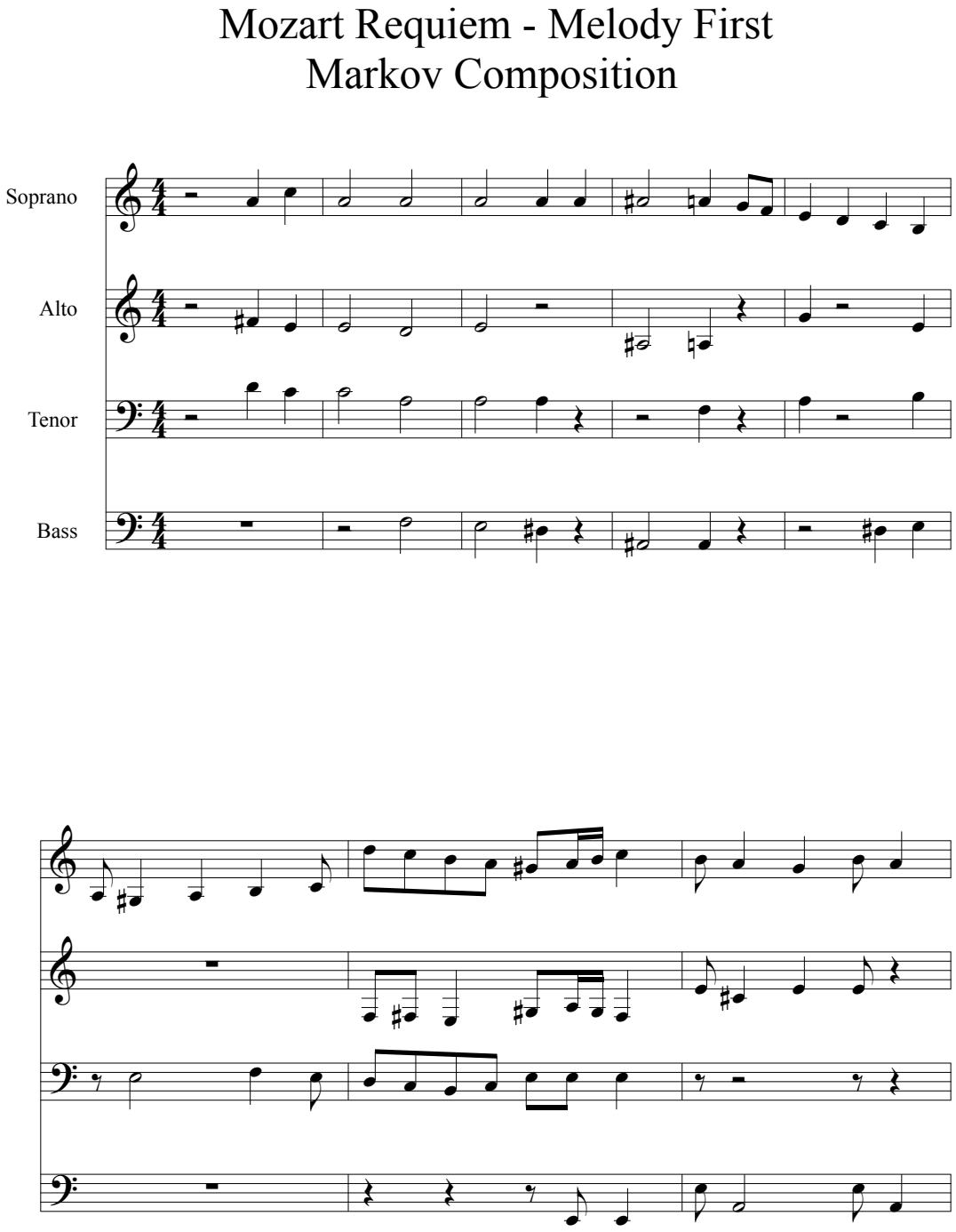


Figure 6.6: Melody First Markov Composition using Mozart's *Requiem* as input data

6.5 Decisions that Affected the Final Product

The central thesis of this project was to construct a context-free composer, however, that involved my choosing of the most fundamental elements of music common to all pieces. For example, with respect to individual notes, I chose to separate pitch from rhythmic value and map the two individually, however, it would have been possible to treat a note-rhythm pair as atomic. Similarly, I did not separate matrices based upon time signature, although there is a reasonable case for doing so, since the construction of melodies varies due to the different beats within a measure that are stressed. Therefore, the system is by no means perfect, but I believe the design offers the greatest ability for the system to learn the pieces' style as the data is analysed at the most fundamental level of pitch, rhythm and harmony. Again, considering the way harmony is analysed, it is the sequence of all notes that appear simultaneously. Each permutation of the same chord is treated atomically, and it is only a music theoretical construct that dictates that the C-E-G and G-C-E are the same chords but in a different inversion. In fact, I believe it is right to treat them as individual entities as certain inversions of chords do function entirely differently to certain other inversions of the same chord, such as the Cadential 6/4, an inversion of the tonic, only ever appearing preceding a perfect cadence.

There are certain other decisions that I had to make arbitrarily, but still have a significant impact on the final product. For example, when a harmonic skeleton around a melody is established, each voice simply takes its appropriately indexed value. It would have been possible, however, for each to randomly generate a new note in a melodic fashion from its own transition matrices, and continually repeat the process until it matches a note in the chord, and then to remove that note from the chord such that other voices won't repeat the note, and all note values will finally be represented. However, I chose not to do this based on the justification of treating inversions of notes as separate, as well as the fear that the dataset held within the matrices be too sparse, and a deadlock be met where the melody cannot generate a note that matches a note within the chord.

One other decision that I did have to make included a certain degree of music theoretical parameterisation, and that was in the way that I chose the scores to analyse. Originally I had access to over 300 Bach Chorales. However, after running the program multiple times and the resulting compositions being much more dissonant than I expected, I looked into the input scores and realised that some of the scores were incorrectly labelled in terms of key signature. This was common for Bach, who often composed with an incorrect key signature, or wrote pieces in other modes. As such, on the first run through, I did clean the data such that if the score did not end in A Major, A minor or C Major, the score was ignored. While this is a paramterisation that is not in keeping with the context-free thesis of this project, as the process establishes and labels a final chord within the confines of a certain music theoretical framework, I believe that this parameterisation effectively occurs outside of the overall algorithm of analysis and composition, since ignoring certain files is akin to never having included them in the first place. As such, I believe the integrity of the context-free nature of the program is maintained.

Ultimately, however, I believe the system is context-free and is not guided or parameterised in any sense. Rather it infers melodic, rhythmic and harmonic rules and reapplies them in a novel way. In fact, by separating pitch and rhythm at a note level, it increases the likelihood that a piece will not mirror one of the melodies in the input, as there is no defined relationship between pitches and rhythm, and in the generation of new melodic material there are two levels of randomisation.

Chapter 7

Conclusion

In this chapter I offer my closing remarks, as well as possible extensions or developments to the program.

7.1 Applications and Future Work

This program can be used as a powerful tool in composition, aiding a composer by providing them with inspiration of harmonic or melodic fragments. The infrastructure of the program can easily be expanded through paramterisation if the user wanted to emulate a certain style, and forbid or require certain melodic or harmonic movements.

As discussed previously, I have proven that a naive Generative Grammar Algorithms that does not look within words is successful in the composition of music. A future proposal that would benefit the future of algorithmic composition would be a grammatical inference algorithm that could successfully read multi-dimensional musical sentences and infer a grammar based on but not limited by:

1. Rhythmic lengths of measures
2. Common motivic movements based on melodic contour, e.g. all melodic arpeggios have a common transition sequence
3. Harmonic equivalences of chords and their inversions, e.g. C-E-G has some sort of equivalence to E-C-G given they have the same member values and the same base, and C-E-G has some equivalence to C-G-E given the have the same member values but a differing base
4. Relationships between chords and rhythmic lengths

Further, the developed program serves as a basis of a much more intelligent composer, which not only considers rather simple 8-bar, four-part harmonies, but could also accept more texturally

complex pieces, such as a Chopin Waltz. The **Grammar_Parser** class can be used on multiple sets of sentences, which could perhaps represent melodic composition, and as such extract melodic grammars, or even infer a sense of overall piece structure, such as Sonata and Ternary Form. A recombination of all distilled grammars could produce an incredibly realistic and impressive outcome. The **Score** class has been developed to accept and represent any number of parts, with any number of bars and any type of notes. It would have to be expanded to accept multiple voices within a **Part**, such as if it were to be supplied with a Piano that has chords within one part, but this would be a simple modification when utilising the libmusicXML2 library.

A future developer might even incorporate a database system, in which the user of the program could start by defining certain sets of pieces by their composer or type (e.g sonata or string quartet), and the program would store its transition matrices and grammars for the future. This would allow for the system to continually evolve the more it was used, and it could also be further expanded into a program capable of recognising a composer given a piece of music or what era a piece is from based on certain harmonic or stylistic features. Similarly, further matrices might be developed measuring other aspects of a score, provided the data analysed was elemental to all music. For example, currently each matrix does not take into account time signature, and as such data from pieces in 3/4 goes into the matrices with data from pieces in 4/4. If enough scores were available, these matrices could be separated, as there are certain characteristics of a 3/4 melody and how harmony is constructed in this time signature that differ from that of 4/4. This is largely due to the different emphasis of stress within each measure. In fact, if stress is considered a fundamental element, the context-free composer could be expanded by creating separate matrices for stressed and unstressed unknown states. If, at time of composition, you are looking for a note that falls at the beginning of the beat, you would query the stressed matrix with the $n - gram$, for example.

One final way in which this platform might be extended is to present certain grammatical rules distilled by the ADIOS algorithm, as well as any significant harmonic or melodic transitions from the matrices, such that a list of rules of construction of the style provided to the program can be read by the user. With this list of rules, the composer could hand write new compositions using the stylistic traits and rules of construction of any composer of their choosing, regardless of their music theoretical education.

7.2 Summary

Overall, this project has demonstrated the ability of Markovian processes to not only create melodic material, a task which they have previously been proven to be successful in, but also to harmonise voice parts and create convincing and complex pieces harmonically and rhythmically. The program developed is capable of creating a piece of music with no parameterisation on any set of music theoretical grammar, and the output produced typically emulates both the style and grammatical rules of the input. Using a simple expansion of the Markovian system, analysing a piece of music

both forwards and backwards temporally, as well as analysing the many facets and dimensions of all music separately, the output that the algorithmic composer can generate is much more complex than many Markovian systems have achieved before, such as the Brooks, Hopkings, Neumann and Wright's melodies seen in 3.3. Further, this project has demonstrated that the ADIOS algorithm, is incredibly powerful at distilling a harmonic and music grammar which truly emulates the style of the input. As such, in the creation of a context-free algorithmic composer, two simple techniques have proven very successful in the emulation of style and inference of music theoretical grammar.

