Computer-assisted music composition – A database-backed algorithmic composition system

Nikolas Borrel-Jensen
mail@nikolasborrel.com

Andreas Hjortgaard Danielsen
andreas@hjortgaard.net

Department of Computer Science,
University of Copenhagen

Copenhagen, January 25, 2011

Supervisor: Jakob Grue Simonsen (simonsen@diku.dk)

Abstract

A database of 86 melodic rock tunes and 43 standard jazz tunes has been created. Each tune is divided into parts corresponding to different sections of the music (e.g. verse and chorus) stored in MIDI format.

Three algorithms for learning musical features and generating new material have been developed: A simple model for generating melodies using two Markov chains; one with notes as the state space and one with note durations as state space. The second algorithm is a Hidden Markov Model that is used to harmonise a given melody in the style of a given genre. The third algorithm is a neural network that learns melodic variations and can vary an input melody in accordance with the database and given chord progressions.

As a follow-up to the implementation, the system has been validated and evaluated by five different musicians in two user tests. The first one was a prototype test of the melody generator and an early version of the harmoniser. From the second test - a test of the entire system - we could conclude that the harmoniser was able to generate style-specific harmonisations that in some cases inspired the musicians to make new compositions, but it can never be used directly as a basis for their original melody. As for the melody generation algorithm, it proved to be too simple to generate coherent and musical melodies but contained inspirational elements.

Keywords: Algorithmic composition, machine learning, Haskell, harmonisation, melodic variation, Markov chains, Hidden Markov Models, Neural Networks

Preface

To be able to learn a computer how to compose music, it is appropriate to investigate how a human is composing music in general. We have not delved deeply into this field of psychology, but we will try to give our view of the creative process of composing music. In our opinion, new music is created by expanding and developing the music already present. It should not necessarily be understood in the sense that a composer chooses a piece of music, alter it and a new, unique piece of music has been created. It should be understood as a person having encountered a lot of music which he has latent in his memory, and when composing a new song he uses this more or less unconsciously as a foundation for new music creation.

We can distinguish between internal and external sources of inspiration. When a composer hears music that he likes, it will become part of his memory and we can think of it as an internal source of inspiration. So, how does a composer alter and expand the corpus of music stuck in his memory? This is the tricky point and probably many aspects play a role. Some people go for a stroll in the woods and is suddenly hit by a branch that sparks a creative musical process; others hear a train screeching giving ideas for a high pitched guitar riff[1]. Such events can be thought of as external sources of inspiration.

What we have done to model this idea of the creative process of composing music is to 1) build a database of music to represent the internal sources of inspiration and 2) build a system that takes some input from the user, which represents the external source (e.g. the branch), and alter it by mingling it with features extracted from the database. We can compare the database with a composer's memory of encountered music. Composers often have an abstract notion of the style of the new composition before any notes have been written down. This style will inherently be inspired by something he has heard before and to some extent, we can

[1]A riff is an ostinato figure: a repeated chord progression, pattern, refrain or melodic figure, often played by the rhythm section instruments or solo instrument, that forms the basis or accompaniment of a musical composition. (http://en.wikipedia.org/wiki/Riff (Tue Dec 29 16:04:01 CET 2009)).

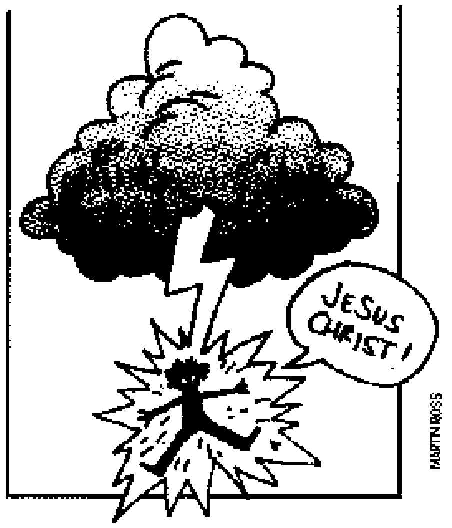


Figure 1: Another example of an external source of inspiration

Composition by humans	Composition by Cremo
Songs stored in the composer's memory (internal sources)	Database of 129 songs
Songs categorised as good or bad	Songs in the database can to some extend be selected by the user
Inspiration from external sources	The user feeds Cremo with a melody line

Table 1: Comparison of the human way of composing and composing by Cremo

simulate this kind of style-decision in our system by making it possible to choose which artists to have as corpus for each new composition. Of course, the number of tunes are very limited compared to the real world, and the subtleties of individual's liking or disliking of some parts of a tune is difficult to do precisely for a large database of tunes.

We have named the project *Cremo*, a short hand for *creative modelling*, because of the way we model the creative process of composing. The comparison is depicted in Table 1. By modelling our composition tool with these thoughts in mind, we hope to capture just a little bit of musical creativity that will inspire composers to create new music.

Acknowledgements

We would like to thank all the people who have helped us in the course of this project. Thanks should go to Paul Liu from the Yale Haskell Group at the University of Yale, developer in the Euterpea project, who was quick to answer all our questions and fixed several bugs in Euterpea, which made this project possible. Thanks to Christian Walder from the Institute of Mathematical Modelling at the Technical University of Denmark for taking the time to explain his article on chorale harmonisation using SVMs.

Big thanks go to all of our test persons: Ras Tind Nielsen, Lauritz Carlsen, Lasse Hansen, Christian Munck and Nils Bouchet for taking the time to test our system. And to Martin Dybdal for proof reading this report.

Finally, the greatest thanks should go to the greatest: Our supervisor Jakob Grue Simonsen for his open-mindedness and excellent counselling (even though he never brought us cake).

Contents

1 Introduction **6**
1.1 Goals . . . 6
1.2 Related work . . . 7
1.3 Expectations of the reader . . . 8
1.4 Overview . . . 9

2 Background **10**
2.1 Machine learning . . . 10
2.2 Harmonic theory . . . 11
2.3 Mathematical musical notation . . . 13

3 Haskell and libraries **16**
3.1 Why Haskell? . . . 16
3.2 Euterpea . . . 17
3.3 Data.MarkovChain . . . 20
3.4 Data.HMM . . . 21
3.5 HFANN . . . 21

4 Database **23**
4.1 Structure of the database . . . 23
4.2 Structure of the tunes . . . 24
4.3 Structure of the MIDI files . . . 25
4.4 Representation of the database in Haskell . . . 25

5 Melodic generation **29**
5.1 Markov chains . . . 29
5.2 Melodic features . . . 30
5.3 Markov chain music . . . 30
5.4 Implementation details . . . 31

6 Harmonisation **33**
6.1 Hidden Markov Models . . . 33
6.2 Harmonic features . . . 36
6.3 Implementation of harmonic features . . . 39
6.4 Harmonising using HMM . . . 41
6.5 Implementation details . . . 43

7 Prototype user test **46**
7.1 Goals of test . . . 46
7.2 The test . . . 47
7.3 Results . . . 48
7.4 Conclusion and further work . . . 51

8 Harmonisation - final version **55**
8.1 Melodic features 55
8.2 Nearest neighbour 56
8.3 Implementation details 57

9 Melodic variation **59**
9.1 Neural networks 59
9.2 Agglomerative hierarchical clustering 64
9.3 Learning melodic variation 65
9.4 Implementation details 70

10 Final user test **76**
10.1 Goals of the test 76
10.2 The test 76
10.3 Results 76
10.4 Conclusion 80

11 Conclusion **81**
11.1 Further work 82

A Contents of the database **83**
A.1 Melodic rock 83
A.2 Standard jazz 84

B Prototype user test **86**
B.1 Protocol 86
B.2 Results 88

C Final user test **92**
C.1 Protocol 92
C.2 Results 94

1 Introduction

This report describes a program for computer-aided music composition, Cremo, in which the computer generates melodic lines and harmonisations and variations of input melodies under certain assumptions by searching for and deriving patterns in a database of music. The system is supposed to be a help for the composing musician who has come across a composer's block or who just want an example of how his music could sound if it was rewritten in a different genre. Many algorithms have been made that generates new music from a set of rules (see Section 1.2 on related work) but since music composition is a much more complex and subjective endurance than a set of rules can model, we have chosen to base this project on machine learning techniques that learns from a large database of music. We believe that this approach is more similar to the way human composers make music: Musicians will often have a large database of music (for example a CD collection) that they have listened to often. When composing new music, the composer will have a lot of melodies in his head that he can reference, thereby remembering chord progressions or melodic features that he can use for his own composition.

Following the same strategy we want Cremo to model this creative process by learning musical features from a database of music in different genres. This way we do not have to incorporate a large set of rules for each genre and we can settle with some parameter adjustments when optimising our learning algorithm for new additions to the database.

Since good music to a large degree is a question of taste and subjective opinion, we can not quantatively measure the quality of Cremo, i.e. whether it creates good or at least usable music. So in order to validate our efforts we have conducted interviews and user tests with several musicians, having a sound knowledge in music theory and who composes music themselves. This way we can at least get educated opinions about the results we produce.

1.1 Goals

The goal of this project is to create a system that helps the composer when composing new pieces by creating inspirational music from a database of tunes in different genres. We therefore assume that the composer has already composed some of his piece, for example a short melody. The goal of Cremo then, is to take the input melody and do the following:

- Harmonise the melody according to a given genre
- Alter the melody according to a given genre
- Develop new melodic content according to a given genre

The harmonisation is done by placing chords at fixed intervals along the melody. These chords have to fit the melody, of course, but also need to have musical direction and obey tension/release rules. The altered melody has to sound similar to the original input melody, but it also has to be varied in the style of the chosen genre and fitted to the underlying harmonies. The development of new melodic content is done to give inspiration for new directions in the music, or perhaps to spark ideas for new compositions.

The database needs to be created such that all songs have the same structure and it is easy to expand with new genres. It is important to limit the restrictions as much as possible, so that the program is able to learn new genres simply by adding songs to the database. This restricts us from coding genre-specific rules into the program and database. We have created a simple database system that ensures such scalability. We do have some restrictions on the format of the songs, however, but we feel that these restrictions are general enough to cover most genres within modern rhythmic music.

1.2 Related work

Overviews of the field of algorithmic composition are given in [17] and [18]. We will now give an overview of projects in algorithmic composition that are similar to Cremo, as defined by the goals above. This overview is sectioned as in [17].

Markov Models

In 1957 Lejaren Hiller and Leonard Isaacson created the "Illiac Suite" for string quartet [11], which is considered the first computer-generated composition [17, p. 72]. It consists of four movements, the last of which uses Markov models for note selection.

Esben Skovenborg and Jens Arnspang [22] have made a similar project as Cremo, using a database of 33 international pop songs and 17 Danish traditional songs from which they extract structural patterns using Markov models.

Hidden Markov Models

Hidden Markov Models (HMMs) have been used by Moray Allan [1] to do harmonisation of Bach chorales, where the notes of the soprano voice are represented as the observable states and the underlying harmony is represented as the hidden states. The Viterbi algorithm is used to find the most probable sequence of harmonies given a soprano voice melody.

Generative Grammars and ATNs

Generative grammars are used for musical analysis and style replication. Particularly interesting is the work done by Mark Steedman who described a grammar for the generation of jazz chord sequences [17, p. 106].

Experiments in Musical Intelligence (EMI) by David Cope, described in his trilogy [5],[6] and [7], uses augmented transition networks. EMI generates entire pieces from analysing parts of a large database (for example, looking only at Bach chorales).

Genetic Algorithms

As seen in [17], genetic algorithms are used in many ways in algorithmic composition. George Papadopoulos and Geraint Wiggins [19], [20] use GAs to generate jazz melodies over input chord progressions and to harmonise chorale melodies.

John A. Biles [2] uses GAs to generate jazz solos over chord progressions. He uses a so-called Interactive Genetic Algorithm, which means that the fitness of a population is evaluated by a human being listening to the output of the program. He also uses two populations: One population with phrases that references the other population, which consists of eighth-notes corresponding to one measure. Each note is a number, representing a pitch value or a rest. For each phrase and measure there is a fitness value. Depending on the input from the user, when evaluating populations this value will change.

Cellular Automata

Cellular Automata are different from other methods in that the way they are used is often by mapping cells to musical notes [17, p. 202]. For example Cellular Automata Music by Dale Millen [17, p. 195] maps cells to MIDI values. Eduardo Reck Miranda [17, p. 197] has developed a composition system, called CAMUS, that uses the output of the popular CA *Conway's Game of Life* to generate pitches and durations.

Neural networks

Hörnel and Menzel [14] have made a system that invents a baroquestyle chorale harmonisation and variation of any chorale voice using neural networks.

Neural networks have also been used to melody generation: Michael C. Mozer [16] has developed a system called CONCERT, that uses a recurrent Elman net to analyse and extract stylistic patterns in a corpus of music in a given style and uses the same net to generate new melodies and harmonic accompaniment in the same style. The work of Eck and Scmidhuber [9] describes a long short-term memory recurrent neural network that generates 12-bar blues.

Support Vector Machines

Christian Walder [25] from the Technical University of Denmark has written an unpublished article on harmonising Bach chorales using Support Vector Machines and dynamic programming. The principles follow the same intuition as Moray [1] but using SVMs instead of HMMs, the algorithm becomes more general and can look back several steps instead of just one step. The Viterbi algorithm used to find the best possible harmonisation in the HMM is replaced by a dynamic programming algorithm that finds the optimal harmonisation according to the SVM model.

1.3 Expectations of the reader

Although the machine learning is introduced in Section 2.1 and the methods we use for extracting patterns from the database are described in detail, some basic knowledge of general machine learning techniques and statistics would be helpful. The number one resource on pattern recognition and machine learning is [3]. Especially, we will discuss the use of Markov chains for melodic development, neural networks for melodic variation and Hidden Markov Models for harmonisation. Theoretical introductions are given to these methods in sections 5.1, 6.1 and 9.1.

We expect that the reader has some basic knowledge of music theory, especially melodic and harmonic analysis for example from [21] but we will give a short introduction to the harmonic theory that is used throughout the text in Section 2.2.

1.4 Overview

Section 2 will give short introductions to machine learning and harmonic theory and introduce the notation we will use when discussing musical features in later sections.

Section 3 describes why we have chosen to write the system in the Haskell programming language and it provides a description of the different non-standard libraries we use. Especially, Euterpea is thoroughly described since it is the basis of how we represent music in an abstract form.

Section 4 gives an overview of our database, how it is build up and how we populated it.

Section 5 describes how we extract musical features from the database for the purpose of generating new melodic lines with Markov chains. We focus here on how we choose the features to look for and how we implement it.

Section 6 gives an introduction to Hidden Markov Models and a detailed description of how we extract features from the database for the purpose of stylistic harmonisation of a melodic line in the prototype version of our harmoniser.

Section 7 describes the protocol and results for our prototype user test, which tested the Markov chain melodic generator and the prototype harmoniser.

Section 8 describes the finalised version of the harmonisation model. It describes the new feature extraction and the implementation details.

Section 9 describes the theory of neural networks in detail and describes the multiscale neural network model for melodic variation.

Section 10 describes the final user test, its protocol and results.

Section 11 concludes how well we reached our goals and discuss prospects for future work.

Appendix A contains a list of all the tunes in our database divided into genres and artists.

Appendix B and C show exact formulation of our protocols and lists all results of our two user tests.

A CD-ROM is enclosed with the report containing: The report in PDF format; the source code; the database in MIDI format; a “READ ME” document describing how to compile and run the program; all generated test files from the user tests.

2 Background

As seen from the related work described above in Section 1.2, a lot of research has been done in the field of algorithmic composition, and therefore many different methods and tools have already been investigated, each having their own advantages and drawbacks. Since we want Cremo to extract information from a database and thereby learn how to harmonise and create melodies according to different genres, we believe that a statistical machine learning approach would be the best choice. Here we will go through the basics of machine learning. For a thorough treatment of the field of machine learning, we refer to [3]. We will also give a very short introduction to harmony theory in order to present the notation and terminology we will use in later chapters on harmonic feature extraction. For more on harmony theory and harmonic analysis, we refer to [21] and [4].