Overall, the developed system shows the capability of this approach as a context-free agent. The design of the program was such that MusicXML files representing any type of notated score can be submitted. Similarly, there is no limit to the number of parts each file contains, or the number of measures in the score. With the extensibility of this design, the program is capable of receiving any set of files representing scores composed under any music theoretical paramaters, and as long as enough data is provided such that a grammar can be induced, the output will be in the style of the given composer. If, say you liked the modal harmonies of Gabriel Fauré, but did not know how they were composed, you could use this program and create entirely new compositions in that style without having to encode any parameters such as Fauré's favoured use of whole-tone melodic lines.

The aim to create a truly intelligent algorithmic composer is by no means a simple feat, however, the work that I have presented here shows that even utilising the simple concepts that have long been used in the realm of algorithmic composition, Markov processes and Generative Grammars, can be extended in their use in order to create more convincing complex music, and without any paramaterisation in terms of rules of construction.

Appendix A

Compositions

In this appendix I shall present some example scores generated by the Algorithmic Composer program. Each score represents the first created score per execution of the program.

A.1 Melodic Markov Composition

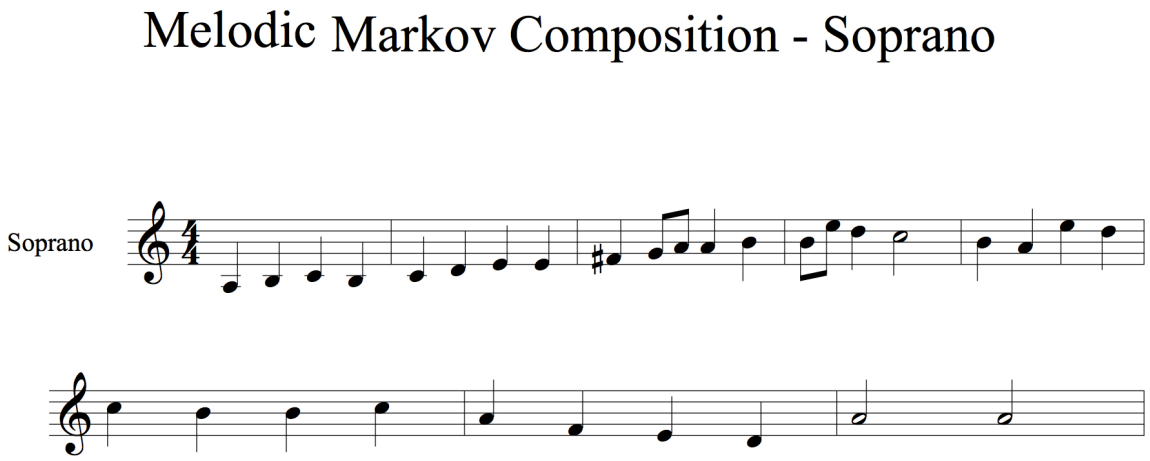


Figure A.1: Markovian Soprano Melody created using order 3 in A Minor

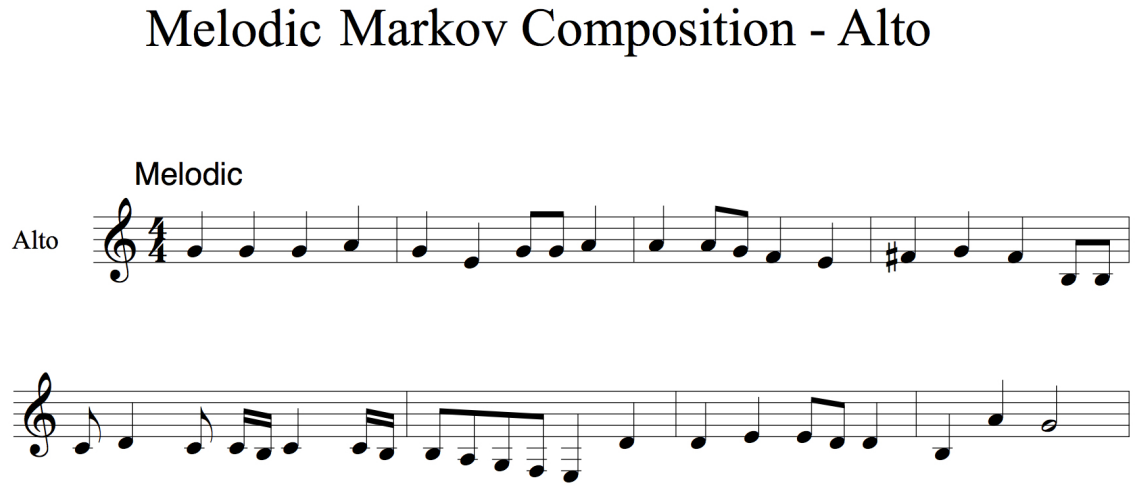


Figure A.2: Markovian Bass Melody created using order 5 in C Major

Figure A.3: Markovian Tenor Melody created using order 4 in C Major

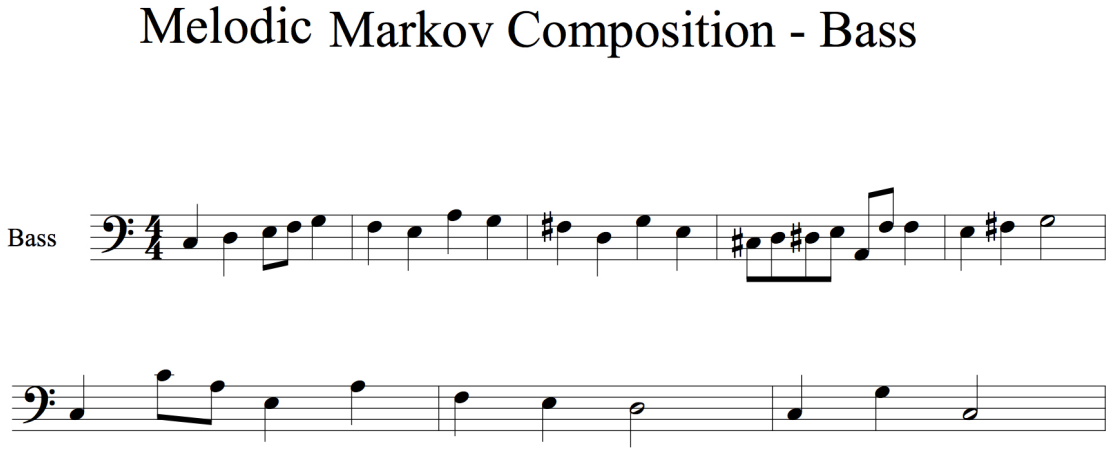


Figure A.4: Markovian Bass Melody created using order 5 in C Major

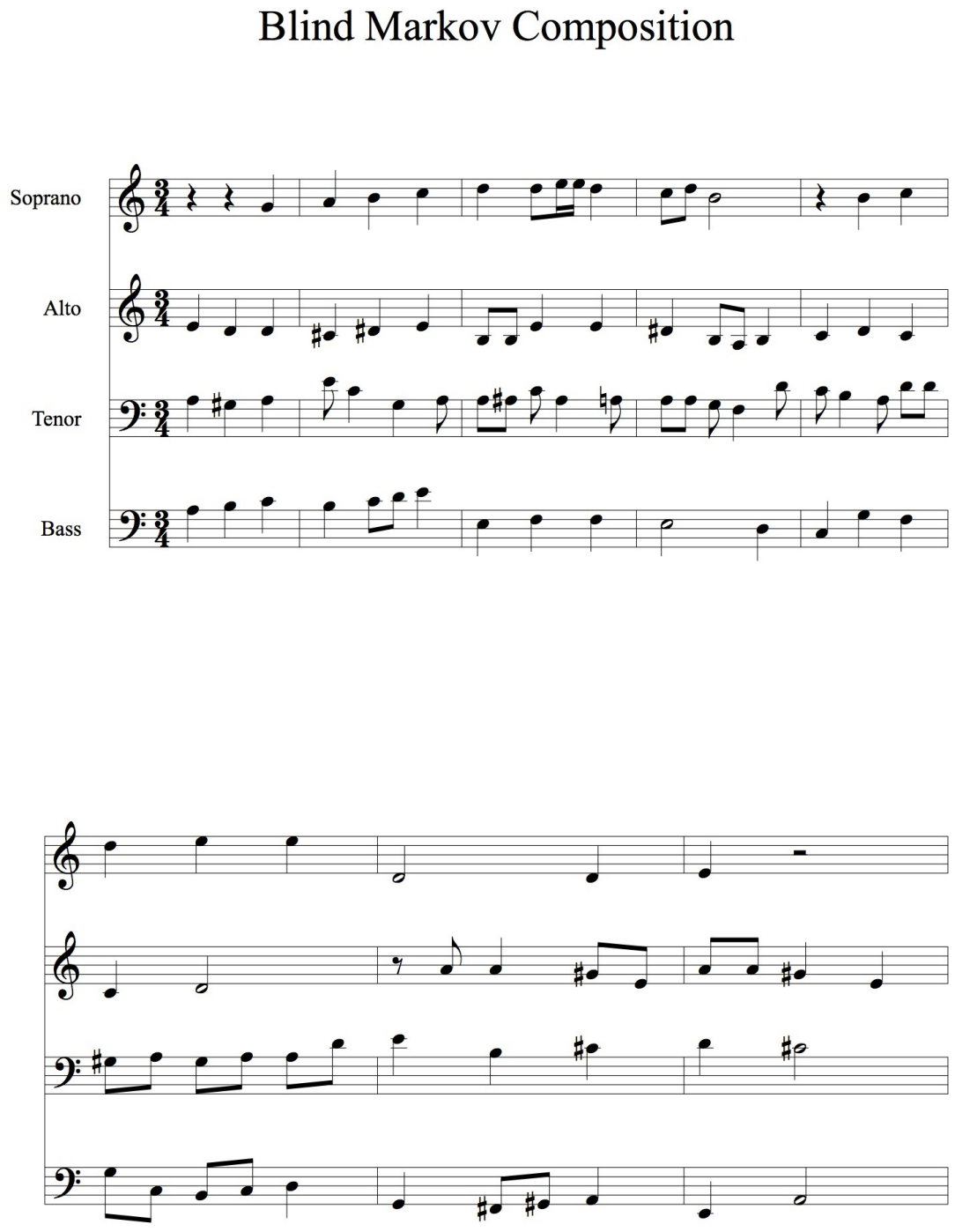


Figure A.5: Blind Four Part Markov Composition using Matrix Order 5 in A minor

A.2 Generative Grammar Composition

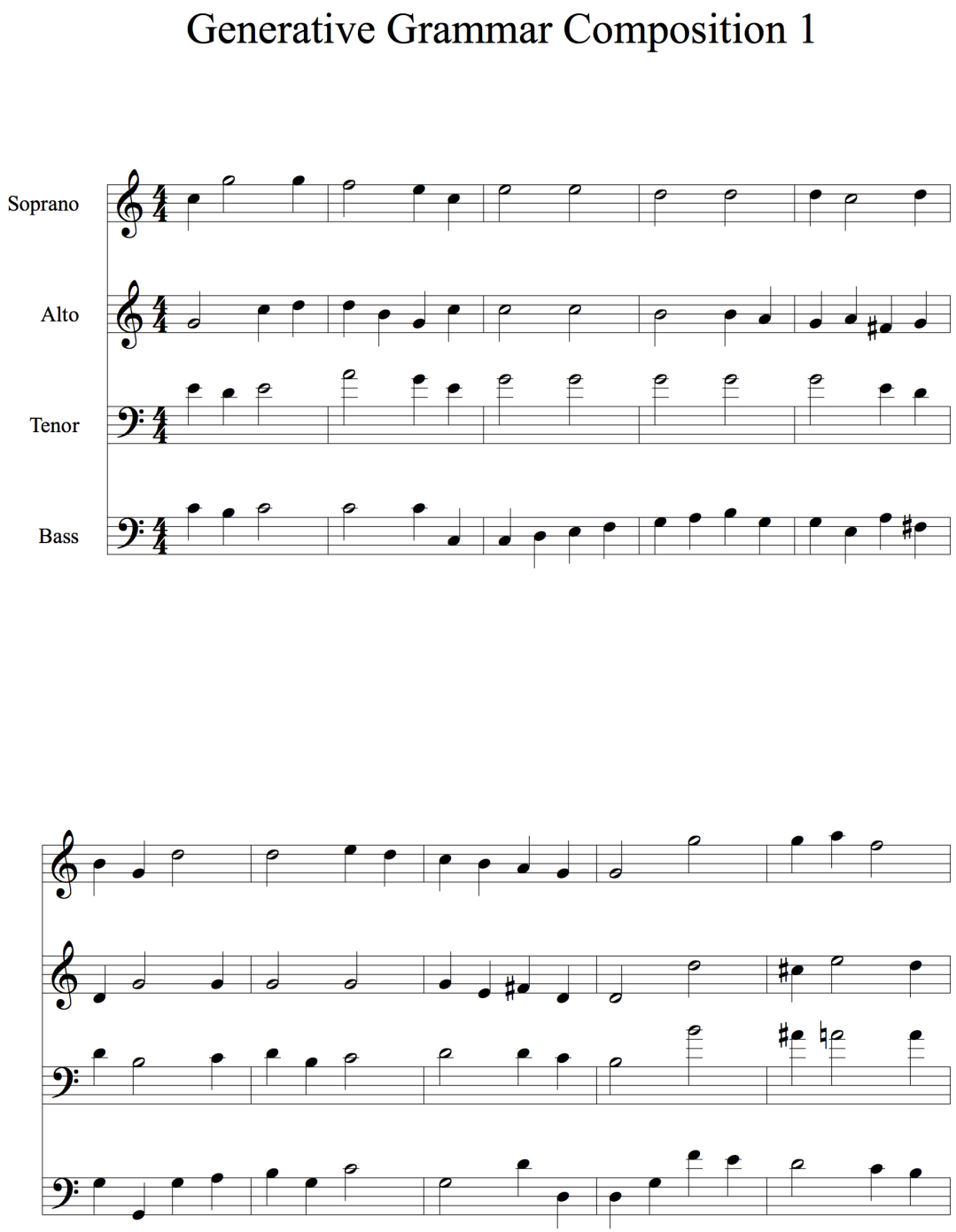
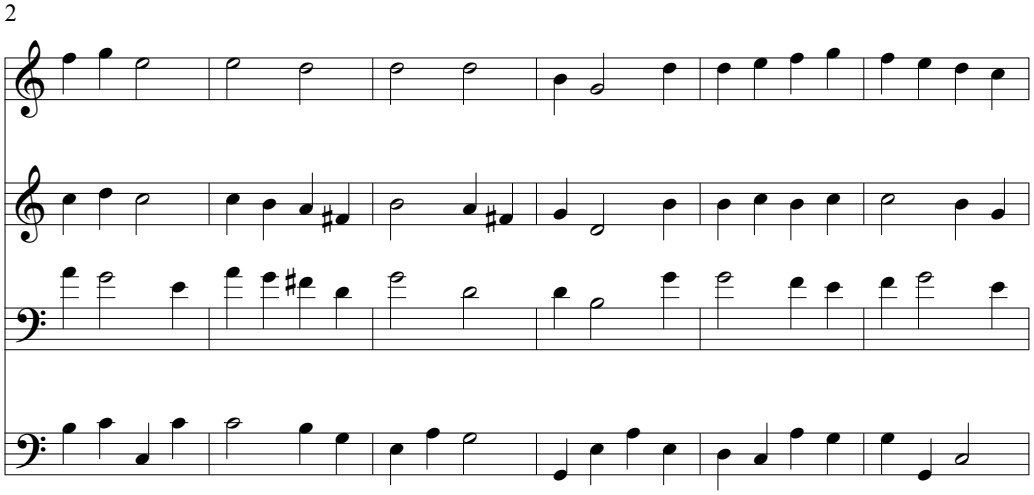


Figure A.6: Generative Grammar Composition


2

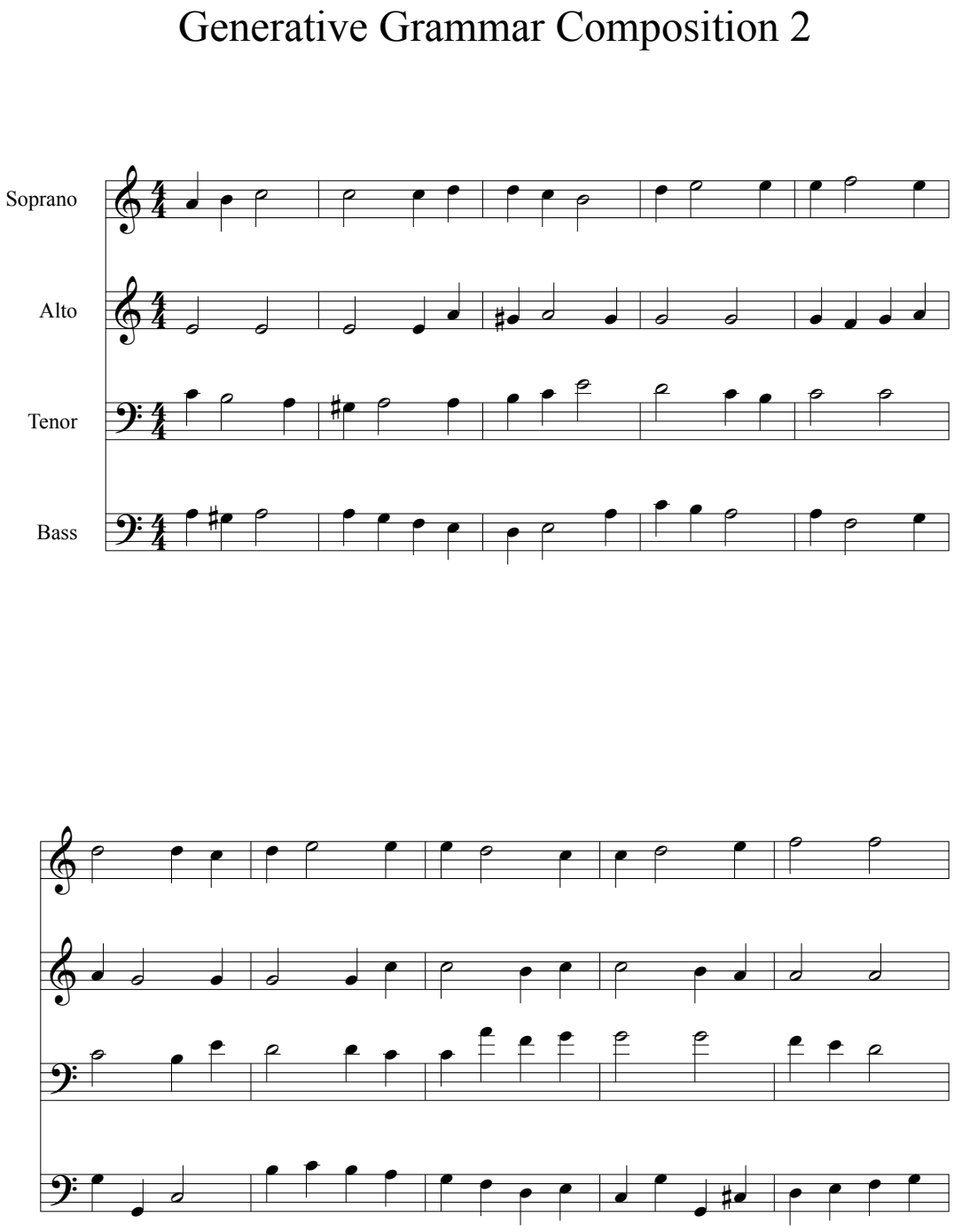
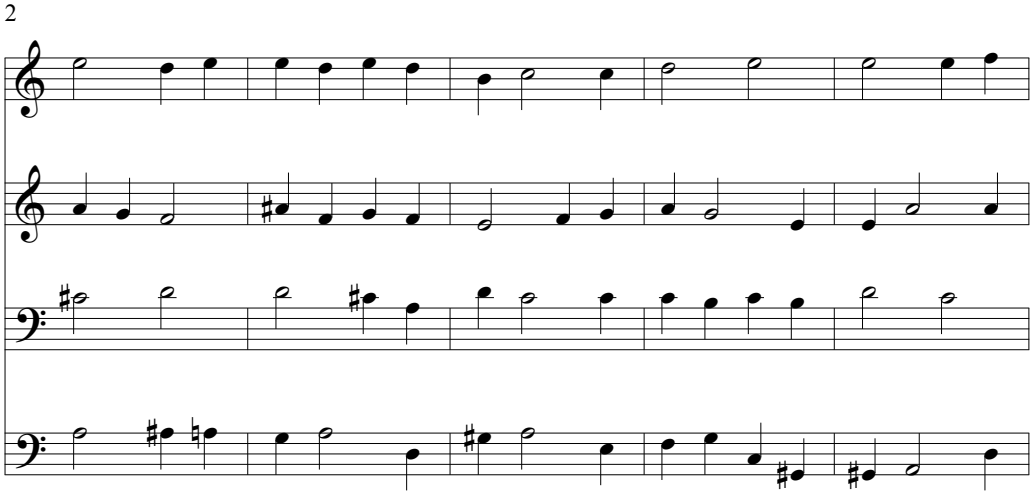
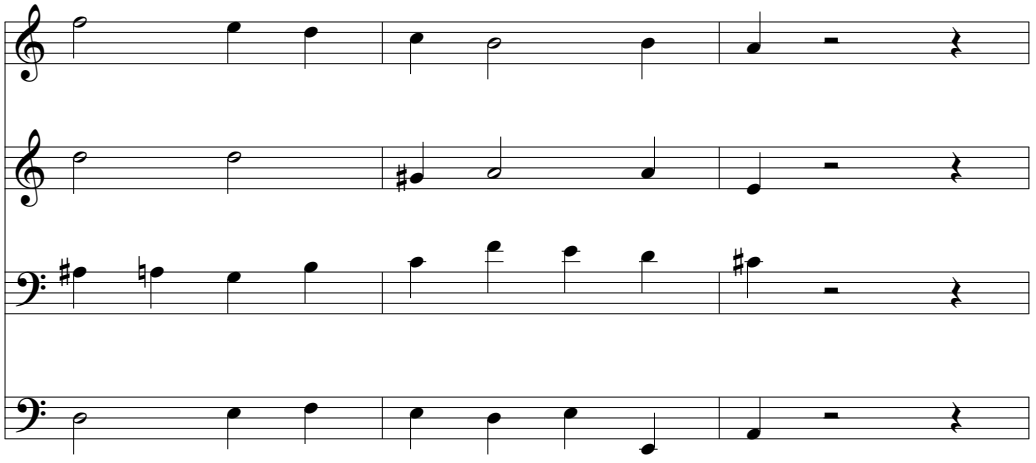


Figure A.7: Generative Grammar Composition 2


2



A.3 Melody First Markov Composition

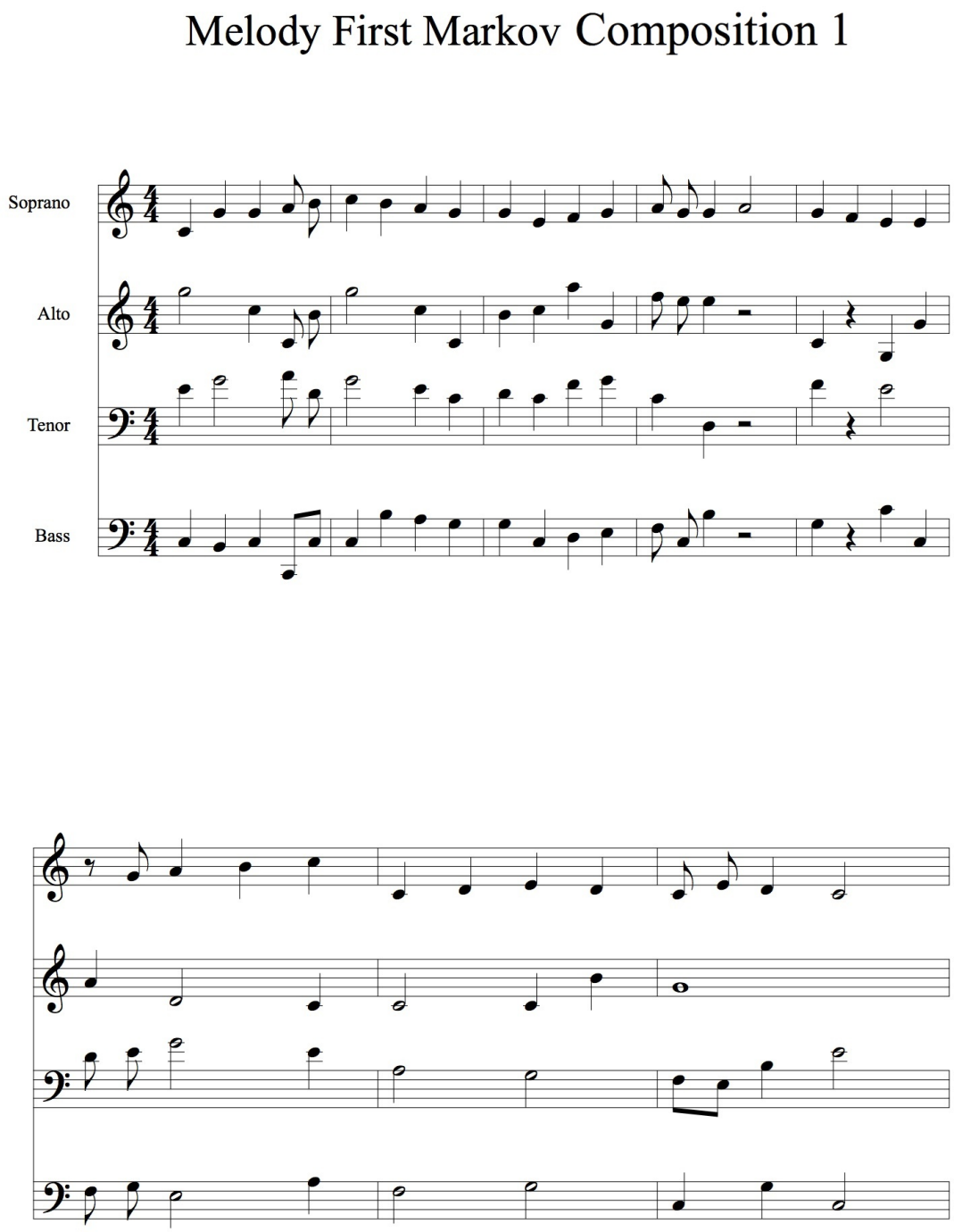


Figure A.8: Melody First Composition using Matrix Order 4 in C Major

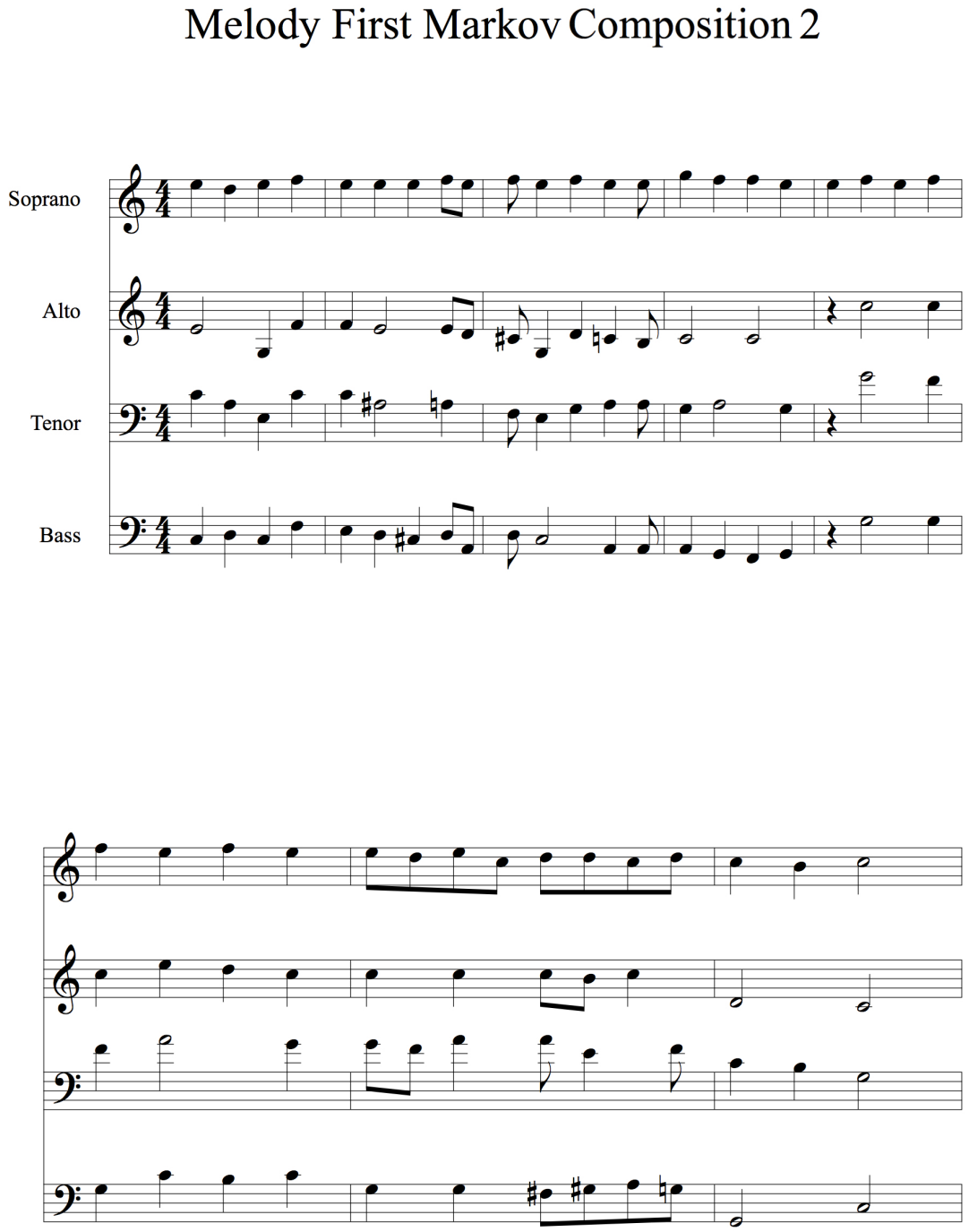


Figure A.9: Melody First Composition using Matrix Order 1 in C Major

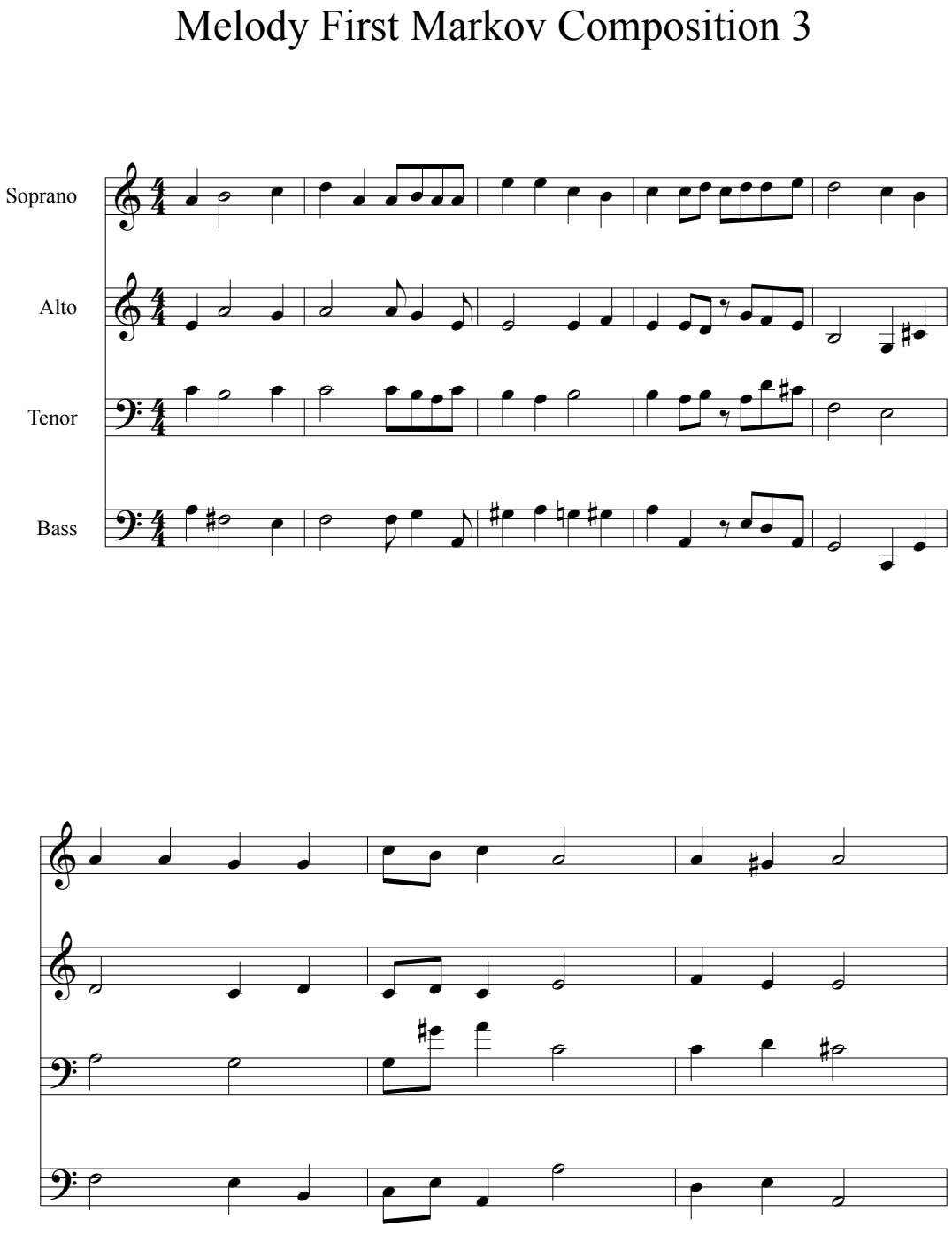


Figure A.10: Melody First Composition using Matrix Order 5 in A Minor

Appendix B

Distilled Grammar

This is an example of a distilled grammar after a successful execution of the program with 250 Bach Chorales. Note that E stands for equivalence class, and P stands for pattern. Please see chapter for an explanation of the significance of these terms. The number to the left E/P signifies the number of times the equivalence class/pattern occurred.

```
S
7 P4498 --> <57,53,48,41> <59,53,48,43> <60,52,48,45>
31 E4499 --> <59,56,50,40>
1 E4499 --> <57,56,50,40>
32 P4500 --> <59,57,53,38> <59,56,52,40> E4499
17 E4501 --> <62,53,47,43>
1 E4501 --> <60,53,47,40>
18 P4502 --> <62,55,47,43> E4501 <60,52,43,36>
3 E4503 --> <62,57,50,47>
3 E4503 --> <62,57,50,45>
6 P4504 --> <62,55,50,47> E4503 <62,59,55,43>
3 P4505 --> <55,52,48,36> <55,52,50,35> <60,57,52,33>
1 E4506 --> <62,55,50,43>
38 E4506 --> <62,59,53,43>
39 P4507 --> <62,59,55,43> E4506 <60,55,52,36>
3 P4508 --> <52,48,43,36> <52,47,43,36> <53,45,41,38> <53,47,41,38>
19 P4509 --> <64,55,48,40> <64,55,48,41>
17 P4510 --> <62,57,53,38> <62,59,53,38>
26 P4511 --> <57,54,50,38> <57,54,48,38>
2 E4512 --> <55,53,50,35>
1 E4512 --> <55,52,45,35>
3 P4513 --> <57,53,50,38> <57,53,50,40> <57,53,50,41> E4512 <57,52,45,37>
3 P4514 --> <57,53,50,38> <59,53,50,38> <60,52,48,45> <60,52,47,43> <62,50,45,41>
23 E4515 --> <48,43,40,24>
12 E4515 --> <48,43,40,36>
29 P4516 --> <50,47,43,31> <50,47,41,31> E4515
4 P4517 --> <50,43,41,36> <52,48,43,36> <53,47,41,38> <55,48,40,40> <55,52,48,36>
<57,53,48,41> <57,52,48,41> <59,50,43,43> <60,52,43,36> <50,43,40,35> <52,48,45,33>
<53,50,45,31> <53,50,45,29> <55,50,43,28> <55,48,40,33> <53,48,41,26> <53,47,41,26>
<52,48,43,28> <52,48,43,29> <50,47,43,31> <52,47,40,32> <48,47,40,33> <48,45,40,36>
<48,45,40,33> <53,45,38,38> <53,47,38,38> <52,48,40,33> <52,48,40,31> <50,48,45,29>
P4516
2 E4518 --> <65,57,48,47>
52 E4518 --> <62,55,50,47>
54 P4519 --> E4518 <64,55,48,48>
7 P4520 --> <62,57,48,41> <62,57,48,38> <62,55,47,43>
2 E4521 --> <60,52,47,43>
1 E4521 --> <59,52,47,43>
3 P4522 --> <64,52,47,44> <60,52,45,45> E4521 <57,57,48,41> <57,57,48,40> P4500
4 E4523 --> <57,54,48,50>
2 E4523 --> <57,54,48,38>
6 P4524 --> <57,54,50,50> E4523 <55,50,47,43> <60,57,52,45>
17 P4525 --> <65,57,48,45> <65,57,50,47> <64,55,52,48>
11 P4526 --> <60,52,43,40> <60,52,45,42> <59,50,47,43>
4 P4527 --> <62,55,50,36> <62,55,50,35> <64,55,48,36>
5 P4528 --> P4525 <64,57,53,48> <62,59,55,43>
23 P4529 --> <64,57,48,45> <64,57,48,43>
25 P4530 --> <59,52,47,44> <60,52,45,45>
1 E4531 --> <60,57,52,45>
2 E4531 --> <60,57,52,36>
3 P4532 --> <59,55,50,43> <60,55,52,36> E4531 <62,59,55,43>
11 P4533 --> <55,48,40,36> <55,48,41,38> <60,48,43,40>
4 P4534 --> <57,52,48,33> <59,52,50,33> <60,57,52,45>
6 P4535 --> <67,60,52,45> <65,60,53,38> <65,59,53,38> <64,60,55,40> <64,60,55,41>
<62,59,55,43>
4 P4536 --> <64,55,48,48> <62,55,50,47> <60,57,52,45> <60,57,54,45> <59,50,55,43>
7 P4537 --> <65,60,57,45> <65,60,57,47> <64,60,55,48>
9 P4538 --> <55,52,47,40> <55,50,47,40> <57,48,45,41>
8 P4539 --> <60,57,53,45> <62,55,52,47> P4519
4 E4540 --> <60,57,53,52>
2 E4540 --> <62,59,53,52>
6 P4541 --> <60,57,53,53> E4540 <62,59,53,50> <64,60,55,48>
9 P4542 --> <64,60,55,47> <65,60,57,45> <67,59,50,43>
5 P4543 --> <59,56,52,40> <60,56,52,38> <60,57,52,36> <59,57,52,38>
6 P4544 --> <47,44,40,28> <47,44,38,28>
3 E4545 --> <64,57,50,47>
3 E4545 --> <64,56,50,47>
6 P4546 --> <64,57,48,45> E4545 <64,57,52,48>
4 P4547 --> <59,50,43,35> <59,52,43,36> <57,53,50,38> <57,53,48,38> P4538
10 P4548 --> <64,59,52,43> <62,57,53,41>
6 P4549 --> <59,53,50,38> <59,53,48,38> <56,52,47,40>
```