2.1 Machine learning

Machine learning is a field that has the purpose of creating machines that learn and recognise patterns. In general, a machine learning problem can be described as follows: Let $\mathbf{x}$ be a vector, called a *feature vector*, and let $\mathbf{t}$ be a vector of the same size as $\mathbf{x}$, called a *target vector*. Given a set of feature vectors, $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ called a *training set* and a corresponding set of target vectors $\mathbf{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n\}$, we want to compute a mapping $y : \mathbf{X} \rightarrow \mathbf{T}$, such that

$$y(\mathbf{x}_i) = \mathbf{t}_i$$

for all i. This mapping can thus be used to take in new feature vectors (called the *test set* or *validation set*) and compute the corresponding target vectors. This is the essence of machine learning. This kind of problem, where the target vector is given in advance, is called a *supervised learning* problem. In such problems we differ between two types of learning problems: When $\mathbf{t}$ is a discrete and finite, we call the problem a *classification* problem, where each value of $\mathbf{t}$ corresponds to a class and each value of $\mathbf{x}$ is mapped to such a class. When $\mathbf{t}$ is continuous we call the problem a *regression* problem.

When no target vector is given in advance, we can use machine learning to compute classifications automatically (called *clustering*) or make *dimensional reduction*. This type of machine learning problems is called *unsupervised learning*.

To compute y we turn to statistical methods, and especially Bayesian statistics [3]. Consider a classification problem where, given an input vector $\mathbf{x}$, we want to compute the probability of $\mathbf{x}$ belonging to a specific class C_k, that is we want to compute $P(C_k|\mathbf{x})$. Using Bayes' theorem, we know that

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}$$

In machine learning jargon, $P(C_k)$ is called the *prior* probability distribution, as it determines the probability of the given class before observing any data. Consequently, $P(C_k|\mathbf{x})$ is called the *posterior* probability distribution, as it determines the probability of a given class after having observed some data. $P(\mathbf{x}|C_k)$ is called the *likelihood function* and $P(\mathbf{x})$ is simply a normalising factor, making sure that the posterior is a proper probability measure. Thus, the posterior is proportional to the likelihood times the prior. For maximising the posterior probability, we can maximise the likelihood using *maximum likelihood* techniques, in which we choose the C_k that maximises $P(\mathbf{x}|C_k)$ and hence also $P(C_k|\mathbf{x})$.

Often, we make use of statistical models instead of computing the classification directly. These models come in two variants: *parametric* and *non-parametric*. Examples of parametric models are Neural Networks, Hidden Markov Models and Support Vector Machines. An example of a non-parametric model is hierarchical clustering, which we will describe in a later section. We can use Bayes' theorem to determine the optimal parameters given a data observation. In this case, let $\boldsymbol{\theta}$ be the parameters to the model and $\mathbf{x}$ be an observation. Then $P(\boldsymbol{\theta})$ is the prior, $P(\boldsymbol{\theta}|\mathbf{x})$ is the posterior and $P(\mathbf{x}|\boldsymbol{\theta})$ is the likelihood function.

Before feeding data to the chosen model it is often wise to do some preprocessing in order to get the best possible results or simply to make the calculations of y tractable. In particular, we are always interested in reducing the dimensionality of the feature space. This preprocessing is usually called *feature extraction*, since we take a complex data set and extract the features that are important for the model. Consider for example the problem of classifying a melody according to key: Given a melody $\mathbf{m}$ consisting of notes, where each note is a triple (p, o, d) where p denotes the pitch class, o denotes the octave and d denotes the duration of the note. Assume that there are 12 pitch classes corresponding the the pitches of the Western 12-tone system and assume that notes of the melody span four octaves. Finally, assume that there are three types of durations: Quarter notes, eighth notes and sixteenth notes. Thus, we have $12 \cdot 4 \cdot 3 = 144$ possible notes. But in order to determine the key of the melody, we do not need to consider the duration nor the octave of each note; the pitch classes themselves would be sufficient. Therefore, we can cut off the o and the d from the data set, which leaves p back and therefore only 12 possibilities. This immensely reduces the dimensionality of the feature space.

2.2 Harmonic theory

In later sections (Section 6, specifically) we will use some harmony theory to argue for our feature extraction. This section will give a short introduction to scale modes and harmonic structures. For more details see [21] and [4].

A *scale* is defined as a sequence of notes in ascending order within an octave. For example, the C major scale is given by

C D E F G A B

More generally, this can be represented by the intervals in semitones between the notes:

2-2-1-2-2-2-1

where the last number is the interval between B and the C in the next octave. This is the general representation of a major scale and can be used to generate major scales in every key. This scale is also called a *ionian* scale. If we number the notes in the scale from 1 to 7, we can refer to the notes in a general major scale independent of the key. These numbers are called *scale degrees*. For example, we see that F is the fourth note in the C major scale. The C ionian scale is seen in notational form at the top of Figure 2.

Rotating the interval representation by one step to the left gives

2-1-2-2-2-1-2

which is another scale, called a *dorian* scale. This can be seen as playing the C major scale, starting and ending with D, which is then a D dorian scale. We see that a C dorian corresponds

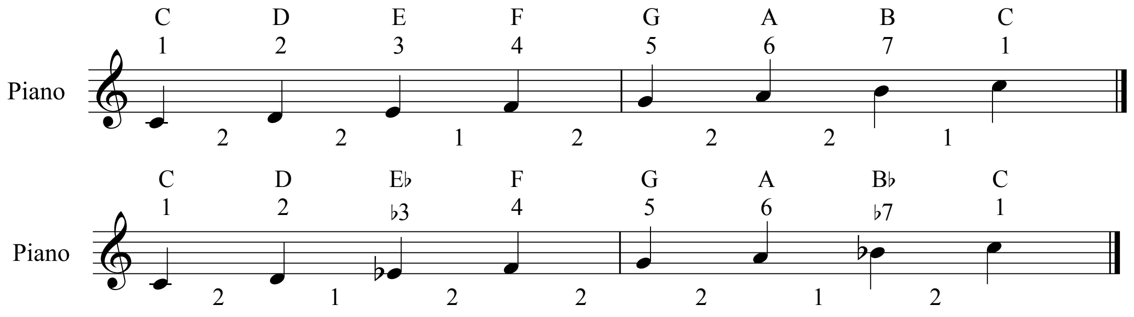


Figure 2: C ionian (top) and dorian (bottom) scales. Note that the dorian intervals are simply the ionian rotated once to the left

to

C D E♭ F G A B♭

or with the more abstract scale degrees:

1 2 ♭3 4 5 6 ♭7

This is seen in the bottom of Figure 2. All seven rotations of this representation give rise to unique scales (collectively called the *major scale modes*), each having a specific interval signature. The scale modes along with their representations are seen in Table 2.

Mode	**Intervals**	**Degrees**
Ionian	2-2-1-2-2-2-1	1 2 3 4 5 6 7
Dorian	2-1-2-2-2-1-2	1 2 ♭3 4 5 6 ♭7
Phrygian	1-2-2-2-1-2-2	1 ♭2 ♭3 4 5 ♭6 ♭7
Lydian	2-2-2-1-2-2-1	1 2 3 ♯4 5 6 7
Mixolydian	2-2-1-2-2-1-2	1 2 3 4 5 6 ♭7
Aeolian	2-1-2-2-1-2-2	1 2 ♭3 4 5 ♭6 ♭7
Locrian	1-2-2-1-2-2-2	1 ♭2 ♭3 4 ♭5 ♭6 ♭7

Table 2: Structure of the major scale modes

A *chord* is a collection of notes played simultaneously. In principle a chord can be any number of notes played, but usually we consider *triads* (chords consisting of three notes) as the basis of all chords. We can define the structure of the chords from the scale degrees. For example, the C major chord consists of the notes C, E and G. This corresponds the the first, third and fifth degree of the C ionian scale. Table 3 and Figure 3 show the structure of the different triads.

In jazz harmony the seventh chords, instead of the triads, are the basic harmonic building blocks. Seventh chords are triads with added 7'th degree note. Corresponding to the C major triad, we have the major seventh chord (denoted Cmaj7), which consists of the notes C, E, G

Chord	**Structure**
Major	$1, 3, 5$
Minor (m)	$1, \flat 3, 5$
Augmented (+)	$1, 3, \sharp 5$
Diminished (°)	$1, \flat 3, \flat 5$
Suspended	$1, 4, 5$

Table 3: Structure of triads

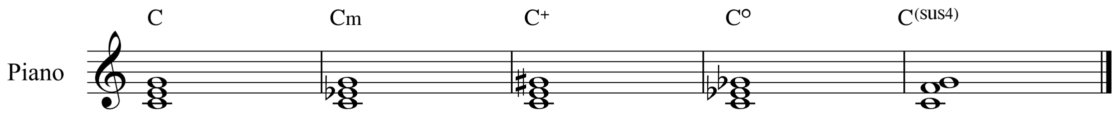


Figure 3: Triads in the key of C

and B, which are the first, third, fifth and seventh degree of the ionian scale. The structure of the seventh chords we will consider are seen in Table 5 and in Figure 4. Each chord can be labeled according their harmonic functions, which are shown in Table 4.

Harmonic function	**Chords**
Tonic	Imaj7, IIIm7, VIm7
Subdominant	IIm7, IVmaj7
Dominant	V7, VII$^{\varnothing}$7

Table 4: Harmonic functions of diatonic chords

We can build a series of seventh chords that consists of notes entirely within a given scale by harmonising each note within the scale. Such chords are called *diatonic* and are designated with a roman numeral corresponding to the scale degree of the root note along with added markings that indicate modifications of the original form of the chord. For example, harmonising the fourth scale degree of the ionian scale as a diatonic chord gives IVmaj7. Figure 5 shows the diatonic chords of the C ionian scale.

One of the most common cadence in standard jazz is the IIm7-V7-Imaj7 chord progression. This notation should be read as the second, the fifth and the first degree of the ionian scale. In the key of C, this corresponds to Dm7-G7-Cmaj7. The equivalent in rock, is the tonal cadence IV-V7-I, corresponding to C-F-G7 in the key of C.

2.3 Mathematical musical notation

In this text we will describe formal methods and algorithms treating musical features and therefore we need to formalise the notions of notes, pitches and durations, so that we can treat them in a mathematical context. We will therefore introduce the mathematical musical notation (not to be confused with musical notation, as in written notes) that will be used throughout this text.

The most basic element of music is the notion of pitches. When describing pitches, we

Chord	**Structure**
maj7	$1, 3, 5, 7$
6	$1, 3, 5, 6$
mmaj7	$1, \flat 3, 5, 7$
m6	$1, \flat 3, 5, 6$
m7	$1, \flat 3, 5, \flat 7$
$^{\varnothing}7$ (half-diminished)	$1, \flat 3, \flat 5, \flat 7$
$^{\circ}7$ (diminished)	$1, \flat 3, \flat 5, \flat\flat 7$
7	$1, 3, 5, \flat 7$
7 ($\flat$9)	$1, 3, 5, \flat 7, \flat 9$
7 (+9)	$1, 3, 5, \flat 7, \sharp 9$
7 (sus4, $\flat$9)	$1, 4, 5, \flat 7, \flat 9$
7 (sus4)	$1, 4, 5, \flat 7$
7 (+5)	$1, 3, \sharp 5, \flat 7$

Table 5: Structure of 7'th chords

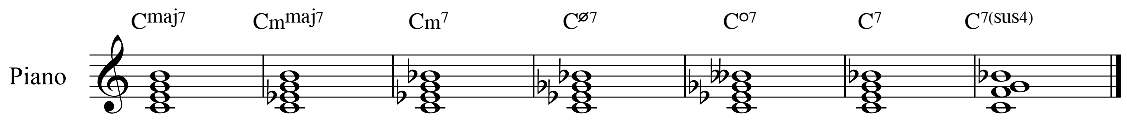


Figure 4: 7'th chords in the key of C

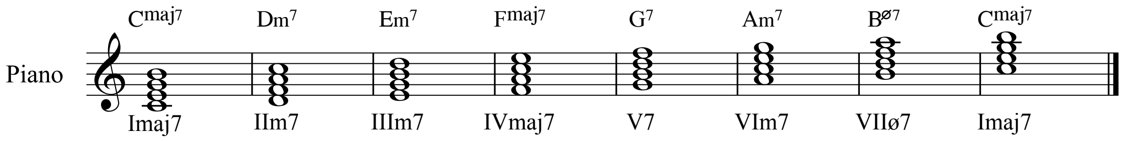


Figure 5: Diatonic chords in C major

will distinguish between pitches and pitch classes, as Hudak does in [13]. We define the *pitch class* as the set of all pitches within an octave and a special value R that denotes no pitch (i.e. a rest). Formally,

$$\mathbf{P}_C = \{\mathrm{C}, \mathrm{C}\sharp, \mathrm{D}, \ldots, \mathrm{B}, \mathrm{R}\} = \{\mathrm{C}, \mathrm{D}\flat, \mathrm{D}, \ldots, \mathrm{B}, \mathrm{R}\}$$

We see that $|\mathbf{P}_C| = 13$ (the twelve pitches of the octave and the rest).

When we talk about *pitches* we will refer to absolute tones, represented as a pitch class and an octave. Formally, we can define this as

$$\mathbf{P} = \{(P, o) \mid P \in \mathbf{P}_C,\ o \in \mathbb{N}\}$$

We will use the short-hand $P_o = (P, o)$ in the rest of this text, such that a C in octave five is notated as C_5. For the durations of notes we choose the standard of dividing into halfs, quarters, eighth and so on. So, we define the set of musical durations as

$$\mathcal{D} = \{a/2^b \mid a, b \in \mathbb{N}\}$$

In most of this text we will describe music in the context of *notes*. We define a note as a pair of a pitch and a duration. The set of all notes is defined as

$$\mathrm{N} = \{(p, d) \mid p \in \mathbf{P}, d \in \mathcal{D}\}$$

The notion of *melody* can be defined as an ordered list[2]

$$\mathbf{m} = [n_1, n_2, \ldots, n_n \mid n_i = (p_i, d_i) \text{ for } i = 1, 2, \ldots, n]$$

of time indexed notes. We will furthermore distinguish between the *length* $|\mathbf{m}|$ of melody, which is the number of elements in the list, i.e. the number of notes and the *duration* of the melody $D(\mathbf{m})$, which is the sum of the durations of all the notes in the melody ($D(\mathbf{m}) = \sum_{i=1}^{k} d_i$).

Chords are also defined as a list of notes, except that in chords, each note will have the same duration and are placed at the same time index. That is

$$\mathbf{c}_k = [n_1, n_2, \ldots, n_m \mid n_i = (p_i, d_k) \text{ for } i = 1, 2, \ldots, m]$$

where k is the time index and $D(\mathbf{c}_k) = d_k$.

These are the basic definitions we need in order to discuss our musical elements in a mathematical framework.

[2]We will use square brackets [...] to denote ordered lists and curly brackets {...} to denote sets

3 Haskell and libraries

In this section, we will discuss why Haskell is a good choice for doing algorithmic composition and go through the libraries that are central. It is intended as a foundation for some parts of the implementation, and may be read later on when needed.

3.1 Why Haskell?

As Samuel Silas Curry[3] once said:

> "All expression comes from within outward, from the center to the surface, from a hidden source to outward manifestation. The study of expression as a natural process brings you into contact with cause and makes you feel the source of reality."

This quote could describe the beauty of Haskell: Haskell is a statically typed, lazy functional programming language which allows for a very elegant and concise programming style. The functional approach allows features that are not possible in most imperative languages [23] in the same elegant way. One of these features is the possibility to operate on infinite data types, such as lists. This is possible, since the desired part of the list is first computed when it is used. Therefore, if a music structure is represented by a list, we do not need to worry about the length of the list. Also, it is possible to work with functions like other kind of data. It means that a function can be passed as an argument to other functions, called higher-order function.