Figure B.1: Generated Grammar by the ADIOS algorithm of the chords from Bach Chorales

```
5 P4550 --> <60,57,53,45> <62,55,52,47> <62,53,50,47> <64,52,48,48>
37 E4551 --> <64,60,55,36>
17 E4551 --> <62,59,53,43>
54 P4552 --> <62,59,55,43> E4551
5 P4553 --> <60,52,45,45> <59,52,45,45> <60,52,44,40>
14 P4554 --> <53,50,43,35> <52,50,43,36> <52,48,43,36>
14 E4555 --> <64,60,55,41>
3 E4555 --> <65,60,55,36>
17 P4556 --> <64,60,55,40> E4555 <62,60,55,43>
12 P4557 --> <62,55,47,43> <62,53,47,43> <60,52,48,45>
4 P4558 --> <60,55,52,36> <62,55,53,36> <64,60,55,48> P4542 <67,59,52,43>
<65,57,53,38>
3 P4559 --> <59,53,45,38> <59,53,45,35> <59,52,44,40>
4 E4560 --> <64,55,48,40>
3 E4560 --> <62,55,47,40>
7 P4561 --> <62,55,47,41> E4560 <64,55,48,38> <64,55,48,36>
6 P4562 --> <55,52,48,36> <55,53,48,38> <60,55,48,40> <60,55,50,41> <60,55,52,43>
<60,55,50,43>
3 P4563 --> <59,57,53,38> <59,57,47,38> <59,56,52,40> <59,56,50,40>
3 E4564 --> <60,59,52,43>
13 E4564 --> <62,59,52,43>
4 E4564 --> <62,59,50,43>
1 E4564 --> <60,55,52,43>
1 E4564 --> <60,59,53,43>
3 E4564 --> <62,57,53,43>
25 P4565 --> <62,59,53,43> E4564
4 P4566 --> <67,62,55,47> <67,62,53,47> <67,60,52,48>
5 P4567 --> <62,59,55,41> <60,60,55,40> <62,60,57,41>
23 E4568 --> <64,55,48,36>
21 E4568 --> <64,55,48,48>
44 P4569 --> <62,55,47,43> E4568
1 E4570 --> <53,47,43,38>
7 E4570 --> <57,48,41,41>
8 P4571 --> <52,48,43,36> E4570 <55,48,43,40>
3 E4572 --> <50,43,41,24>
3 E4572 --> <50,43,41,36>
6 P4573 --> E4515 E4572 <52,48,43,36>
6 P4574 --> <60,57,50,42> <59,57,50,43> <59,55,50,43>
4 P4575 --> <59,55,50,43> <59,55,50,42> <60,55,48,40>
3 E4576 --> <60,54,50,33>
1 E4576 --> <60,55,50,35>
4 P4577 --> <60,55,52,36> E4576 <60,54,48,33> <59,55,50,31>
1 E4578 --> <60,52,45,38>
2 E4578 --> <59,53,45,38>
3 P4579 --> <60,52,45,40> E4578 <59,52,44,40> <59,52,50,40> <57,52,48,33>
4 P4580 --> <50,48,45,29> <52,48,45,29> <53,48,45,29> P4516
3 E4581 --> P4519
1 E4581 --> P4539
4 P4582 --> <59,56,52,40> <60,57,52,45> E4581 <64,55,48,47> P4529 <62,57,48,41>
4 P4583 --> <65,60,57,45> <67,59,55,40> <67,59,53,40> P4535 <64,60,55,48>
<64,60,55,47> <65,60,57,45>
19 P4584 --> <60,52,45,36> <60,52,45,38>
1 E4585 --> <53,47,43,31>
2 E4585 --> <53,50,43,31>
3 P4586 --> <55,50,43,35> <55,48,43,33> <53,50,43,35> E4585 <52,48,43,36>
4 P4587 --> <64,60,55,45> <63,59,54,47> <63,59,54,45> <64,59,55,43> <66,63,57,42>
<67,64,59,40> <69,66,59,39> <67,64,59,40>
4 P4588 --> <48,47,40,33> <48,45,40,33> <50,45,38,35> <50,43,38,35> <52,43,36,36>
4 P4589 --> <55,52,47,40> <55,52,48,33> <53,50,43,35> <52,48,43,36> <57,48,45,41>
1 E4590 --> <62,53,45,38>
1 E4590 --> <60,52,47,38>
8 E4590 --> <59,53,45,38>
10 P4591 --> E4590 <59,52,44,40> <59,50,44,40>
3 P4592 --> <60,52,43,36> <64,57,48,48> <64,59,47,44>
5 P4593 --> <60,48,40,33> <60,50,40,35> P4584
5 P4594 --> <60,55,48,40> <62,55,48,38> <64,55,48,36>
3 P4595 --> <59,50,44,35> <60,48,45,33> P4584
42 P4596 --> <60,57,52,45> <60,57,52,43>
1 E4597 --> <62,59,52,48>
2 E4597 --> <62,57,50,47>
3 P4598 --> <60,57,52,50> <60,57,52,48> E4597 <62,56,50,47>
```

```
4 P4599 --> <60,52,43,46> <62,53,45,45> <64,55,46,43>
3 P4600 --> <65,59,53,38> <64,60,55,40> <64,60,55,41> P4552
6 P4601 --> <60,52,48,45> <60,53,48,45> <60,55,48,40>
4 P4602 --> <55,48,43,40> <55,48,43,38> <53,50,43,36>
11 P4603 --> <57,52,48,45> P4530
9 P4604 --> <69,60,53,41> <69,60,52,41> <67,59,50,43>
4 P4605 --> <64,59,55,45> <66,57,54,50> <66,57,52,48>
3 P4606 --> <62,60,57,53> <62,59,56,-13> <60,59,57,-13> <60,57,48,52> <59,57,53,50>
4 E4607 --> <64,59,55,43>
1 E4607 --> <64,60,55,43>
5 P4608 --> <67,60,52,41> <65,59,50,43> E4607 <64,60,55,36>
5 P4609 --> <52,47,44,40> <52,47,44,38> <57,52,45,36>
6 P4610 --> <60,57,52,48> <60,57,52,50> <59,56,52,52>
4 P4611 --> P4571 <53,48,45,33> P4554 <50,48,45,29> <50,47,43,31> <52,48,43,36>
<59,50,44,35> P4593 <59,52,45,40> <57,52,45,40> <59,52,44,28> <59,50,44,28>
<57,48,45,33>
6 P4612 --> <62,57,53,41> <62,57,53,43> <60,57,52,45>
5 P4613 --> <62,55,47,43> <62,55,48,45> <62,55,50,47>
14 P4614 --> <50,47,41,31> <50,47,40,31>
4 P4615 --> <64,61,57,45> <64,61,55,45> <62,57,53,38>
3 P4616 --> <53,52,45,38> <53,50,45,41> <53,50,45,43> <52,49,45,45> <60,50,45,42>
<59,50,43,43> <60,50,43,43>
4 P4617 --> <57,48,43,40> <59,53,41,38> <59,53,43,38> <60,53,45,36> <60,52,43,36>
<60,50,41,36> <60,52,43,36>
3 P4618 --> <57,52,45,36> <57,54,45,36> <59,56,50,35> <60,57,52,33>
4 P4619 --> <50,47,40,32> <48,47,40,33> <48,45,40,33>
4 P4620 --> <65,55,50,48> <67,57,52,49> <65,57,50,50>
3 P4621 --> <57,52,48,33> <57,52,48,35> <57,52,48,36>
3 P4622 --> <69,57,48,41> <69,59,50,41> <67,60,52,40>
8 P4623 --> <60,50,45,42> <59,50,43,43> <60,52,43,36>
3 P4624 --> <59,56,50,35> <59,56,47,40> <59,52,52,40> <57,52,48,33>
5 E4625 --> <55,50,47,43>
5 E4625 --> <55,50,47,31>
10 P4626 --> <57,55,52,36> P4511 E4625
3 E4627 --> <60,57,50,42>
2 E4627 --> <57,54,50,38>
5 P4628 --> <62,57,45,42> <62,55,45,40> E4627 <59,54,50,38>
3 P4629 --> <60,57,52,48> <60,56,52,50> <59,56,52,52>
2 E4630 --> <57,57,53,40>
3 E4630 --> <59,57,50,35>
2 E4630 --> <59,57,50,38>
7 P4631 --> <59,57,53,38> E4630 <59,56,47,40> <59,56,50,40> <57,52,48,33>
9 P4632 --> <60,57,45,41> <60,55,45,40>
3 P4633 --> <53,45,41,38> <53,47,41,38> <52,48,43,40>
5 P4634 --> <64,60,55,40> <64,60,55,41> <62,59,55,43>
1 E4635 --> <64,59,52,44>
2 E4635 --> P4552
3 P4636 --> <65,60,53,38> <65,59,53,38> P4556 <62,59,55,43> <-13,-13,-13,-13> E4635
<60,59,52,45> <60,57,52,45>
4 P4637 --> <66,63,59,35> <66,63,57,35>
16 P4638 --> <59,55,50,43> <59,55,50,41>
6 E4639 --> <52,48,43,40>
6 E4639 --> <50,47,43,40>
12 P4640 --> <50,47,43,41> E4639
9 P4641 --> <60,57,52,36> <60,57,52,38>
1 E4642 --> <60,53,43,36>
2 E4642 --> <59,55,50,38>
3 P4643 --> <57,52,48,40> <59,53,50,38> E4642 <60,52,43,36>
6 P4644 --> <59,55,52,43> <59,53,50,43> <60,52,48,45>
3 P4645 --> <64,60,57,45> <64,60,57,43> <67,60,55,41> <67,60,55,40> <65,60,57,38>
3 P4646 --> <59,52,43,40> <59,50,43,40> <57,48,45,41> <57,50,47,41> <55,52,48,36>
<55,50,48,35>
3 P4647 --> <48,40,36,31> <48,40,36,29>
3 E4648 --> <65,57,48,41>
2 E4648 --> <67,55,48,41>
5 P4649 --> <64,57,48,45> <65,57,48,43> E4648 <67,55,48,40>
3 P4650 --> <57,54,50,36> <59,55,50,35> <60,55,52,33>
3 E4651 --> <62,50,53,47>
1 E4651 --> <62,50,54,47>
4 P4652 --> <60,52,48,45> <62,50,55,47> E4651 <64,55,52,48>
4 P4653 --> <50,48,45,29> <50,47,41,26> <50,45,41,26>
```

```
3 P4654 --> <62,57,48,41> <62,57,48,40> <62,55,47,41>
3 P4655 --> <57,53,48,41> <57,53,48,45> <55,52,48,48>
4 E4656 --> <52,48,43,38>
1 E4656 --> <50,47,41,38>
5 P4657 --> P4640 E4656 <52,48,43,36>
3 P4658 --> <55,52,48,40> <57,52,48,42> <57,50,50,42>
3 P4659 --> <66,57,51,35> <64,56,47,40> <64,59,52,44> <60,57,52,45>
7 P4660 --> <59,53,45,38> <59,52,44,40> <57,52,45,41>
4 P4661 --> P4660 <57,50,45,41> <56,50,47,35> <57,52,48,33>
6 P4662 --> <60,55,48,40> <62,57,48,41> P4569
3 P4663 --> <52,45,40,36> <52,44,40,36> <50,45,33,30> P4619
2 E4664 --> <65,62,55,47>
1 E4664 --> <65,59,55,43>
3 P4665 --> <65,60,57,47> <65,60,57,45> <67,62,55,47> E4664 <64,60,55,48>
1 E4666 --> <69,58,53,43>
3 E4666 --> <69,60,53,43>
4 P4667 --> <67,60,52,48> <67,60,52,46> <69,60,53,45> E4666 <69,60,53,41>
3 E4668 --> <64,57,49,45>
1 E4668 --> <62,53,50,45>
4 P4669 --> <64,58,50,43> E4668 <64,55,49,45> <62,53,45,38>
8 P4670 --> <60,55,48,40> P4510 <64,60,55,36>
3 E4671 --> <59,52,44,40>
1 E4671 --> <59,47,44,40>
4 P4672 --> <59,53,45,38> <59,50,45,35> E4671
11 P4673 --> <65,60,48,45> <65,60,50,45>
2 E4674 --> <65,60,57,41>
1 E4674 --> <64,60,55,43>
3 P4675 --> <60,57,52,45> <60,59,52,45> E4674 <62,60,55,43>
2 E4676 --> <50,48,41,43>
1 E4676 --> <50,48,40,43>
3 P4677 --> <50,48,43,43> E4676 <50,47,41,31> <48,43,40,36>
6 P4678 --> <57,48,45,41> <57,50,47,41> <55,52,48,40>
4 P4679 --> <62,57,48,41> <64,57,48,41> <65,57,48,41>
5 P4680 --> <57,52,43,36> <57,50,42,38> <57,48,42,38>
8 P4681 --> <60,52,48,45> <60,52,47,43> <62,57,45,41>
3 P4682 --> <60,53,48,33> <60,53,48,35> <55,52,48,36> <55,52,46,36>
3 P4683 --> <61,57,52,45> <61,57,52,43> <62,57,53,41> <62,57,53,40> <64,57,52,38>
<64,57,52,37>
3 P4684 --> <48,45,40,29> <50,47,41,28> <50,47,41,26> <52,48,43,24>
8 P4685 --> <59,55,50,31> <60,55,52,36> <59,55,50,43>
3 P4686 --> <55,48,43,28> <53,48,45,26> <53,50,45,26>
4 E4687 --> <50,48,45,29>
1 E4687 --> <50,48,45,30>
5 P4688 --> <53,48,45,33> P4554 E4687
9 P4689 --> <60,55,52,48> <60,55,52,47> <60,57,52,45>
6 P4690 --> <57,53,48,41> <59,53,50,40> <59,53,50,38>
3 P4691 --> <55,50,47,41> <60,55,48,40> <62,57,41,38> <62,59,41,38> <64,60,43,36>
3 P4692 --> <62,59,47,44> <60,59,48,45> <60,57,52,45>
6 P4693 --> <62,55,50,47> <64,55,50,48> <64,55,48,48>
4 P4694 --> <62,55,47,43> <60,55,48,45> <60,54,48,45> <59,55,50,47> <59,55,50,48>
P4524 P4519 <64,55,47,48>
6 P4695 --> <57,52,43,33> <57,50,42,38> <55,50,47,31>
1 E4696 --> <57,54,50,38>
2 E4696 --> <57,54,50,50>
3 P4697 --> <59,55,50,43> E4696 <59,56,50,50> P4596
14 P4698 --> <53,50,43,35> <52,48,43,36>
3 P4699 --> <65,57,48,45> <64,57,48,43> <62,57,48,41> <60,55,47,43>
4 P4700 --> <55,52,45,37> <53,52,45,38> <53,50,45,38> <52,49,45,33>
2 E4701 --> <67,59,50,47>
2 E4701 --> <67,55,50,47>
4 P4702 --> P4605 E4701 <64,55,48,48> <64,55,50,47> <69,60,52,45>
3 P4703 --> <59,52,43,36> <60,52,43,36> <59,50,43,38>
4 P4704 --> <48,45,40,33> <47,45,41,26> P4544
1 E4705 --> <57,52,45,42>
3 E4705 --> <57,52,45,40>
4 P4706 --> <57,52,48,33> <57,52,47,43> <57,50,45,42> E4705
3 P4707 --> <55,48,43,40> <57,48,48,41> <57,50,48,41> P4562 P4685 <57,53,50,38>
<57,52,50,40> <62,50,45,42> <62,52,47,44> <60,52,48,45> <59,54,50,45> P4543
<59,56,50,40> <59,54,48,40> <59,56,50,40> <59,56,48,40> <57,52,48,33>
3 E4708 --> <60,55,50,40>
3 E4708 --> <59,55,50,40>
```

```
6 P4709 --> <60,57,52,45> <60,55,52,40> E4708
4 P4710 --> <55,52,48,36> <55,50,47,36> <57,48,45,41>
3 P4711 --> <55,47,43,40> <55,48,43,40> <55,50,43,35>
3 P4712 --> P4556 <62,60,53,43> P4565 <60,55,52,36>
6 P4713 --> <55,48,43,40> <57,48,41,41> P4533 <60,50,45,41> <60,52,43,43>
<60,53,43,43>
3 P4714 --> <59,55,43,40> <59,53,43,38> <60,52,43,36>
6 P4715 --> <55,48,43,40> <55,48,40,36> <57,48,41,41>
3 E4716 --> <57,52,48,45>
1 E4716 --> <57,52,48,33>
4 P4717 --> <64,59,56,40> <64,59,52,44> <60,57,52,45> P4519 <64,55,50,48>
<62,57,52,45> <60,57,52,45> P4500 E4716
4 P4718 --> <60,57,52,45> P4500 <57,57,48,41>
4 P4719 --> <57,48,43,29> <57,48,41,33> <57,48,41,35>
3 P4720 --> <48,45,45,29> <50,47,45,29> <52,48,43,28> <52,48,43,29> <50,48,43,31>
<50,48,41,31> <50,47,41,31> <48,43,40,24>
1 E4721 --> <60,55,52,36>
3 E4721 --> <60,52,43,36>
4 P4722 --> <62,55,48,40> <62,57,48,41> <60,55,47,43> E4721
2 E4723 --> <59,53,50,38>
2 E4723 --> <57,53,47,38>
4 P4724 --> <57,48,45,41> <57,48,45,40> E4723 <57,53,48,38>
3 P4725 --> <67,55,48,40> <67,57,48,41> <62,59,55,43>
5 P4726 --> <55,48,41,38> P4715
3 P4727 --> <48,45,40,33> <50,43,38,35> <52,43,36,36> <53,43,38,36>
4 P4728 --> <59,52,44,28> <59,50,44,28> <57,48,40,33>
1 E4729 --> <62,60,55,48>
2 E4729 --> P4665
2 E4729 --> <62,60,55,41>
5 P4730 --> <64,60,55,48> E4729 <62,60,57,41> P4507
1 E4731 --> <60,52,52,36>
2 E4731 --> <57,54,52,38>
3 P4732 --> <59,55,52,36> E4731 P4511 <55,50,47,31>
2 E4733 --> <60,55,45,40>
2 E4733 --> <62,55,45,40>
4 P4734 --> <62,57,45,41> E4733 P4591 <57,48,45,33>
2 E4735 --> P4679
2 E4735 --> <62,57,48,42>
4 P4736 --> <57,53,48,41> <59,53,48,41> P4594 E4735 P4502
3 P4737 --> <62,55,48,43> <60,55,47,43> <62,55,47,43> <60,55,52,36>
2 E4738 --> P4694
1 E4738 --> <64,55,47,48>
3 P4739 --> P4519 E4738 <65,53,48,45> <64,55,48,43> <64,55,48,41>
2 E4740 --> <62,60,53,38>
1 E4740 --> <62,60,53,41>
3 P4741 --> <62,60,57,41> E4740 <62,59,50,43> <62,59,53,43> <60,55,52,36>
4 P4742 --> <52,48,43,28> <52,48,43,29> <50,47,43,31>
4 P4743 --> <53,48,45,33> <53,48,45,35> <52,48,43,36>
3 P4744 --> <60,52,48,45> <62,50,55,47> <64,48,55,48>
2 E4745 --> <53,48,43,31>
1 E4745 --> <53,48,45,26>
3 P4746 --> P4686 E4745 <53,47,43,31> <52,48,43,24>
2 E4747 --> <64,60,55,48>
1 E4747 --> <62,60,55,43>
3 P4748 --> <65,60,57,45> <65,62,55,47> <64,62,55,48> E4747 <62,59,55,43>
3 P4749 --> <55,48,40,36> <55,48,38,35> P4533
1 E4750 --> <64,60,57,45>
2 E4750 --> <64,60,55,40>
3 P4751 --> <65,62,55,47> <64,62,55,48> E4750 <62,60,57,41>
3 P4752 --> <57,48,42,38> <57,47,42,38> <55,47,38,31>
6 P4753 --> <57,53,48,41> <57,53,48,40> <57,53,50,38>
```