In Haskell, programs can be written more quickly, and the use of a static type system eliminates many common programming errors and guarantees run-time type safety. Paul Hudak argues [13], that with Haskell it is possible to allow musical ideas to take their natural form as Haskell expression by focusing on *what* a musical entity is rather than *how* to create it.

We have chosen Haskell to implement our methods for algorithmic composition because of all these benefits, and because the Yale Haskell Group of the University of Yale is currently developing a music library called Euterpea [13], which is a new version of the Haskore [12] library also developed at Yale. The version of Euterpea is not officially released yet, and is to be considered as "experimental". By choosing Haskell and Euterpea we benefit from all the before mentioned advantages, but the drawback has been that we have encountered some serious bugs in Euterpea, though these bugs have been successfully corrected with great help from Paul Liu at the University of Yale.

As the project evolved we encountered some issues concerning the design of Euterpea that complicated our project. A major issue with the library is the lack of time signature in the representation of music. Although many aspects are easy to work with, it seems like the way of representing music is still too close to the way MIDI represents notes: Notes have a duration given by a fraction, but since no bars are present it almost corresponds to writing music in the traditional way using only one bar where all notes are contained. This becomes cumbersome when big pieces of music divided into several channels are to be analysed.

[3]Haskell Curry's father. Haskell Curry who, along with Alonzo Church, helped establish the theoretical foundations of functional programming in the 1940's.

3.2 Euterpea

Euterpea is a collection of Haskell modules designed for expressing musical structures in the high-level, declarative style of functional programming. These musical structures consist of primitive notions such as notes and rests, operations to transform musical objects such as transpose and tempo-scaling, and operations to combine musical objects to form more complex ones, such as concurrent and sequential composition. From these simple roots, much richer musical ideas can easily be developed[12].

In Euterpea, the representation of musical structures is done with the `Music a` data type, and the semantic (or rather interpretation when talking about music) is done by transforming `Music a` to the `Performance` data type, from where it is possible to export to MIDI [15], CSound [24] or other low-level representation.

Music representation

Let's start by looking at the pitch type, which consists of a *pitch class* and the octave in which the pitch is defined:

```
type Pitch      = (PitchClass, Octave)
data PitchClass = Cff | Cf | C | Dff | Cs | Df | Css | D | Eff | Ds
                | Ef | Fff | Dss | E | Es | Ff | F | Gff | Ess | Fs
                | Gf | Fss | G | Aff | Gs | Af | Gss | A | Bff | As
                | Bf | Ass | B | Bs | Bss
 deriving (Eq,Ord,Show,Read,Enum)
type Octave     = Int
```

The *PitchClass* data type declares all 12 semitone by 35 pitch class names. *Cf* is read as "C-flat" and normally written as C♭ and *Cs* is read as "C-sharp", normally written as C♯, and so on. For a reference, the notion of "the concert key A" is denoted $(A, 4)$ in the above design. The representation of musical structures is done with the `Music a` type, here depicted together with the data types used in *Music*:

```
data Music a = Prim (Primitive a)
             | Music a :+: Music a
             | Music a :=: Music a
             | Music a :=/ Music a
             | Modify Control (Music a)
     deriving (Show, Eq, Ord)

data Primitive a = Note Dur a
                 | Rest Dur
     deriving (Show, Eq, Ord)

data Control = Tempo Rational
             | Transpose  AbsPitch
             | Instrument InstrumentName
             | Phrase     [PhraseAttribute]
             | Player     PlayerName
```

```
j251 :: Music Pitch
j251 = let dMinor = (d 4 hn :=: f 4 hn :=: a 4 hn :=: c 4 hn)
           gMajor = g 4 hn :=: b 4 hn :=: d 4 hn :=: f 4 hn
           cMajor = c 4 bn :=: e 4 bn :=: g 4 bn :=: b 4 bn
   in Modify (Tempo 2) (Modify (Transpose 6) (dMinor :+: gMajor :+: cMajor))
```

Figure 6: II-V-I cadence in C major, transposed by 2 half notes to D major, and scaled in tempo by a factor of 2.

```
     deriving (Show, Eq, Ord)

type PlayerName = String
```

The *a* type can either be a *Pitch* type or a *(Pitch, [NoteAttribute])*, where the *NoteAttribute* is a type, that the *Player* function can use for interpreting the *Music*, and is explained in the next subsection. A *Note* is its duration paired with the *a* type, where a *Rest* only has a duration. The duration is defining *whole notes (wn), half notes (hn)* and so on, together with convenient pitch conversion functions. From the two atomic constructors, *Note* and *Rest*, we can build more complex musical structures as follows [12]:

- *m1* :+: *m2* is the *sequencial composition* of *m1* and *m2* ; i.e., *m1* and *m2* are played in sequence.
- *m1* :=: *m2* is the *parallel composition* of *m1* and *m2* ; i.e., *m1* and *m2* are played in simultaneously. The duration of the result is the duration of the longer of *m1* and *m2* .
- *m1* :=/ *m2* is also the *parallel composition* of *m1* and *m2* , but the duration of the result is the duration of the shorter of *m1* and *m2* .
- *Tempo a m* scales the rate at which *m* is played (i.e., its tempo) by a factor *a*.
- *Transpose i m* transposes *m* by an interval *i* measured in semitones.
- *Instrument iName* declares, that *m* is to be performed by instrument *iName*, which is one of 129 names defined in the data type *InstrumentName*.
- *Phrase* and *Player* are closely related, and has to do with the interpretation of the music. This will be explained in the next subsection.

A simple example using these constructors is shown in Figure 6. Here, we have a II-V-I cadence transposed 2 half tones from C major to D major, and scale in tempo by a factor of 2.

Interpretation and Performance

The presentation so far has only been considering the syntax of the representation of music structures in Euterpea. We have to give a meaning to these musical values, namely the semantics, or interpretation. In conventional music this process is very subjective, appealing to aesthetic judgment. Therefore, Euterpea has a notion of *player* that can manage these aesthetic judgments such as dynamics, subtleties in tempo and phrasing. We will not go too much into the details, since the goal of this project is to focus on the compositional aspect of music, and not how to perform a composition. But since a *player* is necessary to transform the music structure to MIDI, it is important to know the construction. A Euterpea player is a 4-tuple consisting of a name and three functions: One for interpreting notes, one for phrases, and one for producing a properly notated score:

```
data Player a = MkPlayer{pName :: PlayerName,
                         playNote :: NoteFun a,
                         interpPhrase :: PhraseFun a,
                         notatePlayer :: NotateFun a }
     deriving Show

type NoteFun a = Context a -> Dur -> a -> Performance
type PhraseFun a = PMap a -> Context a ->
                   [ PhraseAttribute ] ->
                   Music a -> (Performance , DurT )
type NotateFun a = ()
```

The *PhaseAttribute* in the *Control* data type defined in 3.2 contains *Dynamic, Tempo, Articulation* and *Ornament* attributes, and gives great flexibility in the interpretation process, because they can be interpreted by different players in different ways.

The *Performance* type is the lowest of the music representation not yet comitted to MIDI or other low-level representation.

```
type Performance = [Event]
data Event = Event { eTime :: PTime , eInst :: InstrumentName,
                     ePitch :: AbsPitch, eDur :: DurT,
                     eVol :: Volume , pFields :: [ Float ] }
     deriving (Eq,Ord,Show)

type PTime = Rational
type DurT = Rational
type Volume = Integer
```

An event *Event*$\{eTime = s, eInst = i, ePitch = p, eDur = d, eVol = v\}$ captures the fact that at start time s, instrument i is playing the pitch p with volume v for a duration d, where the duration is here measured in seconds. The *pField* is for special instruments that require extra parameters. To generate a *Performance*, the function *perform* with has the signature *perform :: PMap a* $\rightarrow$ *Context a* $\rightarrow$ *Music a* $\rightarrow$ *Performance*. The *Context* is telling when to start the performance, which instrument to use, the key and the tempo. The *Context* is depicted below:

```
data Context a = Context{cTime :: PTime, cPlayer :: Player a,
                         cInst :: InstrumentName, cDur :: DurT,
                         cKey :: Key, cVol :: Volume }

     deriving Show
```

```
type PMap a = PlayerName -> Player a
type Key = AbsPitch
```

Here *cTime* is the time to begin the *Performance*, *cPlayer* is the player that we want to use for interpreting the music, *cInst* is the instrument, *cDur* is the time it takes to play one whole note, and *cKey* and *cVol* is the key and volume, respectively. The *PMap a* is mapping the player name to the right *Player a* type.

From Perfomance to MIDI

When having the music represented as a *Performance* data type, the translation to MIDI can be done by the function *toMidi* which has the signature *toMidi :: Performance → UserPatchMap → Midi*. The *UserPatchMap* consists of a list of tuples containing a channel and an instrument to be assigned to that channel.

```
type UserPatchMap = [(InstrumentName,Channel)]

type Channel = Int
```

The General MIDI instrument family consist of 128 different instruments and the *InstrumentName* data type comes directly from these, except for *Percussion* and *Custom* which were added for convenience and extensibility. A MIDI channel is a kind of a programmable instrument, and there are 16 different channels nummerated from 0 to 15 in the Euterpea implementation.

3.3 Data.MarkovChain

Data.MarkovChain is a simple library for working with Markov chains and consists of the two functions *run* and *runMulti*. The signature are:

```
run :: (Ord a, RandomGen g) =>
      Int  -- ^ size of prediction context
   -> [a]  -- ^ training sequence, the one to walk through randomly
   -> Int  -- ^ index to start the random walk within the training sequence
   -> g    -- ^ random generator state
   -> [a]
```

```
runMulti :: (Ord a, RandomGen g) =>
      Int    -- ^ size of prediction context
   -> [[a]]  -- ^ training sequences, the order is relevant
   -> Int    -- ^ index of starting training sequence
   -> g      -- ^ random generator state
   -> [[a]]
```

The *size of prediction context* is the order of the Markov chain, i.e. the number of states to be considered for the next state. The *training sequence* `[a]` is the data on which the Markov chain is build and has to be an instance of *Ord*, the *index to start the random walk* is simply the index of the element in the training sequence to start the random walk based on the Markov transitions, and the *random generator state* `g` creates a new *StdGen* pseudo-random number generator state.

3.4 Data.HMM

Data.HMM is a simple library for working with Hidden Markov Models. It implements the Viterbi's algorithm and the forward algorithm, and consists of the three functions *train, bestSequence* and *sequenceProb*. The signatures are:

```
train :: (Ord observation, Ord state) =>
      [(observation, state)] -> HMM state observation
bestSequence :: Ord observation =>
      HMM state observation -> [observation] -> [state]
sequenceProb :: Ord observation =>
      HMM state observation -> [observation] -> Prob
```

Here, *train* calculates the parameters of an HMM from a list of observations and the corresponding states, *bestSequence* calculates the most likely sequence of states for a given sequence of observations using Viterbi's algorithm and *sequenceProb* calculates the probability of a given sequence of observations using the forward algorithm.

3.5 HFANN

HFANN is a Haskell binding for the Fast Artificial Neural Network (FANN) library[4], which is a library for working neural networks in C. It implements different kinds of networks and backpropagation algorithms. The main functions we use from the HFANN library are the following:

```
withStandardFann
::
=> [Int] The ANN structure
-> FannPtr -> IO a A function using the ANN
-> IO a The return value

trainOnFile
:: FannPtr The ANN to be trained
-> String The path to the training data file
-> Int The max number of epochs to train
-> Int The number of epochs between reports
-> Double The desired error
-> IO ()

runFann
:: FannPtr The ANN
-> [FannType] A list of inputs
-> IO [FannType] A list of outputs
```

withStandardFann generates a new neural network with the structure given as a list of integers, where the first entry is the number on input units, the last entry is the number of output units and entries in-between are numbers of hidden units in the hidden layers. The second

[4]http://leenissen.dk/fann/ (Tue Jan 12 13:35:37 CET 2010)

argument is a function that determines what to do with the neural network, e.g. to train a network on a data set placed in a file on the harddisk, you would define a function that sets the parameters for training and calls *trainOnFile*.

runFann takes a pointer to the trained network and a list of inputs, where each entry in the list is the input to the corresponding input unit in the network.

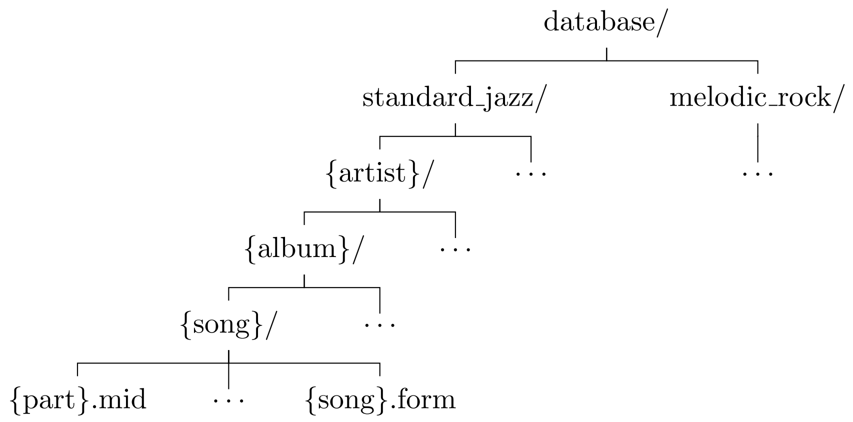


Figure 7: The structure of the database. The text inside the brackets {} is telling which type of function the file/directory has, where text without brackets are actual folders in the database structure. The structure in the folders standard_jazz and melodic_rock are the same, and therefore only the one of them is depicted, the same with files/directories depicted with . . .; they have the same structure as the one at the same level.

4 Database

For this project to be successful, a large database of tunes needed to be established. We managed to create 129 tunes divided into two genres; 86 in melodic rock and and 43 in standard jazz. The list of tunes can be found in Appendix A. We have constructed the database by carefully choosing the tunes in each genre and gathering the material in note written form. Using a note-OCR program, we have corrected and modified the tunes as described in this section. First of all, we had to settle on some predefined form for all tunes, so that analysing the database would become easier. Dividing the tunes into parts has the advantage, that it is possible to choose which parts of the tunes to use as the foundation for the new composition, and making it a much easier task for the system to create a similar form of the new composition in correlation with the tunes in the database. We decided to use MIDI as the format for the files. Building such an immense database customised for our purpose has been a very time consuming job and we have used about 200 construction-hours.

4.1 Structure of the database

Generally seen, the structure of the database is depicted in Figure 7, and consists of nested directories as follows: The outmost directory consists of the genres, and in each of these subdirectories we have all the artists in the given genre. For each of the artists, a subdirectory contains each of the divided tunes in MIDI-format together with a form file (*.form), telling the sequence of the parts for the song. Depending on the number of tunes for an artist, the tunes can be classified in a subdirectory by album or other meaningful classification.

4.2 Structure of the tunes

We have chosen to split the tunes into the following parts: *intro, a,b,c,.., bridge, chorus* and *outro*. For variations in the different parts, we numerate the parts, e.g. *a1, a2*, and for each of these parts, except the bridge, a tag tells if it consists of a melody belonging to the main melody, *mm*, if the melody is of a different character, *dcm* or there are no clear melody, *nm*. The main melody is defined as a melody that is central in the tune, and often performed by the main instrument (e.g. the singer), and the tag *mm* can be omitted since it is often the case that we have a main melody. A different character melody (*dcm*) is a melody that diverge from the main melody in some way, and it can be a very subtle decision to distinguish between a main melody and a melody with a different character. As a rule of thumb, we will characterise a melody with a different character in the melodic rock-genre, if the melody is being played by another instrument than the voice, and the melody is not suitable for being sung. In the standard jazz genre, it is more difficult to make a similar rule of thumb, but often we are spared to make these decisions in that genre due to the form of the note material, since the melody will almost always be the main melody. For a part with no melody, it is often a sequence only consisting of chords. If the part contains no melody, and act like a glue between two parts, we will call the part *bridge*.

We also have to deal with voltiges, and this is done by adding the voltige number after a part by an underscore, e.g. *a1_volt1*. Another issue is how to cope with upbeats. For instance we can have an upbeat to for example the *a* part located in the *intro* part. Clearly, this upbeat should not be considered belonging to the *intro* part, but should instead belong to the *a* part. Furthermore, we can have a series of parts each having an upbeat to the following part, say: *intro, a1, a2, chorus, outro*. A way to cope with this, could be to declare which parts that have common notes, and to which part the upbeat is belonging, but this would involve embedding all the given parts, e.g.

$$\textbf{upbeat(}\textit{intro},\textbf{upbeat(}\textit{a1}, \textbf{upbeat(}\textit{a2}, \textbf{upbeat(}\textit{chorus, outro})))) $$

This would be unnecessary complicated both in the parsing and in the representation in Cremo, and therefore we have chosen to solve the problem by modifying the involved parts by remove the notes for the upbeat, and instead keep this upbeat (together with the chords) in a part of its own with the name *u* with an optional numeration, followed by an underscore and the name for the part that the upbeat belongs. In the form, we use the syntax *(u_part, part)*, which tells that we have a upbeat *u_part* to part *part*, e.g.

intro, (u1_a1, a1), (u1_a2, a2), chorus, (u_outro, outro)

which means that we have an upbeat to *a1, a2* and *outro* located in the respective files. The regular expression for the form of the database is depicted in the regular expression in Figure 8, where part $\in \alpha \cup \{\text{chorus}, \text{bridge}\}$, for $\alpha = \{a, .., z\}$. We have omitted the syntax for upbeat in the regular expression, since this would entangle the readability of the form of the database. The regular expression for the upbeat construction is instead depicted separately in regular expression in Figure 9. For generality, we here define parts $\in \{\text{intro}, \text{intro}_{\text{volt}}, \text{part}, \text{part}_{\text{volt}}, \text{bridge}, \text{bridge}_{\text{volt}}, \text{outro}, \text{outro}_{\text{volt}}\}$.

An example is given in Figure 10. First, we have some meta data telling which song is parsed and the key signature of the song. After that, the actual form of the song is listed. We see that the *intro* consists of two voltiges, succeeded by *a1*, both parts with a main melody.

$$(\text{intro}^{+}\ \text{intro}_{\text{volt}}?)^{*}\ (\text{part}^{+}\ \text{part}_{\text{volt}}?)^{*}(\text{outro}^{+}\ \text{outro}_{\text{volt}}?)^{*}$$

Figure 8: Regular expression for the form of the tunes in the database

$$(\text{u\_parts}_{x}, \text{parts}_{x})$$

Figure 9: Regular expression for the upbeat construction

Then we have an upbeat *u1_bridge1* to the *bridge1* part with a different character melody succeeded by *a2*. The form continuous with structure and finish with an *outro* part. In section 4.4 we will show how these files are being represented in Haskell.

4.3 Structure of the MIDI files

The structure of the MIDI file has been made, so that melody, chords and accompaniment are separated in different channels. The structure is depicted below:

Channel 1 consists of the melody.

Channel 2 consists of the chords generated from the chords diagrams in the score with the root located as the lowermost note of the chord. For every chord symbol, the duration of the notes last to the next chord unless a new bar starts without a chord symbol. In this particular case, the chord is entered again in beat 1 of the new bar.

Channel 3-5 are optional and are not present in all tunes. They can each consist of guitar, piano or other instrument playing riffs, chords etc.

It is crucial to have separated these different channels, since extracting chords or melody in a mixed environment is not a trivial task. Figure 11 shows the structure of the tunes.

4.4 Representation of the database in Haskell

We can search and extract music from the database by genre and a number of artists in the given genre. This search will return a music tree, `MTree`, which is defined in Figure 12.

From that we see that a genre node (`GenreNode`) has a list of artist nodes (`ArtistNode`) that stores the name of the artist and has a reference to a list of tune nodes (`TuneNote`). `TuneNote` stores the name of the tune and a list containing the form of tune. The `TuneNode` has a reference to list of channel nodes (`ChannelNode`), which stores the number of the MIDI channel and a list of part nodes (`PartNode`). `PartNode` stores the name of the part as a string (including the *type*) together with the key of the part represented as the numbers of semitones from C. For example the key E is represented by 4, since there are 4 half-notes from C to E. `MusicPart` stores the actual representation of the part as the `Music` type generated from the Euterpea library. There is another attribute, namely `UpbeatPart`. This stores the upbeat to the part `MusicPart`. If there is no upbeat to the part, the attribute contains `Nothing`. With this representation, we have all the information needed for composing new tunes, both with respect to the form and the individual parts.

```
song=Trouble
key=G
intro
intro_volt1
intro
intro_volt2
a1
(u1_bridge1,bridge1 dcm)
a2
(u1_chorus,chorus)
(u1_bridge1,bridge1 dcm)
a2
(u1_chorus,chorus)
bridge2
b1
outro
```

Figure 10: Content of the form file for the song "Trouble" by Coldplay

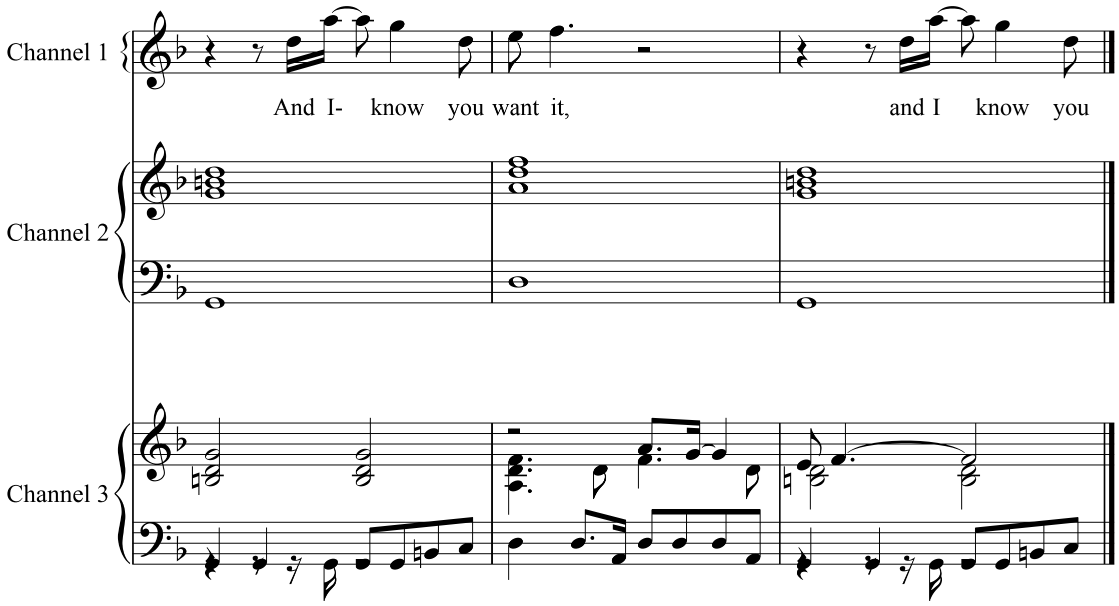


Figure 11: This snippet of the chorus of the Mercury Rev song *Chains* shows the structure of the MIDI files. Channel 1 contains the main melody, Channel 2 contains the chords and Channel 3 contains other voices.

```
data MTree = GenreNode Genre [AN]
data AN = ArtistNode Artist [TN]
data TN = TuneNode (SongTitle, [Form]) [CN]
data CN = ChannelNode Ch [PN]
data PN = PartNode ((PartType, KeyInt), MusicPart) Upbeat

type Genre = String
type Artist = String
type SongTitle = String
type Form = String
type PartType = String
type MusicPart = Music (Pitch, [NoteAttribute])
type Ch = Integer
type KeyInt = Int
type Upbeat = Maybe (Music (Pitch, [NoteAttribute]))
```

Figure 12: The MTree structure in which the tunes are represented

If the structure of the database directories is obeyed, it is possible to add more genres and artists to the database by simply adding the directories, including a correct form file, in the directory structure. This is possible, because we recursively construct the directory structure in a tree structure `DTree`, collecting the paths to all `*.form` files with respect to the selected artists. These form files are parsed, and the information is then saved in the `MTree` data structure, we just discussed. This construction makes searching the database very flexible, since the size of the database can be scaled without configuring anything. The `DTree` structure can be seen in Figure 13, and is simply a structure of directory nodes (`DirNode`) storing the path of the directory and a list of `DTree`, or it can be a file node (`FileNode`) storing the path to the file. When constructing the `DTree`, we are only fetching the path of the `FileNode`'s ending on `.form`.

```
data DTree = FileNode String | DirNode String [DTree]
```

Figure 13: The DTree structure for representing the structure of the directories for the database

Using the example in Figure 10, where only the first three lines are parsed, we get the MTree structure depicted in Figure 14.

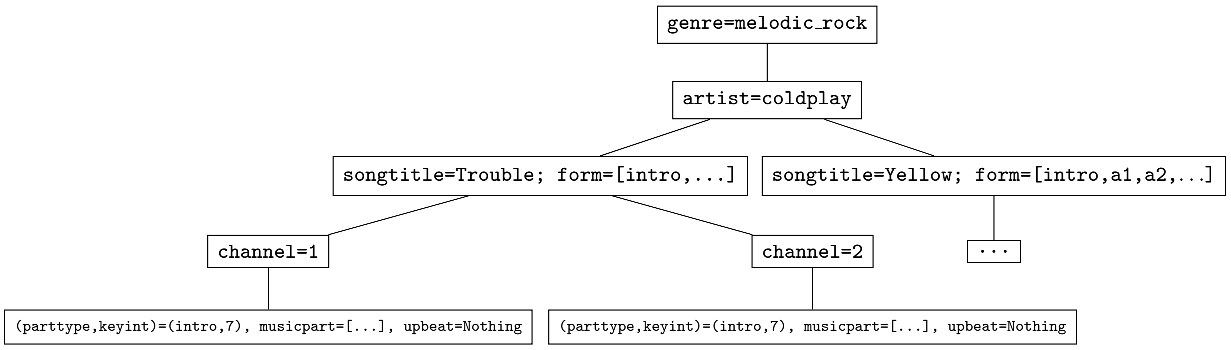

genre=melodic_rock
artist=coldplay
songtitle=Trouble; form=[intro,...]
songtitle=Yellow; form=[intro,a1,a2,...]
channel=1
channel=2
...
(parttype,keyint)=(intro,7), musicpart=[...], upbeat=Nothing
(parttype,keyint)=(intro,7), musicpart=[...], upbeat=Nothing

Figure 14: The MTree structure for the first 3 lines of example 10

5 Melodic generation

This part of Cremo generates new melodic lines from the melody contents of the database. This section describes how we determine the features of the melody lines and how we generate new lines using simple Markov models.

5.1 Markov chains

In order to determine the best possible features, we need to study the method we will be using for the generation of melodies, namely Markov chains. A Markov chain is a discrete stochastic process that has the *Markov property*, meaning that all future states only depend on the present state [10]. Following a similar notation as in [3], let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ be a sequence of stochastic variables that take values in some countable set S, called the *state space*, consisting of K states $s_1, s_2, \ldots, s_K$. This can be represented using a 1-of-K coding, such that $\mathbf{x}_k$ is K-dimensional and each element represents af state in the state space. If we denote the i'th element of $\mathbf{x}_k$ as x_{ki}, we have

$$x_{ki} = \begin{cases} 1 \text{ if } \mathbf{x}_k \text{ is in state } s_i \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

The Markov property can thus be stated as

$$P(\mathbf{x}_n | \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}) = P(\mathbf{x}_n | \mathbf{x}_{n-1}) \tag{2}$$

Actually, we can generalise and call a process with this property a first-order Markov chain. Consequently, the k'th-order Markov chain will be a process where future states only depend on the k last states. A Markov chain can be described by its state-transition matrix $\mathbf{A}$, where each entry A_{ij} is the transition probability

$$A_{ij} = P(x_{nj} = 1 | x_{n-1,i} = 1) \tag{3}$$

Thus, $\mathbf{A}$ is an $K \times K$ matrix. Since $\mathbf{A}$'s entries are probabilities, we see that each row of $\mathbf{A}$ sums to one, that is

$$\sum_{j=1}^{K} A_{ij} = 1 \tag{4}$$

In Figure 15 we see a graphical example of a first-order Markov chain.

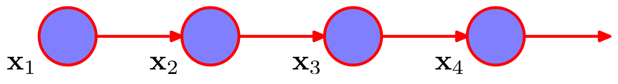


Figure 15: First-order Markov chain

5.2 Melodic features

For generating melodic lines we make use of Markov chains and we therefore need a way to represent the melodic content that makes it easy to map to a Markov chain. If we have a melody line of length m returned by the search in the database, where we have extracted the pitches and durations, we can define the new feature vector as

$$\mathbf{m} = [n_1, n_2, \ldots, n_n] \tag{5}$$

where $n_k = (p_k, d_k) \in \mathrm{N}$ for $k = 1, \ldots, n$.

We see, however, that with this representation, we will have $(\mathrm{C}_4, \frac{1}{4}) \neq (\mathrm{C}_4, \frac{1}{8})$, which is inappropriate as it makes the state space much larger than it has to. This gives a dimensionality of $|\mathbf{P_m} \times \mathcal{D}_\mathbf{m}|$ where $\mathbf{P_m}$ is the set of pitches in $\mathbf{m}$ and $\mathcal{D}_\mathbf{m}$ is the set of durations in $\mathbf{m}$. Assuming for example that the melodic content of the database moves within 4 octaves and is divided into whole, half, quarter, eigthth and sixtheenth notes, we have $|\mathbf{P_m}| = 13 \cdot 4$ and $|\mathcal{D}_\mathbf{m}| = 5$ and thus the features have dimensionality $13 \cdot 4 \cdot 5 = 260$ states. So, to reduce the dimensionality of the feature space we unzip pitches and durations to two different lists. We can then treat each list as independent feature vectors and supply them to two different Markov chains with considerably lower dimensionality. We have extracted the rhythmic content from the melodic content and thus train these two separately, and we therefore have the following two feature vectors in our implementation:

$$\begin{aligned} \mathbf{m}_p &= [p_1, p_2, \ldots, p_n] && (6) \\ \mathbf{m}_d &= [d_1, d_2, \ldots, d_n] && (7) \end{aligned}$$

We now have dimension $|\mathbf{P_m}|$ for the pitches and dimension $|\mathcal{D}_\mathbf{m}|$ for the durations.

There are some drawbacks of considering pitches and durations independently. Consider a small chromatic movement somewhere in the corpus, say $(\mathrm{D}_5, \frac{1}{16}), (\mathrm{E}\flat_5, \frac{1}{16}), (\mathrm{E}_5, \frac{1}{16})$. The durations here are important because the short durations makes the melody move along, whereas if the duration of the $\mathrm{E}\flat_5$ was longer it would be dissonant in C major.

5.3 Markov chain music

Now that we have separated pitches from durations, we can begin constructing a Markov chain of pitches. Each state of the Markov chain is a pitch p_k. Each entry in the transition matrix $\mathbf{A}$ will thus be the probability of playing pitch p_k given the last ω pitches played, where ω is the order of the Markov chain. The Markov chain can then be used to generate a new list of pitches, which will be the pitches of the new melody.

For the durations, we construct a Markov chain in a similar fashion; only now the states represent durations. This way we can create a new list of durations, which models the rhythmic figures of the melodies.