Appendix C

Transition Matrices

This is an example of some of the transition matrices after analysis of 250 Bach Chorales. Each matrix represents a 1-order Part Matrix, which contain pitch transitions. Note that as opposed to probabilities, the transition numbers represent the number of instances read in the input files.

Part Matrix: Soprano_Major

n-gram	A#4	A#5	A3	A4	A5	B3	B4	B5	C#4	C#5	C4	C5	C6	D#5	D4	D5	E4	E5	F#4	F#5	F4	F5	G#4	G#5	G3	G4	G5	REST-1
A#4	0	0	0	20	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
A#5	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A3	0	0	1	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
A4	11	0	0	80	1	0	195	0	0	0	0	9	0	0	4	11	12	5	0	0	9	0	0	6	0	262	0	6
A5	0	2	0	0	14	0	0	11	0	0	0	0	0	0	0	3	0	1	0	1	0	8	0	0	0	0	54	0
B3	0	0	1	0	0	1	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B4	0	0	0	208	1	0	50	0	0	0	0	246	0	0	0	2	5	5	0	0	0	0	0	0	0	38	0	5
B5	0	0	0	0	8	0	0	0	0	0	0	0	10	0	2	0	0	0	0	0	0	0	0	3	0	0	1	0
C#4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C#5	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C4	0	0	2	1	0	8	0	0	0	0	42	9	0	0	59	11	0	0	0	0	11	0	0	0	0	48	0	4
C5	8	0	0	37	3	0	307	6	0	0	8	257	0	0	0	359	5	69	0	0	2	17	0	0	0	47	15	9
C6	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	5	0	0	0	0	0	0	0	0	3	0
D#5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D4	0	0	0	0	0	0	1	0	0	0	95	0	0	1	37	0	122	0	0	0	2	0	0	0	0	26	0	1
D5	3	0	0	4	0	0	11	0	0	0	9	328	0	0	0	165	0	447	0	0	16	0	0	0	0	10	20	5
E4	0	0	1	35	0	0	9	0	0	0	16	12	0	0	198	0	26	2	0	27	0	0	0	0	0	27	0	9
E5	0	0	0	6	17	0	0	0	0	2	0	100	0	1	0	463	0	159	0	0	19	293	0	0	0	1	31	2
F#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	40	0	0
F#5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33	0
F4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	15	0	210	0	0	0	18	0	0	0	0	69	0	2
F5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	29	0	325	0	0	0	32	0	0	0	0	168	1
G#4	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	1	0	0	0	0	0
G#5	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
G4	3	0	0	203	46	0	4	2	0	0	13	115	3	0	5	11	62	4	15	13	161	190	1	0	0	199	8	4
G5	0	0	0	0	1	0	0	0	0	0	0	16	0	0	0	25	0	40	0	0	0	0	0	0	0	0	113	1
REST-1	0	0	0	6	1	0	4	0	0	0	0	4	0	0	4	3	0	5	0	0	0	0	0	0	0	14	2	0

Figure C.1: First order, Major mode Soprano Matrix after analysing 250 Bach Chorales

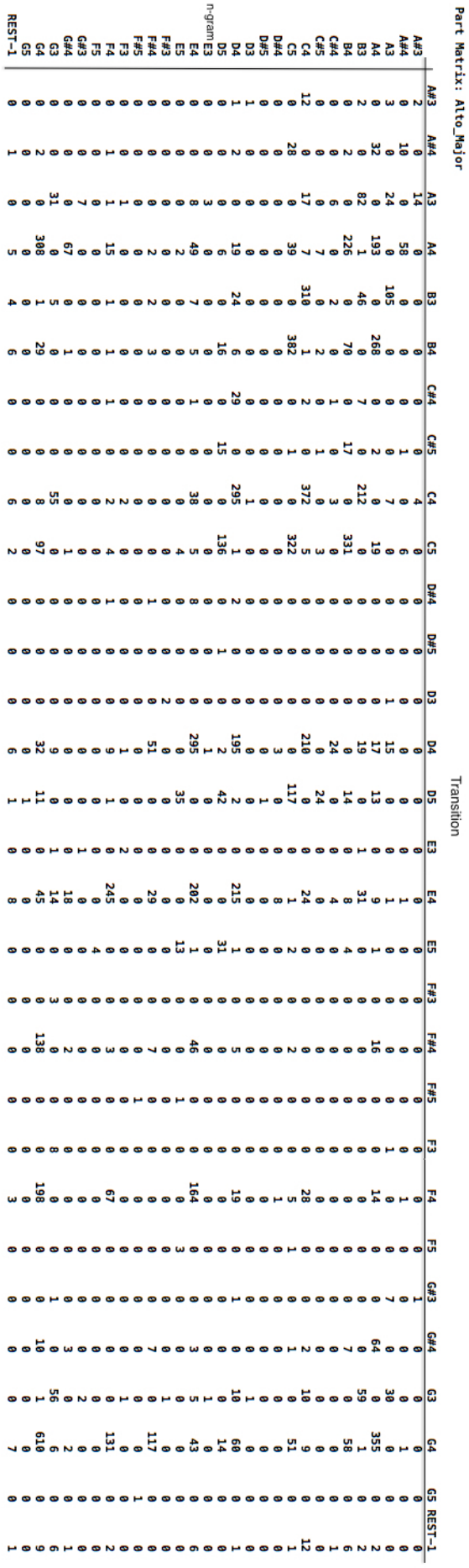


Figure C.2: First order, Major mode Alto Matrix after analysing 250 Bach Chorales