We have thus represented the style of a given artist or genre as two Markov chains, one representing the pitches to be used and the second representing the rhythm. From these chains we can generate new vectors of length m of pitches and durations:

$$\begin{aligned} \mathbf{m'}_p &= [p'_1, p'_2, \ldots, p'_m] \\ \mathbf{m'}_d &= [d'_1, d'_2, \ldots, d'_m] \end{aligned}$$

Note that the apostrophe denotes a feature extraction of the input lists. These lists can be merged into the new melody line

$$\mathbf{m}_{new} = [(p'_1, d'_1), (p'_2, d'_2), \ldots, (p'_m, d'_m)]$$

which should sound similar in style as the artists from which the training set of the Markov model was sampled.

5.4 Implementation details

The implementation of the melodic development just described uses the `Data.MarkovChain` library for Haskell, which was introduced in Section 3.3. We start by concatenating all melodic lines from the search result from the database and extract the pitches. Having extracted the pitches from the music data type into a list, we use the `run` function to generate a new list of pitches. The function takes four arguments: The order of the Markov chain ω, the training set, i.e. a list of pitches $\mathbf{m}$, an index i from where to start in the training set, and random generator state s. It then returns a new list of pitches, generated from the training set.

ω, i and s are now three parameters we can adjust freely before generating new melodic lines. The order determines how much the newly generated melody will sound like the melodies in the database. Because we have a rather large database of music, we can also set the order of the Markov chain somewhat high. An order of three or four generates melodies where the artist influence is clearly noticeable. This is depicted in Figure 16. For lower orders, the melodies become more random.

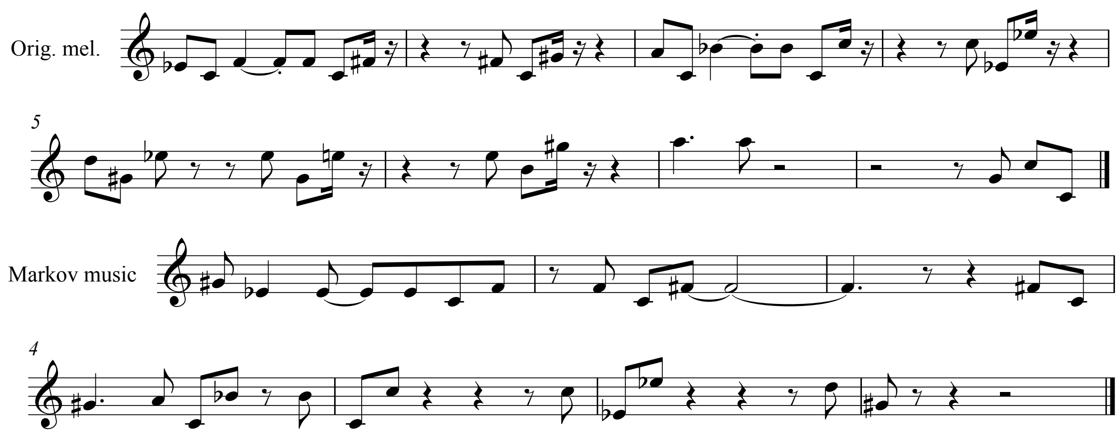


Figure 16: The staff in the top is from original tune "Speak No Evil" composed by Wayne Shorter, where the bottom staff is a tune composed by using the Markov model with $\omega = 4$ for both chains on different artists and all their melodies in the jazz genre. Notice that both the notes and the rhythmic figures for the first 5 bars of the original melody and the 7 bars of the Markov music have many similarities. E.g. E♭-C-F-F-C-F♯ from the first bar in the original tune is similar to the notes in Markov music in the first bar on the eighth note after beat 3.

The consequences of the other two parameters are harder to interpret. The index of the initial pitch determines which note to start with in the new melody but depending on the parameters of the database search, we cannot know anything about the indices of the pitches in the training set. So, the parameters i and s are best interpreted as introducing some randomness in the newly generated melody.

6 Harmonisation

This section will describe in detail how we constructed the first implementation of the harmonisation part of Cremo. In our database, we have divided all tunes into at least two channels: One channel for the melody line and one for the chords. Each chord in the database can be placed at any time within a bar but will primarily be placed at the first and perhaps the third beat in a bar. When we generate the new harmonisation we will place chords at fixed intervals, e.g. at the first beat of each bar or the first and third beat of each bar. This gives some inconsistencies with respect to feature extraction of the melodic content, as the chords will vary in duration. In this first attempt we therefore choose a very simple melodic feature by considering only the two notes immediately above the chord. This feature extraction is very simple and has some drawbacks that we will discuss in this section. We therefore implemented a more proper feature extraction after our prototype user test. This implementation will be explained in Section 8.

6.1 Hidden Markov Models

Following the same observations as Moray Allan [1], we consider the chords as an underlying Markov chain emitting a melody according to a Hidden Markov Model. In the following we will describe the HMM method in details and use the notation from [3]. The idea behind HMM is that we *observe* some data that is *emitted* by an underlying Markov chain (see Section 5.1). Thus, the observed data depends on the states of the underlying variables. Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ be the observed variables. Thus, for each observable variable $\mathbf{x}_n$, we introduce a *latent* (or *hidden*) variable $\mathbf{z}_n$, which can be of a different distribution or dimension than the observable variable, and we introduce $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$. Now it is the $\mathbf{z}_n$'s that constitute a Markov chain and each $\mathbf{x}_n$ is dependent on $\mathbf{z}_n$. Figure 17 shows the structure of a HMM.

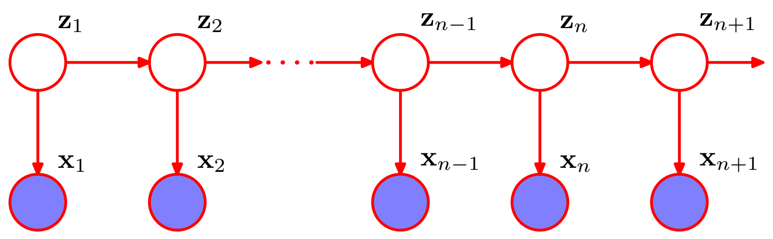


Figure 17: Graphical representation of a Hidden Markov Model

The joint distribution for this model is given by

$$P(\mathbf{X}, \mathbf{Z}) = P(\mathbf{z}_1) \prod_{n=2}^{N} P(\mathbf{z}_n | \mathbf{z}_{n-1}) \prod_{n=1}^{N} P(\mathbf{x}_n | \mathbf{z}_n)$$

Just as in Section 5.1 we will use a 1-of-K coding scheme for the latent variables, where the each element in $\mathbf{z}_n$ corresponds one of the K states in the Markov chain and exactly one of these elements is equal to 1 and all other elements are equal to 0. Because of the Markov property of the latent variables, we have the conditional distribution

$$P(\mathbf{z}_n|\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{n-1}) = P(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Each latent variable has dimension K and we can describe this conditional distribution by a matrix $\mathbf{A}$ consisting of the *transition probabilities* which are given by

$$A_{ij} = P(z_{nj} = 1|z_{n-1,i} = 1) \tag{8}$$

and because they are are probabilities, we have that $0 \leq A_{ij} \leq 1$ with $\sum_j A_{ij} = 1$. According to the Chapman-Kolmogorov equations [10], we can now write the conditional distribution in the form

$$P(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) = \prod_{j=1}^{K}\prod_{i=1}^{K} A_{ij}^{z_{n-1,i}z_{nj}} \tag{9}$$

This is a compact notation and can be elucidated by noticing that the exponent $z_{n-1,i}z_{nj}$ is 1 when both are 1 resulting in the transition probability A_{ij}; otherwise the exponent is zero resulting in multiplying with 1.

The initial latent node $\mathbf{z}_1$ has no parent and so we represent the marginal distribution $p(\mathbf{z}_1)$ by a vector of probabilities $\boldsymbol{\pi}$ with elements

$$\pi_k = P(z_{1k} = 1)$$

Like before, we can write the conditional distribution in the form

$$P(\mathbf{z}_1|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{1k}} \tag{10}$$

where $\sum_k \pi_k = 1$.

We have now defined the role of the latent variables $\mathbf{z}_n$ by their conditional distribution. We now finish the model by doing the same for the observed variables $\mathbf{x}_n$, thus defining the conditional distribution of $P(\mathbf{x}_n|\mathbf{z}_m, \boldsymbol{\phi})$, where $\boldsymbol{\phi}$ is a set of parameters governing the distribution. These are known as *emission probabilities.* We can represent the emission probabilities in the form

$$P(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) = \prod_{k=1}^{K} P(\mathbf{x}_n|\boldsymbol{\phi}_k)^{z_{nk}} \tag{11}$$

where we see that z_{nk} is 1 for the index of $\mathbf{z}_n$ describing the component responsible for generating the corresponding observation $\mathbf{x}_n$, giving the probability of x_n given $\boldsymbol{\phi}_k$. We call the model *homogeneous* if the latent variables share the same parameters $\mathbf{A}$ and all of the emission probabilities share the same $\boldsymbol{\phi}$.

Given the parameters to the entire model $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}\}$, the joint probability is then given by

$$P(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = P(\mathbf{z}_1|\boldsymbol{\pi}) \prod_{n=2}^{N} P(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) \prod_{m=1}^{N} P(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi}) \tag{12}$$

Estimating the parameters of the HMM is done using equation (12). See [3] for the full treatment.

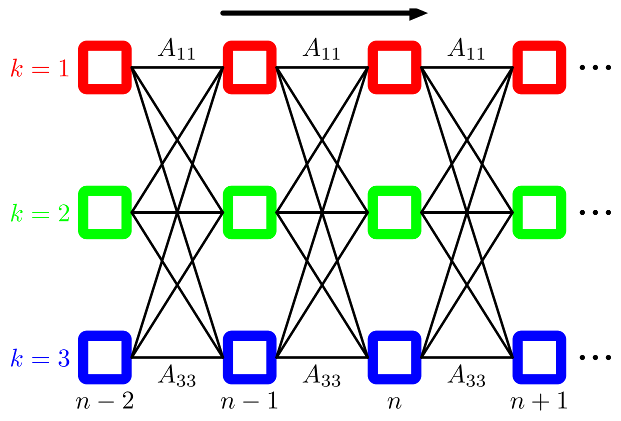


Figure 18: Lattice diagram of a Hidden Markov Model, where the columns correspond to the latent variables $\mathbf{z}_n$ and the rows correspond to states.

The Viterbi algorithm

Given an observation sequence $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, we want to find the most probable sequence of hidden states. Finding the locally most probable state for each node will not in general give the most probable sequence globally because of the dependence between hidden variables. Setting the most probable state in node $\mathbf{z}_n$ may cause all remaining possible states in $\mathbf{z}_{n+1}$ to have very small probability. This may even propagate all the way through the rest of the nodes. Therefore, we need to consider all variables globally and find the states in all nodes that give the overall highest probability. The brute force way of doing this would have to keep track of exponentially many sequences. Figure 18 shows a *lattice* diagram (see [3]) of a HMM with three hidden states. Note that there are connections between all states in two successive time steps. Therefore if we have K states and N latent variables we would have to consider K^N sequences.

The *Viterbi* algorithm is a dynamic programming algorithm that solves this problem more efficiently. Dynamic programming algorithms use the notion of *optimal substructure* [8] to solve optimisation problems. A problem is said to have optimal substructure if the optimal solution to the entire problem contains optimal solutions to subproblems. Assume $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N\}$ is the sequence of latent variables and that we want to find the most probable sequence going through $\mathbf{z}_n$. The idea is to fix a state in $\mathbf{z}_n$ and compute the optimal sequence from $\mathbf{z}_1$ through this state in $\mathbf{z}_n$.

Consider state s_k, that is $\mathbf{z}_n$ where $z_{nk} = 1$ and assume that the most probable sequence from $\mathbf{z}_1$ to $\mathbf{z}_n$ goes through $\mathbf{z}_{n-1}$ in state s_l. The sequence from $\mathbf{z}_1$ to $\mathbf{z}_{n-1}$ must thus be the most probable also. If not, we could replace the sequence with a more probable sequence from $\mathbf{z}_1$ to $\mathbf{z}_{n-1}$ and thereby retrieve a more probable sequence to $\mathbf{z}_n$, which contradicts the assumption of an optimal sequence. Therefore, the solution to finding the most probable sequence $\mathbf{z}_n$ has embedded a solution to the most probable sequence to $\mathbf{z}_{n-1}$. Viterbi thus computes the K optimal sequences for each $\mathbf{z}_n$ in ascending order and maintains a list of these. When this list is computed, it is a simple matter of tracing back the optimal sequence. So, instead of considering all K^N sequences, Viterbi only considers K sequences for each $\mathbf{z}_n$, i.e. KN sequences. Figure 19 shows how the Viterbi algorithm keeps track on only a subset

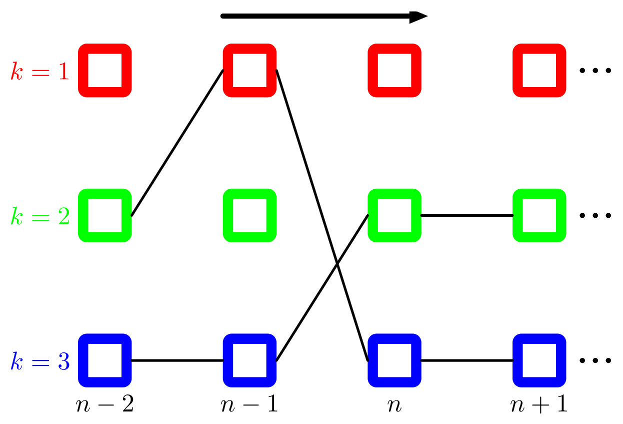


Figure 19: The Viterbi algorithm finding optimal sequences

of all possible sequences.

6.2 Harmonic features

When dealing with real data, it is often necessary to decrease the dimension of the data, so that the most important features are extracted in favour for the less important features. This is typically done by preprocessing the data, hoping that the pattern recognition problem will be easier to solve. In our case, we want to decrease the number of different chords, such that the HMM has the best possibilities of choosing the appropriate chords for a given melody line. In the following, we will describe how we have performed feature extraction for the chords of the database.

To describe how we have chosen our feature extraction for harmonisation, we need to dive into the realm of music theory. Recall from Section 2.2 that in rock and jazz, triads and seventh chords are the basic harmonic units, respectively for the two genres[5], and therefore we will examine the two chord types isolated. We can build triads and seventh chords from diatonic scales, depicted in Table 6 and 7, respectively. These chords are the 'building blocks' for the harmony of most Western music. Extending the chords with the tones depicted under 'Implied Extension' is not changing the harmonic function of the chords but simply 'colours' it.

Chord Type	**Implied Extensions**
Major	9,13, (sus2, sus4)
Minor (m)	9, (sus2, sus4)
+	9
Diminished	Major 7 above any chord tone

Table 6: Triads

[5]The boundary between the two genres is quite fluid, so it is quite normal for especially melodic rock music to borrow aspects from the jazz music.

Chord Type	Implied Extensions	
	more common	less common
maj7	9, 13	+11
6	9	+11
mmaj7	9, 11, 13	–
m6	9,11	–
m7	9,11	13
$^{\varnothing}7$	9,11	♭13
°7	Major 7 above any chord tone	°7 chord Major 7 above original
7	9,13	+11
7 (♭9)	+9, +11	+5 or 13
7 (+9)	♭9, +11	+5 or 13
7 (sus4, ♭9)	13	+9
7 (sus4)	9, 13	3
7 (+5)	9	or +9, ♭9, +11

Table 7: Seventh chords

Chord Type (Major Key Harmony)	Possible chords
I (tonic)	maj7, 6
IIIm (tonic)	m7, m6
VIm7 (tonic)	m7
IIm7 (subdominant)	m7
IVmaj7 (subdominant)	maj7
V7 (dominant)	7, 7 ♭9, 7 +9, 7 (sus4, ♭9), 7 (sus4), 7 (+5)
VII$^{\varnothing}$7 (dominant)	$^{\varnothing}7$

Table 8: Functional harmony chords in major

We can reduce the quantity of chords by only considering the functional harmony of the chords (tonic, subdominant etc.). When categorising the chords by harmonic function, the important notes are the third, the fifth and the seventh scale degrees. Roughly, we can categorise chords as minor and major chords, where we normally think of minor as the aeolean minor scale and major as the ionian major scale. These functional chords are depicted in Table 8 and 9, respectively for major and minor.

Ignoring extensions on dominant chords, we can reduce these chords further as depicted in Table 10. As mention above, these extensions are flavours to the chords and not important for the function of the chords. Though, it should be said that the extensions can enrich the chords dramatically, but as we see it, the skilled musician should be able to abstract from this and by himself be able to add the extensions. This is mostly an issue for the jazz musicians, since in rock music chords are not very often extended to more than four voices.

We should also be able to perform feature extraction on slash chords [21], where the bass note of the chord is not the actual root of the chord, e.g. C/E or Cmaj7/D. In popular music, where the particular arrangement of notes is less important than some other forms, slash chords are generally used only when the specific bass note is important. In jazz this

Chord function (Minor Key Harmonies)	**Possible chord types**
Im (tonic)	m7, mmaj7, m6
♭III (tonic)	maj7, 6, maj7+5, maj7+4
♭VI (tonic)	maj7, 6
VI$^{\varnothing}$	$\varnothing$
II (subdominant)	$^{\varnothing}7$
IVm/IV7 (subdominant)	m7, 7
V (dominant)	7
VII$^{\circ}$ (dominant)	$^{\circ}7$, $^{\varnothing}7$

Table 9: Functional harmony chords in minor

Sixth/Seventh chords	**Triads**
maj7	Major
6	Major
m7	Minor
mmaj7	Minor
m6	Minor
7	Major
7, +5	+
$^{\varnothing}7$	$^{\circ}$ (diminished)
$^{\circ}7$	$^{\circ}$ (diminished)

Table 10: Basic chord types to distinguish between, given that no extensions are used.

Slash chord	**Chord described with respect to the root**
I/II	9sus4
I/III	m♭6
I/IV	maj7 sus2
I/V	6sus4
I/VI	m7
I/♭VII	6♯11add9
I/VII	6♭9sus4

Table 11: Basic major slash chords from the ionian scale.

Slash chord	**Chord described with respect to the root**
Im/II	7♭9sus4
Im/♭III	6
Im/IV	maj7 9omit3
Im/V	9omit3
Im/♭VI	maj7
Im/VII	6add9sus4

Table 12: Basic minor slash chords from the aeolean scale.

could also be the case, but particularly in modern jazz this notation is often used to describe chords that would otherwise be cumbersome to describe with respect to the root. In the case of popular music, the functional aspect of the slash chords will in many cases[6] still be with respect to the root of the chord, since using another note as the bass note is only giving another flavour to the chord. As we see it, this kind of flavour can give some interesting dimensions to the music and therefore we have decided to keep the slash chords and distinguish them from chords with the original bass note. The most common slash chords can be seen in Table 11 for major chords and Table 12 for minor chords.

6.3 Implementation of harmonic features

To ease the task of simplifying the chords, we have chosen to represent the chords as the absolute pitch of the bass note together with the relative distances from that note. We denote the notes from C to B by integer values from 0 to 12, with C (prime) corresponding to 0, C♯/D♭ (minor second) corresponding to 1, and so on, and together with the absolute bass note, this notation is unambiguous, implying that we can reconstruct the chords in the usual representation recognised by Euterpea[7]. As mentioned in the previous section, we can distinguish the function of the chords by only looking at the third, fifth and seventh, though in absence of the seventh, it can be necessary to consider the sixth, if present. For simplicity and for minimising the number of chords available, we will not consider the sixth. Now, it is an easy task of extracting the desired notes in the chords. Let H_k be a set of relative

[6]This would be the case for all the music in our database

[7]We can not reconstruct the original voicing, but we do not think that this is important since the goal are not to arrange the chord notes, i.e. create interesting voicings, but only to find interesting chord types for the melody.

Slash chord	**Relative representation**
I/II	$\{2, 5, 10\}$
I/III	$\{3, 8\}$
I/IV	$\{2, 7, 11\}$
I/V	$\{5, 9\}$
I/♭VII	$\{2, 6, 9\}$
I/VII	{1,5,7}

Table 13: Relative representation of major slash chords.

distances to the bass note of the k'th chord, and let E be the set containing the notes we want to extract. Then we can get the simplified chord S_k as

$$S_k = H_k \cap E \tag{13}$$

We will use $E = \{3, 4, 6, 8, 10, 11\}$, since by using this set, all the important notes from the chords depicted in Table 10 can be extracted, except slash chords and chords with a sixth. Normally, we will not encounter a ♭7 and a maj7 in the same chord, nor a minor and major third on non-dominant chords, whereas on dominant 7 chords, both a major third and a ♯9[8] and/or a ♭5 and a ♯5 can occur simultaneously in the same chord, but for simplicity these chords are not futher reduced[9]. Notice that we only extract the extensions ♯11/♭5 and ♯5 but not the natural fifth. This is due to the fact, that the natural fifth, in opposite to the ♭5 and ♯5, is not important for the perception of the chord quality.

For example, from the chord $\{2, 4, 7, 10\}$, corresponding to a major 9 chord (the ♭7 note is implicit included), the important notes are the third and the seventh. By using $E = \{3, 4, 6, 8, 10, 11\}$, we get

$$\begin{aligned} S_k &= \{2, 4, 7, 10\} \cap \{3, 4, 6, 8, 10, 11\} && (14) \\ &= \{4, 10\} && (15) \end{aligned}$$

By this method we get a simpler chord without the ninth, which is not important for the harmony.

For slash chords we need another approach, since we can not immediately discover the important notes from the bass note. We will distinguish between the major and minor slash chords from Table 11 and 12. We would like to simplify slash chord to these simple forms, even if the encountered chord is extended. Using the relative representation from before, we get the simplified forms in Table 13 and 14. We have not included the I/VI, Im/III and Im/VI, since these are identical with m7, 6 and maj7, respectively, from Table 10, and therefore these chords can be analysed by using the previous method.

We observe that the minor third, with respect to the bass note, is only present in I/III. Because we have the important ♭6, we can distinguish it from the chords processed in the previous method, and so, when we observe a minor chord with the ♭6 present, we classify the

[8] If we have no major third, then the ♯9 would instead be the minor third of the chord, resulting in a minor chord instead of a dominant 7 chord.

[9] The melody might be closely related to these alterations, so keeping these chords will support the melody. On the contrary, the number of chords are not reduced as much as possible.

Slash chord	Relative representation
Im/II	$\{1, 5, 10\}$
Im/IV	$\{2, 7, 10\}$
Im/V	$\{2, 7, 10\}$
Im/VII	{2,5,9}

Table 14: Relative representation of minor slash chords.

chord as a I/III chord. When no minor third is present, we take the set difference between the chord and every instance of the slash chord in Table 13 and 14, and we classify the chord according to the smallest set difference.

Mathematically, this can be written as follows: Let again H_k be a set of relative distances to the bass note of the k'th chord, and let S_m be the set of each slash chord m represented in Table 13 and 14. Then the index for chord S_m, for which the set difference between H_k is smallest, can be found by

$$\text{index}_k = \arg\min_m (H_k \setminus S_m \cup S_m \setminus H_k) \ , \ m = 1, 2, \dots, M \tag{16}$$

where M is the number of different classifications of slash chords.

For example, if we encounter the chord Cmaj7/D with the representation $\{2, 5, 9, 10\}$ respective to bass note C, we get

$$\begin{aligned}
\{2,5,9,10\} \setminus \{2,5,10\} \cup \{2,5,10\} \setminus \{2,5,9,10\} &= \{9\} \\
\{2,5,9,10\} \setminus \{3,8\} \cup \{3,8\} \setminus \{2,5,9,10\} &= \{2,3,5,8,9,10\} \\
\{2,5,9,10\} \setminus \{2,7,11\} \cup \{2,7,11\} \setminus \{2,5,9,10\} &= \{5,7,9,10,11\} \\
\{2,5,9,10\} \setminus \{5,9\} \cup \{5,9\} \setminus \{2,5,9,10\} &= \{2,10\} \\
\{2,5,9,10\} \setminus \{1,5,8\} \cup \{1,5,8\} \setminus \{2,5,9,10\} &= \{1,2,8,9,10\}
\end{aligned}$$

and we see that the chord $\{2, 5, 10\}$ is the nearest one, corresponding to the slash chord I/II, which is resolved to the absolute chord C/D if the bass note D is considered.

6.4 Harmonising using HMM

Having proper features for the harmonic and melodic contents of our music data type, we are now ready to train the HMM and generate new harmonisations. We can view this as a classification problem. Let us assume that we are given an input melody $\mathbf{m}_I$, represented as a list of pitches and durations defined as

$$\mathbf{m}_I = [n_1, n_2, \dots, n_n]$$

where $n_k \in \mathrm{N}$ for $k = 1, \dots, n$. We want to place a chord at each bar of this melody, which corresponds to creating a list of chords $\mathbf{h}_I$ defined as

$$\mathbf{h}_I = [\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m]$$

where $\mathbf{c}_k$ is a list of notes corresponding to the possible chords and where $D(\mathbf{c}_k) = d_h$ for all k. Since we make sure that each chord is boiled down to its function by removing extensions and because the same chords are voiced identically, we can uniquely identify each chord by its list representation. Therefore, these lists act as classes of chords. So, given d_h, we want to classify each time period of length d_h in the input melody with a chord class. For the case of $d_h = 4/4$ and where the input melody has time signature 4/4 we have to classify each measure in the input melody with a specific chord.

Training

The training of the HMM consists of extracting features from the database and letting the HMM compute the probabilities of each classification. In the training set (the database) we do not have chords placed at specific time instants with fixed duration, so we extract features by looking at the two notes above each chord, as long as the duration of the first note does not extend beyond the next chord. In order to avoid too much notation, we assume now that we look at one specific part of a song in the database and call that $\mathbf{m}_{db}$, and that $\mathbf{c}_k$ is the k'th chord in this song. We assume that $\mathbf{c}_k$ has already been feature extracted according to the methods described in Section 6.2. Let $d_k = D(\mathbf{c}_k)$ be the duration of the chord and $d_{1,k-1}$ be the sum of the durations of the chords $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{k-1}$, that is the duration of all chords up until the k'th, which is equal to the duration of the part up until this chord.

Now, let $p_{(k,1)}$ denote the first note above the k'th chord, such that

$$p_{(k,1)} = p_j \ , \text{ where } (p_j, d_j) \in \mathbf{m}_{db} \ , \ j = \arg\min_x \left(\sum_{n=1}^{x} d_n \geq d_{1,k-1} \right) \tag{17}$$

Thus, the second note above the chord is simply $p_{(k,2)} = p_{j+1}$. If the duration of $p_{(k,1)}$, denoted $d_{(k,1)}$, exceeds the duration of the underlying chord, we do not consider the next note, as it will belong to the next chord and instead we interpret the note $p_{(k,1)}$ as being played twice. This gives rise to the following melodic feature for the k'th chord:

$$f_{m,k} = \begin{cases} (p_{(k,1)}, p_{(k,1)}) \text{ if } d_{(k,1)} \geq d_k \\ (p_{(k,1)}, p_{(k,2)}) \text{ otherwise} \end{cases} \tag{18}$$

This corresponds to the observable variables $\mathbf{x}_n$ in Section 6.1 whereas the $\mathbf{c}_k$'s correspond to the latent variables $\mathbf{z}_n$.

Thus, we will have a melodic feature vector $\mathbf{m}'_{db}$ and the classification pairs $\mathbf{C}_{db}$ given by

$$\begin{aligned} \mathbf{m}'_{db} &= [f_{m,k} \mid k = 1, 2, \ldots] \qquad (19) \\ \mathbf{C}_{db} &= [(f_{m,k}, \mathbf{c}_k) \mid k = 1, 2, \ldots] \qquad (20) \end{aligned}$$

This classification is thus used to train the HMM with $f_{m,k}$ as the observable states and $\mathbf{c}_k$ as the hidden states. The parameters of the HMM is estimated from this training set using equation (12).

Harmonising

Having estimated the parameters of the HMM on the basis of the database, we are now ready to harmonise an input melody. To do this, we first set the duration of the chords, say d_h, and then extract the melodic features from the input melody, corresponding to where the chords

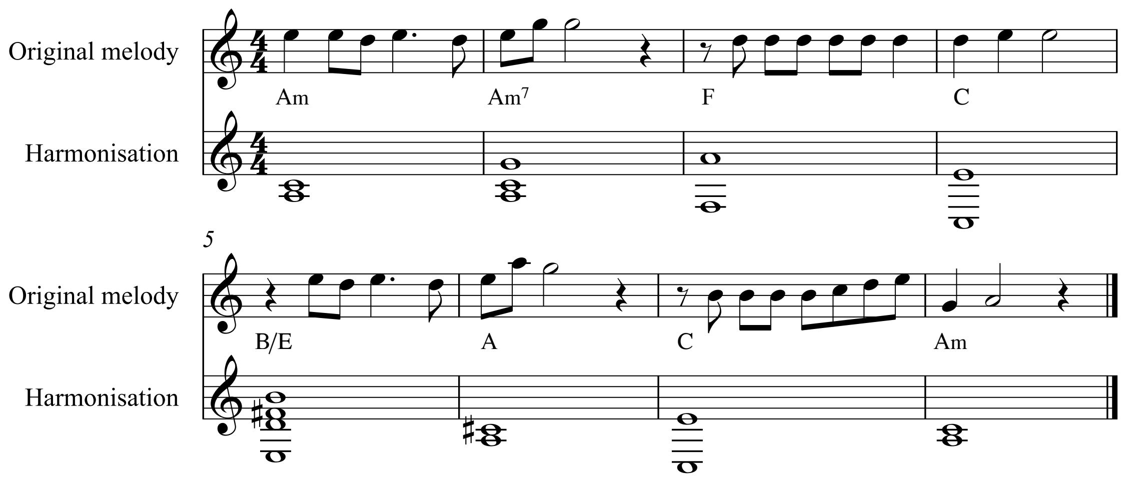


Figure 20: First 8 bars of the melody from "Burma" composed by Börsenfiber together with generated chords on the basis of all the artists in the melodic rock genre.

will be placed, i.e. extract the two notes immediately above the placement of the chord as described above. Mathematically, we can describe this as follows. Let $\mathbf{m}_I$ be a list of notes and durations, corresponding to the input melody, say

$$\mathbf{m}_I = [(p_1, d_1), (p_2, d_2), \ldots, (p_n, d_n)]$$

We now need to harmonise this input melody, i.e. make a classification of each section of length d_h, thereby creating a list of chords for the input melody:

$$\mathbf{h}_I = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m]$$

Since we have chosen the HMM model for this classification, we just need to run the Viterbi algorithm using the features of the $\mathbf{m}_I$. These features are given by equations (18) and (19), only now all chords have the same duration, namely d_h.

As described in Section 6.1, the Viterbi algorithm is a dynamic programming algorithm for finding the most probable sequence of hidden states in the HMM. We can see this as finding the most probable harmonisation of the input melody using chords from the database only and since the parameters of the HMM are set according to the contents of the database, we can interpret this as harmonising in the style of the artists in the database.

In Figure 20 the first 8 bars of the melody in "Burma", composed by Börsenfiber, is depicted. We see the original melody in staff 1 and the generated chords in staff 2.