Part Matrix: Tenor_Major

n-gram \ Transition	A#3	A#4	A2	A3	A4	B2	B3	B4	C#3	C#4	C3	C4	C5	D#3	D#4	D3	D4	D5	E3	E4	F#3	F#4	F3	F4	G#3	G#4	G3	G4	REST-1
A#3	6	0	0	58	0	0	3	0	0	0	0	16	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
A#4	0	1	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
A2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A3	20	0	0	148	3	0	160	0	0	0	23	0	0	0	0	20	52	0	11	13	6	0	24	2	32	0	251	0	5
A4	0	10	0	1	37	0	0	16	0	0	0	2	1	0	0	0	33	1	1	9	0	2	0	17	0	5	0	142	1
B2	0	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	2
B3	1	0	155	0	1	1	55	3	0	13	0	331	0	0	0	1	15	0	18	67	0	1	0	0	0	0	52	9	3
B4	0	1	0	0	11	0	0	2	0	0	0	0	4	0	0	0	0	0	0	5	0	0	0	0	0	0	0	3	0
C#3	0	0	0	6	1	0	1	0	0	0	0	8	0	0	0	4	0	0	0	0	0	1	0	0	0	0	0	0	0
C#4	0	0	0	6	0	5	0	0	0	0	6	3	0	0	0	24	0	0	8	3	0	4	0	0	0	0	12	0	0
C3	0	0	0	31	9	0	380	0	1	3	358	1	0	0	0	0	297	0	1	37	2	0	11	41	0	0	53	37	5
C4	53	0	0	0	0	0	0	3	0	0	1	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	3	0
C5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
D#3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D3	0	0	0	4	0	0	3	0	0	0	45	1	0	0	0	11	1	0	60	5	0	0	4	0	0	0	49	0	2
D4	0	0	0	12	11	0	57	1	0	24	0	356	0	0	0	0	216	0	0	278	0	3	0	9	3	0	48	74	5
D5	0	0	0	0	0	0	9	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
E3	1	0	1	43	0	0	7	0	0	0	5	9	0	0	0	77	0	0	68	8	23	0	136	0	0	0	40	0	6
E4	0	0	0	27	28	0	23	0	0	0	0	52	1	0	0	0	355	0	1	213	0	27	0	219	0	0	1	45	5
F#3	0	0	0	0	0	0	14	0	0	0	0	1	0	0	0	18	0	0	10	0	0	2	0	0	5	0	53	0	0
F#4	0	0	0	7	0	0	8	0	0	0	4	1	0	0	1	20	9	0	163	17	0	16	0	53	0	0	167	33	4
F3	2	0	0	1	4	0	15	0	0	0	0	11	0	0	2	0	39	0	7	234	0	3	0	0	0	0	0	180	0
F4	0	0	0	36	5	0	2	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1
G#3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
G#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
G3	3	0	229	0	154	0	24	0	0	0	5	71	0	0	2	10	22	0	46	15	69	30	226	0	6	3	397	2	0
G4	0	0	0	0	0	0	5	1	0	3	0	32	1	0	1	0	33	0	0	44	0	0	0	242	0	0	14	356	4
REST-1	0	0	0	4	0	0	0	0	0	1	1	13	0	0	1	1	3	0	1	4	0	2	0	0	0	1	14	4	2

Figure C.3: First order, Major mode Tenor Matrix after analysing 250 Bach Chorales

Part Matrix: Bass_Major

n-gram	A#2	A#3	A2	A3	B2	B3	C#3	C#4	C2	C3	C4	D#2	D#3	D#4	D2	D3	D4	E2	E3	E4	F#2	F#3	F#4	F2	F3	F4	G#2	G#3	G1	G2	G3	G4	REST-1
A#2	1	1	23	2	2	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
A#3	0	2	0	39	0	2	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A2	8	0	17	21	146	1	0	0	0	15	1	2	0	0	15	65	0	0	24	4	1	2	0	0	2	0	0	0	0	88	3	2	2
A3	0	14	9	34	1	192	1	1	0	1	13	0	0	0	0	46	17	0	65	4	0	15	0	0	37	0	12	29	0	0	308	1	6
B2	3	0	102	0	2	0	11	1	1	225	0	0	0	0	0	0	1	0	20	0	0	2	0	0	0	1	0	1	0	39	0	0	6
B3	0	4	0	194	0	0	0	0	0	0	207	0	0	0	0	0	0	0	3	6	0	0	0	0	0	0	0	1	0	0	31	0	0
C#3	0	0	7	1	0	0	1	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C#4	0	0	1	0	3	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0	0	1	0	3	0	1	0	0	31	2	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	3
C3	19	0	61	3	184	4	9	0	5	118	105	0	0	0	0	274	82	11	50	24	5	1	0	39	135	53	0	1	1	61	79	58	14
C4	0	23	0	55	0	224	2	17	0	21	33	0	0	0	0	0	0	0	24	1	0	9	0	0	0	0	0	0	0	0	0	3	1
D#2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D#3	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	3	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
D2	0	0	32	21	18	2	18	9	6	207	0	8	8	0	16	10	10	28	0	0	0	0	0	2	7	0	0	0	0	38	132	26	0
D3	0	0	29	11	0	6	0	3	0	0	57	0	0	2	0	23	0	0	303	44	0	0	0	0	0	0	0	7	0	86	0	1	0
D4	0	0	43	109	14	0	4	1	0	90	12	0	0	17	0	284	34	11	21	2	1	59	19	0	310	13	0	2	0	1	15	15	2
E2	0	0	0	9	0	2	0	0	0	0	0	0	0	0	0	1	0	0	3	3	0	0	0	0	1	0	0	0	0	1	0	0	0
E3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F#2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F#3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2	0	0	1	0	3	0	0	0	1	9	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F3	0	0	0	0	0	0	0	0	0	38	21	0	0	0	0	59	1	0	320	0	0	19	0	0	22	4	0	0	0	0	0	0	0
F4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	1	0	0	0	0	0	0	0	0
G#2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G#3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G2	1	0	91	2	12	0	0	0	35	146	3	1	0	0	3	12	1	10	59	0	0	0	0	0	4	2	0	0	0	0	0	0	0
G3	0	0	2	188	0	11	1	1	0	204	107	0	1	0	12	35	8	10	0	1	0	0	0	0	0	4	0	0	0	0	0	0	0
G4	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
REST-1	0	0	1	2	4	3	0	0	0	5	2	0	0	0	0	1	0	1	12	0	0	1	0	0	7	0	1	2	0	4	3	0	3

Transition

Figure C.4: First order, Major mode Bass Matrix after analysing 250 Bach Chorales

Bibliography

[1] Baggi, D; Haus, G, Encoding Music Information, *Music Navigation with Symbols and Layers:Toward Content Browsing with IEEE 1599 XML Encoding*. pp 21-37, Wiley-IEEE Computer Society Press, USA, 2013.

[2] Baggi, D; Haus, G, The IEEE 1599 Standard, *Music Navigation with Symbols and Layers:Toward Content Browsing with IEEE 1599 XML Encoding*. pp 2-20, Wiley-IEEE Computer Society Press, USA, 2013.

[3] Bernstein, S, Sur l'extension du theoreme limite du calcul des probabilies, *Math. Annalen*. 97 pp 1-59, Springer, Germany, 1926.

[4] Brin, S; Page, L, The anatomy of a large-scale hypertextual Web search engine, *Computer Networks and ISDN Systems*. 30 pp 107-117, Stanford, USA, 1998.

[5] Brooks, F; Hopkins, A; Neumann, P; Wright, W, An experiment in musical composition, *Machine models of music*. MIT Press, USA, 1993.

[6] Chomsky, N, Three models for the description of language, *IRE Transactions on Information Theory*. 2 pp 113124, Institute of Electrical and Electronics Engineers, USA, 1956.

[7] Cope, D, *Computers and Musical Style*. Oxford University Press, UK, 1991.

[8] Cope, D, *Computer Models of Musical Creativity*. MIT Press, USA, 2004.

[9] Dawes, Beman, *Proposal for a C++ Library Repository Web Site* Accessed Online, August 2014, "http://www.boost.org/users/proposal.pdf, 1998.

[10] Fanselow, G; Felix, S, *Sprachtheorie: Eine Einführung in die generative Grammatik*. Francke, Germany, 1956.

[11] Guido, Smits Van Waesberghe, J, *Micrologus*. American Institute of Musicology, Netherlands, 1955.

[12] Hild, H; Feulner, J; Menzel, W, HARMONET: a neural net for harmonizing chorales in the style of J.S. Bach, *Advances in Neural Information Processing Systems*. 4, MIT Press, USA, 1992.

[13] Horner, A; Goldberg, D, Genetic algorithms and computer-assisted music composition, *Proceedings of the 1991 International Computer Music Conference*. International Computer Music Association, USA, 1991.

[14] Huber, D, *The MIDI Manual: A Practical Guide to MIDI in the Project Studio*. Elvesier, USA, 2007.

[15] Johnson-Laird, PN, Jazz Improvisation:a theory at the computational level, *Representing musical structure*. Academic Press, UK, 1991.

[16] Kogan, R, *A Brief History of Electronic and Computer Musical Instruments*. Accessed Online, June 2014, "http://www.math.tamu.edu/ romwell/electron-music/electron-music.pdf".

[17] Mandelbrot, B, The Fractal Universe, *The Origins of Creativity*. pp 191-212, Oxford University Press, UK, 2001.

[18] Markov, A, Extension of the law of large numbers to dependent quantities, *Izvestiia Fiz.-Matem*. 15 pp 135-156, Kazan University Press, Russia, 1906.

[19] *Webster's Seventh New Collegiate Dictionary*. Meriam, USA, 1976.

[20] Meyer, L, *Style and Music: Theory, History, and Ideology*. University of Chicago Press, USA, 1996.

[21] Nevill-Manning, C; Witten, I, Identifying hierarchical structure in sequences: A linear-time algorithm, *Journal of Artificial Intelligence Research*. 7 pp 6782, AAAI Press, USA, 1997.

[22] Nierhaus, G, *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer-WienNewYork, USA, 2009.

[23] Pardo, B; Birmingham, W, Automated partitioning of tonal music, *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*. pp 23-27, AAAI Press, USA, 2000.

[24] Recordare Inc, *The MusicXML Library*. Accessed Online, July 2014, "http://libmusicxml.sourceforge.net".

[25] Rothstein, J, *MIDI: A Comprehensive Introduction*. Oxford University Press, UK, 1992.

[26] Shacham, O; Hermesh, O; Mendiuk, K; Sandbank, B, *Automatic Distillation of Musical Structures:Learning the Grammar of Music*. School of Computer Sciences,Tel Aviv University, Tel Aviv, Israel, 2005.

[27] Shaney, V, *Party Politics (follow-up)*. net.singles. Google Groups Usenet archive, 1984.

[28] Shaobohou, *My implementation of the ADIOS algorithm*. Accessed Online, July 2014, "https://github.com/shaobohou/madios".

[29] Simpson, J, *The Oxford English Dictionary*. Clarendon Press, UK, 1989.

[30] Solan, Z; Horn, D; Ruppin, E; Edelman, S Unsupervised learning of natural languages, *Proceedings of the National Academy of Science*. 102 pp11629-11634, Tel Aviv University Press, Israel, 2005.

[31] Steedman, M, A generative grammar for jazz chord sequences, *Music Perception*. 2, University of California Press, USA, 1984.

[32] Todd, P, A connectionist approach to algorithmic composition, *Computer Music Journal*. 13/4, MIT Press, USA, 1989.

[33] Walshaw, C, *A Brief History of ABC*. Online, Accessed June 2014, "http://abcnotation.com/history".

[34] Walshaw, C, *ABC Notation*. Online, Accessed June 2014, "http://abcnotation.com/" .

[35] Xenakis, I, Three models for the description of language, *Formalized music: Thought and mathematics in music*. Pendgragon, USA, 1992.