6.5 Implementation details

Using Euterpea we convert a melodic line into a list of pitches and durations determined by the `Primitive` type. Likewise we convert the chord line into a list of lists of pitches and durations. We then run through both of these lists sequentially and extract the features as described in the previous subsections. The overall structure of the harmonisation algorithm

```
HARMONISATION1 $\mathbf{m}_I$ $\mathbf{m}_{db}$ $\mathbf{h}_{db}$ $d_h$
let
    - - Training
    $\mathbf{m}'_{db}$ = EXTRACTFEATURES1 $\mathbf{m}_{db}$ $\mathbf{h}_{db}$
    $\mathbf{h}'_{db}$ = SIMPLIFYCHORDLINES $\mathbf{h}_{db}$
    $states$ = ZIP $\mathbf{m}'_{db}$ $\mathbf{h}'_{db}$
    $hmm$ = HMM.TRAIN $states$
    - - Harmonising
    $\mathbf{m}'_I$ = EXTRACTINPUTFEATURES1 $\mathbf{m}_I$ $d_h$
    $\mathbf{h}'_I$ = HMM.BESTSEQUENCE $hmm$ $\mathbf{m}'_I$
    $\mathbf{h}_I$ = REBUILDCHORDNOTES $\mathbf{h}'_I$ $d_h$
in
    $(\mathbf{m}_I, \mathbf{h}_I)$
```

Figure 21: Harmonisation – prototype version

is seen in pseudo-code in Figure 21. EXTRACTFEATURES1 extracts the features as described above from the melody and chord lines of a given part of a song in the database. It is described in detail below and seen in Figure 22. The SIMPLIFYCHORDLINES function extracts features as described in Section 6.3. The melodic and harmonic features are then zipped together to form a list of pairs, that correspond to the pairs of observable and hidden states. This list is used by the HMM.TRAIN function to set the parameters of the HMM. These parameters are stored in the variable *hmm*. For the actual harmonisation, EXTRACTINPUTFEATURES1 works just as EXTRACTFEATURES1, except that it uses the static chord duration d_h, which is the duration of the chords we want to place under the melody. HMM.BESTSEQUENCE then takes the input melody features and calculates the best sequence of chords using the Viterbi algorithm. REBUILDCHORDNOTES take the list of lists of pitch classes $\mathbf{h}'_I$ generated by the Viterbi algorithm and make real notes out of the them by adding an octave number (which is set globally set) and the duration d_h to all pitch classes. So, $\mathbf{h}_I(k) \in \mathrm{N}$ for all k.

The EXTRACTFEATURES1 function works as follows: The first two notes are extracted from the melody list of the database and the first chord is extracted from the chord list. RESTMELODY cuts the beginning of the list corresponding to the duration given as the first argument. If the note durations do not add up exactly to this duration, the note that exceeds the duration is cut in two. This is used to find the rest of the melody beginning at the same time as the next chord. EXTRACTFEATURES1 is thus called recursively, adding melodic features to a list. This corresponds to calculating equation (18). The function terminates when $\mathbf{m}_{db}$ has less than two elements or when $\mathbf{h}_{db}$ is empty.

The greatest weakness of the Euterpea library is that the `Music` type only consists of notes and rests as well as sequential and parallel composition of these. Common musical information such as bars and time signature is missing. This turned out to be a major problem when implementing the HMM harmoniser. In order to find exactly the notes above each chord we have to keep track of durations in boths lists as seen in the EXTRACTFEATURES1 function. This is very error prone.

```
ExtractFeatures1 $\mathbf{m}_{db}$ $\mathbf{h}_{db}$
let
    $(p_1, d_1) =$ head $\mathbf{m}_{db}$
    $(p_2, d_2) =$ head (tail $\mathbf{m}_{db}$)
    $\mathbf{c}_1 =$ head $\mathbf{h}_{db}$
    $\mathbf{m}_{db,r} =$ RestMelody $D(\mathbf{c}_1)$ $\mathbf{m}_{db}$
    $\mathbf{h}_{db,r} = \mathbf{h}_{db} \setminus \mathbf{c}_1$
in
    if $d_1 > D(\mathbf{c}_1)$
        $(p_1, p_1) \cup$ ExtractFeatures1 $\mathbf{m}_{db,r}$ $\mathbf{h}_{db,r}$
    else $(p_1, p_2) \cup$ ExtractFeatures1 $\mathbf{m}_{db,r}$ $\mathbf{h}_{db,r}$
```

Figure 22: Database feature extraction – prototype version

7 Prototype user test

Due to the complexity of the nature of music and the individual taste and subjective opinion of humans, we can not evaluate the quality of the music created by Cremo by a quantitative test categorising the output as correct or wrong. So, in order to validate our efforts we have conducted interviews and user tests with several musicians with knowledge in music theory and who composes music themselves. We have conducted two separate user tests, one for the prototype of Cremo, which we will go through in this section, and one for the final version of Cremo, which we will treat in Section 10.

7.1 Goals of test

The goal of our first user test is to give us a rough idea of whether our prototype system works according to our synopsis description. We have chosen to let four musicians test Cremo; two in the standard jazz genre and two in the melodic rock genre. The goal of the user test is to test the harmonisation and melody generation with respect to three aspects:

1. The objective quality of the harmonisation and melody generation. With objective quality we mean "how well" or "how correct" Cremo is harmonising the input melody and generating melodies with respect to dissonance, tension/release etc. without considering subjective aspects as taste or genre.

2. Whether the created harmonisation and melody is within the given genre, i.e. could the cadences be from the melodic rock or standard jazz genre or are they atypical for the genre.

3. Whether the output from Cremo is inspiring and could it be useful when composing new songs. This is the subjective view of Cremo, and should be answering whether Cremo could enhance the creativity of a composer.

To find musicians with background and knowledge suitable for evaluating Cremo, we set up criteria that the test person should fulfill. The selection of musicians was done by letting the test person answer a *preprocessing questionnaire* which gave us the needed information about the following skills:

1. Should have some knowledge in music theory judged from a series of five theoretical questions of increasing difficulty. For rock musicians two questions has to be answered correctly. In addition, the jazz musician has to answer at least one of the last three questions correctly.

2. Should be a composer and have composed at least 10 songs.

3. Should play or have played in at least one band.

4. Should have played their main instrument for at least 10 years.

5. Should be able to describe the style of at least 3 artists in the rock genre or 4 artists in the jazz genre with a few words[10].

[10]Artists in the database, of course.

More details about these thresholds can be seen in Appendix B. The reason for 1) is that it should be easier for the test person to give more precise answers by pick point aspects that he likes or dislikes. Furthermore, if the test person knows the theory, he should have a reasonable commitment to music in general, since music theory for its own sake should not be very common. On the contrary to rock music, it is necessary to know music theory to be a skilled improviser and composer in jazz, therefore we have a higher threshold in 1) for the test persons evaluating standard jazz genre rather than for test persons evaluating the melodic rock genre. We clearly need 2) to be fulfilled, since a test person that is not composing music should not be able to compose a melody line for Cremo, nor answer whether Cremo's compositions are inspiring. 3) and 4) tell us something about the general musical experience of the musician, which is important for evaluating music. 5) is also important, since the harmonisation and melody are created on the basis of the database, and therefore 2) in the listing for the goals for the user test can not be answered. Though, it is reasonable to constraint the test person to have a knowledge about the artist in the database. Point 3) in the listing for the goals for the user test may not be answered with the necessary knowledge, since the output is indeed determined by the genre, and therefore the test person either dislikes the genre, giving a biased result, or is simply not aware of the genre. In the latter, both a positive and a negative answer should be possible.

A thorough test of Cremo would require 1) a wide range of musicians, 2) the ability for the musicians to adjust parameters, 3) a registration of their parameter choices and 4) the evaluation of the outcomes for each of these choices. Due to a different main focus and time constraints, we have decided to conduct the test as a combination of an interview and a questionnaire survey with four musicians which we believe to be the easiest and most fruitful method for evaluating these outputs.

7.2 The test

We started the interview by explaining the goal of the project as defined in the synopsis. For the actual evaluation of Cremo, we chose three of the artists from the database that the test person knows the best according to preprocessing questionnaire, and from a list of all the songs in the database with these artists, the test person was told to choose one song that he would like to hear, to become accustomed to the MIDI sound and format. He also heard his own melody in the proper Cremo MIDI format.

We created four different harmonisations of the input melody and generated one new melody for individual test persons, depending on the knowledge of artists in our database, so that we can make sure that the output should be reminiscent of an artist that is known to the test person.

We then gave him the *evaluation questionnaire* and left him alone to listen to the harmonisations and melodies as many times he wished and let him answer the questions.

The evaluation protocol can be summarised as follows:

1. Generate harmonisations and a melody for the individual test persons with parameters seen in Appendix B.

2. Let the musician listen to his own melody and a song of a known artist from the database, both in MIDI format, to get accustomed to the MIDI sound.

3. Explain what he is about to listen to and how the questionnaire is build up.
4. Leave the room and let the musician listen thoroughly to the different harmonisations and melody while answering the evaluation questionnaire.
5. Collect the questionnaire, and ask elaborative questions.

7.3 Results

In this section, we summarise the answers from the evaluation questionnaire. In summarising, be aware of the following issues:

- Four musicians are far too few to make universal conclusions.
- The order of the generated music that the musicians evaluate can influence the overall evaluation. For example, if the musician found the first harmonisation to be very dissonant and unmusical, the rest of harmonisation might be evaluated relatively to this experience, i.e. giving a skewed overall result.
- The test person might feel bad about answering critically, and therefore answers more positively than he really feels.
- The sound of MIDI do not sound very natural and if the test person is not used to the sound, it can influence the perception of the generated music.

All answers from the multiple choices considering harmonisation and melody generation for the melodic rock genre can be seen in Table 15 and Table 16, respectively, and the answers for the user test considering harmonisation and melody generation for the standard jazz genre can be seen in Table 17 and Table 18, respectively. These tables are read as follows: Each row is a question and the columns labeled **A** to **E** are the multiple choice answers. These answers are different for the various questions, so each question row is divided into two: The top row stating the question along with the answer alternatives and the bottom row stating the results for each answer alternative. The numbers are the number of harmonisations (or melodies in the case of Table 16 and 18) that have been given the particular answer to the particular question. Since we have tested on two rock musicians and two jazz musicians, giving them four harmonisations and one melody each, we have eight harmonisations and two melodies per genre.

We will evaluate the data for standard jazz and melodic rock separately, since the results differ for the two genres, as we will see. Questions 2.1-2.7 deal with the harmonisation without considering the melody, whereas Questions 2.8-2.10 deal with the harmonisation together with the melody. Questions 3.1-3.6 deal with the generation of new melodies.

For the melodic rock genre, we see in Question 2.1 from Table 15 that 7 out of 8 harmonisations have more or less reasonable structure; 3 harmonisations have really good coherence between chords (D), where 1 harmonisation has very bad coherence (A). The negative remarks can be summarised as the chords being either too stochastic and or too monotonous and having generally weird structure. On the positive side, they state that the periods make sense and chords have direction. We see the same pattern for Question 2.3. Since the distribution of the answers to these questions does not have a clear mean, we can not conclude whether Cremo makes reasonable cadences. The answers to the same questions for the jazz genre

Question	**A**	**B**	**C**	**D**	**E**
Q2.1: How does the harmonisation of Cremo sounds without the melody?	No coherence	Some	Good	Very good	
Answers	1	3	1	3	-
Q2.3: How would you describe the variation of the relation of the chords?	No variation	Some	Medium	Good	Very good
Answers	1	2	1	2	2
Q2.4: Are the variation of the chords appropriate for the genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	4	1	3	0
Q2.5: Are the types of the chord progressions representative for the given genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	3	4	1	0
Q2.7: Could you use the generated chords as a source of inspiration for a new melody?	Yes	No	Maybe		
Answers	3	4	1	-	-
Q2.8: How does the harmonisation of Cremo sound together with the melody?	Chords don't fit	Fit somewhat	Fit well	Fit very well	
Answers	2	5	1	0	-
Q2.9: Could you use the generated chords as a source of inspiration for new chords for your melody?	Yes	No	Maybe		
Answers	2	4	2	-	-
Q2.10: Could you use the generated chords directly as a basis for your melody?	Yes	No	Maybe		
Answers	0	7	1	-	-

Table 15: Multiple choice summary for harmonisation generation in the melodic rock genre. Notice how Q2.1 and Q2.3 have no clear mean and that 7 of 8 harmonisations cannot be used as a direct basis for the melody.

are more centred and the respondents remark that the harmonisations are monotonous and lack direction, yet sometimes also real melodic. So, again Cremo is not consistent in creating either good or bad harmonies. Whether the chords are representative for the genre (Question 2.5), we see that there is a rather clear mean at C ("to some extend") in both genres.

It is interesting to note that a majority of the rock musicians answer, that they can not use the chords as a source of inspiration for new melodies, whereas in 7 out of 8 of the jazz harmonisations, the jazz musicians answer that they could use the chords as a source of inspiration.

Question	**A**	**B**	**C**	**D**	**E**
Q3.1: Are there aspects of the melody that sounds musical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	0	2	0	0
Q3.2: Are there aspects of the melody that sounds unmusical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	0	2	0	0
Q3.4: Can you recognize one or more artists in the generated melody?	Yes	No	Maybe		
Answers	0	0	2	-	-
Q3.6: Could you use the melody as a source of inspiration?	Yes	No	Maybe		
Answers	0	2	0	-	-

Table 16: Multiple choice summary for melody generation in the melodic rock genre. Clearly, both musical and unmusical elements are present and artists are recognisable. Only one of the four respondents could use the melody as a source of inspiration (see Table 18)

When the chords are played together with the melody, we see a clear pattern in both genres: The chords either fit somewhat with some wrong notes compared to the melody or the chords do not fit at all. This is supported by the remarks, from which we emphasise the following:

> "In several of the passages, the melody fits the harmony with respect to music theory, i.e. the notes are ok, but the musical expression is not very strong. Several places, the harmonisation hurts the ears a little."
> *Respondent 2*

> "Besides very few bars, it seems like a struggle between chords and melody."
> *Respondent 2*

and

> "It sounds generally weird together with the melody but there are some good ideas."
> *Respondent 1*

These quotes speak for themselves. In general, the harmonisations are very dissonant and do not follow the melodies. Some more constructive remarks are

> "Seems as if it is not capable of finding the 'stations' of the melody and it has trouble distinguishing between passage notes and chord notes."
> *Respondent 4*

and

"Melody seems too diatonic in comparison with the chords and Cremo can't read the dissonances."
Respondent 4

This also manifests itself in that in all but one harmonisation the respondents have answered that they can not use the harmonisation directly for their melody.

For the generated melody, we see that most respondents agree in that there are both musical and unmusical aspects of the melody and the artists from which the melodies are generated are recognisable (perhaps too recognisable?). Only one of the respondents could use the melody as inspiration for new melodies. The following three remarks sum up the general opinion:

"Overall, the melody seems somewhat random and unmusical in the sense, that no musical universe is built up in which the melody makes sense."
Respondent 3

"Some good fragments and ideas, but lacks coherence and direction."
Respondent 4

"Somewhat too incoherent melody as a whole, but some good lines."
Respondent 1

7.4 Conclusion and further work

As can be seen from the above analysis of the user test results, the harmonisations alone are often reasonable and the chord progressions are somewhat representative for the genre. Together with the melody the harmonisations fail as it introduces a lot of dissonances and the harmonisations seem unnatural for the melodies.

It turned out that during the day of testing we discovered a programming error in our code, that caused the feature extraction to be flawed. This could very well be one source of the dissonant harmonisations. Unfortunately, we didn't have time to fix this error before the scheduled tests began. Also, our feature extraction for the harmonisation may be too simple to generate proper harmonisations. Only considering the two notes above a chord does not tell us much about the harmonic basis of the melody. These two notes could be part of a fast, chromatic sequence perhaps, from which no tonal centre can be read. This may very well produce dissonant harmonies with respect to notes in the rest of the bar.

For the generation of new melodies we notice that in both genres, there are both musical and unmusical aspects and the comments from the musicians tell us that the melodies are simply too random to be considered as proper musical melodic sequences as they lack structure and often copy melodic lines directly from the database. The latter can be altered by setting the order of Markov chain down. Since the Markov chains only consider 1-4 notes back in time, they do not capture higher level structure of melodic sequences. Markov chains themselves are not capable of capturing such higher level structure of the melody. But since variation of the input melody is actually more important part of our project than melodic generation, we choose to proceed with developing a more advanced method for varying the input melody.

Thus, having fixed the programming errors that turned up in the course of the tests, we turn to further develop Cremo by

- improving the harmonisation
- implementing a melodic variation algorithm
- test these new implementations and thus the final version of Cremo

Question	A	B	C	D	E
Q2.1: How does the harmonisation of Cremo sounds without the melody?	No coherence	Some	Good	Very good	
Answers	0	2	5	1	-
Q2.3: How would you describe the variation of the relation of the chords?	No variation	Some	Medium	Good	Very good
Answers	0	2	2	4	0
Q2.4: Are the variation of the chords appropriate for the genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	3	3	1	1	0
Q2.5: Are the types of the chord progressions representative for the given genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	1	6	1	0
Q2.7: Could you use the generated chords as a source of inspiration for a new melody?	Yes	No	Maybe		
Answers	7	1	0	-	-
Q2.8: How does the harmonisation of Cremo sound together with the melody?	Chords don't fit	Fit somewhat	Fit well	Fit very well	
Answers	5	2	1	0	-
Q2.9: Could you use the generated chords as a source of inspiration for new chords for your melody?	Yes	No	Maybe		
Answers	2	4	2	-	-
Q2.10: Could you use the generated chords directly as a basis for your melody?	Yes	No	Maybe		
Answers	0	8	0	-	-

Table 17: Multiple choice summary for harmonisation generation in the standard jazz genre. Notice that a clearer mean is present in Q2.1 and Q2.3 than in Table 15 and that 7 of 8 harmonisations could be used for new melodies whereas all harmonisations are deemed unusable as a basis for the melody.

Question	**A**	**B**	**C**	**D**	**E**
Q3.1: Are there aspects of the melody that sounds musical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	0	1	1	0
Q3.2: Are there aspects of the melody that sounds unmusical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	0	0	0	2	0
Q3.4: Can you recognize one or more artists in the generated melody?	Yes	No	Maybe		
Answers	2	0	0	-	-
Q3.6: Could you use the melody as a source of inspiration?	Yes	No	Maybe		
Answers	1	1	0	-	-

Table 18: Multiple choice summary for melody generation in the standard jazz genre. Notice how one respondent could use the melody as a source of inspiration

8 Harmonisation - final version

We will now describe the final version of the harmonisation part of Cremo. It is still based on the Hidden Markov Model, so recall from Section 6, that we estimate parameters by feeding features derived from the database to the HMM. The melodic features are the observable states and the harmonic features are the hidden states. When trained we feed the HMM with the features of the input melody and use the Viterbi algorithm to generate the most probable harmonisation.

Considering only the two notes above the chord proved too simple a feature for the melody, so instead we thought of looking at downbeat notes for a certain duration. For example, we could quantise the melodic line into the four beats per measure and extract the pitch classes of the notes at these time instants. Having extracted this info we look at the melodic content without the contour and matches this against the database; that is, we perform a simple distance calculation between the input feature and the melody features of the database.

8.1 Melodic features

Given a melody from the database $\mathbf{m}_{db}$ and the corresponding chords $\mathbf{h}_{db}$

$$\mathbf{m}_{db} = [n_1, n_2, \ldots, n_n] \tag{21}$$

$$\mathbf{h}_{db} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m] \tag{22}$$

where $n \geq m$, we need to extract features from the melody for each chord. In this version of the harmonisation we extract the notes at specific beats for a specific duration, so we introduce these two parameters:

- The quantisation resolution $q \in \mathcal{D}$, which determines the interval at which we choose to quantise. Usually, we look at either every quarter note ($q = 1/4$) or every eighth note ($q = 1/8$).
- The analysis duration $a \in \mathcal{D}$, which is how far ahead from the chord we pick the notes. Usually, we analyse a whole measure ($a = 4/4$) or a half measure ($a = 2/4$).

These parameters must be chosen so that $a = kq$ for $k \in \mathbb{N}$. First of all we need to extract a subset of length a of the melody above the chord. The subset $\mathbf{m}_k$ of length a/q above chord $\mathbf{c}_k$ can be calculated by first quantising the entire melody according to q:

$$Q_q(\mathbf{m}_{db}) = [n'_1, n'_2, \ldots, n'_{n_q} | n'_k = (p'_k, q)] \tag{23}$$

where n_q is now the length of the quantised melody. Note that durations are identical, $D(\mathbf{m}_{db,q}) = D(\mathbf{m}_{db})$, but that the lengths of the lists in general are not. This quantisation can be calculated recursively as

$$Q_q(\mathbf{m}_{db}) = [(p_1, q)] \cup Q_q(\hat{\mathbf{m}}_{db}) \tag{24}$$

where $\hat{\mathbf{m}}_{db}$ is cut to have length $D(\mathbf{m}_{db}) - q$. It is cut from the beginning of the list as

$$\hat{\mathbf{m}}_{db} = \begin{cases} [n_2, n_3, \ldots, n_n] & \text{, if } d_1 = q \\ [n^c_k, n_{k+1}, \ldots, n_n] & \text{, if } d_1 < q \\ [(p_1, d_1 - q), n_2, \ldots, n_n] & \text{, if } d_1 > q \end{cases} \tag{25}$$

where the note n_k^c is chosen as

$$n_k^c = \begin{cases} n_k & \text{, if } D([n_1, n_2, \dots, n_{k-1}]) = q \\ (p_k, q - d_k) & \text{, if } D([n_1, n_2, \dots, n_k]) > q \text{ and } D([n_1, n_2, \dots, n_{k-1}]) < q \end{cases} \tag{26}$$

Now that we have quantised the entire melody we can easily extract the beat notes above chord $\mathbf{c}_k$ simply as

$$\mathbf{m}_{db,k} = [n'_i, n'_{i+1}, \dots, n'_{i+a/q-1}] \tag{27}$$

where i is chosen such that

$$D([n'_1, n'_2, \dots, n'_{i-1}]) = \sum_{n=1}^{k-1} d_{\mathbf{c}_n} = d_{1,k-1} \tag{28}$$

Note that if $\mathbf{c}_k$ has a duration less than a, there will be an overlap of beat notes between $\mathbf{m}_{db,k}$ and $\mathbf{m}_{db,k+1}$. But we have to keep the analysis duration a fixed in order to get consistent features for each chord. Having $\mathbf{m}_{db,k}$ we further reduce the dimensionality by removing all information except pitch class. Thus, we have

$$\mathbf{m}'_{db,k} \;=\; [p'_i, p'_{i+1}, \dots, p'_{i+a/q-1}] \tag{29}$$

Likewise for the each $\mathbf{c}_k$ obtaining $\mathbf{c}'_k$, we thus have two lists of pitch classes and we feed the pairs $(\mathbf{m}'_{db,k}, \mathbf{c}'_k)$ to the HMM and estimate its parameters as described in Section 6.

8.2 Nearest neighbour

Given the input melody, we need to extract melodic features similar to those in the database. This is done in exactly the same way as described above, except we do it for each time instant d_h which is the chosen duration of the chords we wish to place in the input melody. For consistency between features we use the same q and a as for the training. We define $\mathbf{m}'_{I,k}$ as the k'th quantised part of the input melody of length a, and the Viterbi algorithm could then be run the directly and retrieve a harmonisation. But in order to help Viterbi with the harmonisation we perform a nearest neighbour transformation of $\mathbf{m}'_{I,k}$ in which we map it to a melodic feature existing in the database.

We calculate the distance $\delta_{k,l}$ between $\mathbf{m}'_{I,k}$ and $\mathbf{m}'_{db,l}$ as

$$\delta_{k,l} = |(\mathbf{m}'_{I,k} \setminus \mathbf{m}'_{db,l}) \cup (\mathbf{m}'_{db,l} \setminus \mathbf{m}'_{I,k})| \tag{30}$$

and we define the nearest neighbour for $\mathbf{m}'_{I,k}$ as the $\mathbf{m}'_{db,l}$ where $\delta_{k,l}$ is smallest, i.e. the nearest neighbour is the melody with most pitch classes in common:

$$\mathbf{m}''_{I,k} = \mathbf{m}'_{db,l} \text{ , where } l = \arg\min_x(\delta_{k,x}) \tag{31}$$

We then run the Viterbi algorithm on $\mathbf{m}''_{I,k}$, which is a feature that is definitely present in the database.

```
HARMONISATION2 $\mathbf{m}_I$ $\mathbf{m}_{db}$ $\mathbf{h}_{db}$ $q$ $a$ $d_h$
let
    - - Training
    $\mathbf{m}'_{db}$ = EXTRACTFEATURES2 $q$ $a$ $\mathbf{m}_{db}$ $\mathbf{h}_{db}$
    $\mathbf{h}'_{db}$ = SIMPLIFYCHORDLINES $\mathbf{h}_{db}$
    $states$ = ZIP $\mathbf{m}'_{db}$ $\mathbf{h}'_{db}$
    $hmm$ = HMM.TRAIN $states$
    - - Harmonising
    $\mathbf{m}'_I$ = EXTRACTINPUTFEATURES2 $q$ $\mathbf{m}_I$ $d_h$
    $\mathbf{m}''_I$ = NEARESTNEIGHBOUR $\mathbf{m}'_I$ $\mathbf{m}'_{db}$
    $\mathbf{h}'_I$ = HMM.BESTSEQUENCE $hmm$ $\mathbf{m}''_I$
    $\mathbf{h}_I$ = REBUILDCHORDNOTES $\mathbf{h}'_I$ $d_h$
in
    $(\mathbf{m}_I, \mathbf{h}_I)$
```

Figure 23: Harmonisation - final version

8.3 Implementation details

The overall structure of the implementation of this final harmonisation model is almost identical to HARMONISATION1, except EXTRACTFEATURES1 and EXTRACTINPUTFEATURES1 have been replaced by EXTRACTFEATURES2 and EXTRACTINPUTFEATURES2 and the nearest neighbour call has been added. The pseudo code for HARMONISATION2 is seen in Figure 23. For performance purposes we extract the subset melody from the entire melody and then quantise this subset instead of quantising the entire melody and then extract. EXTRACTMELODY works as an opposite version of RESTMELODY; instead of cutting the beginning of the list, it returns it. We thus have the relation

$$\mathbf{m} = \text{EXTRACTMELODY } k \ \mathbf{m} \cup \text{RESTMELODY } k \ \mathbf{m} \tag{32}$$

for all k.

Having returned the list $\mathbf{m}'_{db}$ with a duration corresponding to the analysis duration a, we can quantise it using the EXTRACTBEATNOTES function. The pseudo-code for this function is shown in Figure 25 and simply extracts the notes above each beat, determined by q. Note that it also replaces the duration of the note with q. Thus, the resulting list will have a duration that is a integer multiple of q. The result of the call to EXTRACTBEATNOTES is then placed in a list and EXTRACTFEATURES2 is called recursively on the rest of the melody and chords just as in EXTRACTFEATURES1.

EXTRACTINPUTFEATURES2 work much the same way except we have a fixed duration d_h for each chord. But in contrast to the first harmonisation we run the results of it through a nearest neighbour algorithm first, as described above.

```
ExtractFeatures2 $q$ $a$ $\mathbf{m}_{db}$ $\mathbf{h}_{db}$
1 let
2     $\mathbf{m}'_{db} = $ ExtractMelody $a$ $\mathbf{m}_{db}$
3     $\mathbf{m}''_{db} = $ ExtractBeatNotes $q$ $\mathbf{m}'_{db}$
4     $\mathbf{c}_1 = $ head $\mathbf{h}_{db}$
5     $\mathbf{m}_{db,r} = $ RestMelody $D(\mathbf{c}_1)$ $\mathbf{m}_{db}$
6     $\mathbf{h}_{db,r} = \mathbf{h}_{db} \setminus \mathbf{c}_1$
7 in
8     $\mathbf{m}''_{db} \cup$ ExtractFeatures2 $q$ $a$ $\mathbf{m}_{db,r}$ $\mathbf{h}_{db,r}$
```

Figure 24: Database feature extraction – final version

```
ExtractBeatNotes $q$ $\mathbf{m}_{db}$
1 let
2     $(p_1, d_1) = $ head $\mathbf{m}_{db}$
3     $\mathbf{m}_{db,r} = $ RestMelody $q$ $\mathbf{m}_{db}$
4 in
5     $(p_1, q) \cup$ ExtractBeatNotes $q$ $\mathbf{m}_{db,r}$
```

Figure 25: Extract notes at intervals of duration q

9 Melodic variation

This section will describe in detail, how the variation of a melody has been implemented. The method presented in this section is greatly influenced by Melonet [14]: Giving a melody line, the system invents a baroque style chorale harmonisation and variation of any chorale voice. In our implementation, only the methods for varying the melody is used. Though, there are some important differences between the musical structure of a chorale and the songs in the styles of both melodic rock and jazz. The biggest difference is, that the chorales tested with Melonet consists almost entirely of quarter notes and very few rests, whereas the structure of the songs in our database can vary strongly between whole notes up to 32nd notes and rests. These differences necessitate some modifications of the Melonet, and will be highlighted in the subsection treating the implementation.

The input to Cremo[11] consists of a melody line together with chords in separate channels as depicted in Figure 11 in Section 4, and from this information, the melody line is varied in a given style contained in the database. The learning goal of the melodic variation is twofold: We would like the melodic lines to conform to some musical rules. Rules are hard to define in music but we can formulate them more loosely, as the "correct resolving" of dissonances or the "appropriate use" of successive interval leaps[12]. On the other hand, the system should also be able to capture the stylistic features of the style in the melodic rock and jazz genre from the database. This is attempted by using a multiscale neural network.

The implemented method uses a multiscale neural network to learn and reproduce higher-level structure in melodic sequences. It employs two mutually interacting networks, the supernet and the subnet, working at different time scales. The main idea of the approach is a combination of unsupervised and supervised learning, where unsupervised learning is used to classify and recognise musical structure, while supervised learning is used for prediction in time. Agglomerative hierarchical clustering is used for classifying the motifs, where neural networks are used for prediction in time. Before going into the specific implementation details, the two methods, hierarchical clustering and neural networks, are covered in some detail in the following subsections.

9.1 Neural networks

A neural network is a mathematical model that mimics the function of biological neural networks and consists of interconnected computational nodes called *neurons*, and it is the way that these nodes are connected that defines the function of the network. One of the simplest and most used types of neural networks is the feed-forward neural network, which is described in the next section.

Feed-forward network function

As described in [3], the neurons in a feed-forward neural network are divided into layers: An input layer, a number of so-called hidden layers and an output layer. The connections are then

[11]When refering to Cremo in this section, we mean the melodic variation part of Cremo, unless other is mentioned.

[12]As Miles Davis once said: "If you hit a wrong note, it's the next note that you play that determines if it's good or bad."

set in such a way, that information is fed forward from the input layer through the hidden layers and finally through the output layer.

A feed-forward neural networks can be thought of as a series functional transformations. Consider a network with one hidden layer, as seen in Figure 26. Given an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ of D variables and a dummy variable $x_0 = 1$, the reason for which will be revealed shortly, we let the input layer consist of these variables. A linear combination of these variables can be constructed as

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{33}$$

which we call the *activations* and where $w_{ji}^{(1)}$ are called *weights* of the first layer. By having the dummy variable x_0 we can use $w_{ji}^{(1)}$ as a bias. The units of the hidden layer are then defined as $z_j = h_1(a_j)$ where h_1 is a differentiable, non-linear function, called the *activation function.* The dimension of the layers need not be identical, so let $j = 1, \ldots, M$. Then the units $(y_1, y_2, \ldots, y_K)$ of the output layer is formed by output unit activations

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{34}$$

evaluated by the output activation function f, which is dependant on the type of machine learning problem we wish to solve. Thus, we can define the output of the entire neural network as

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_{kj}^{(2)} h_1\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{35}$$

Generally, if we have n hidden layers, we can define the output of the network as

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{i_n=0}^{M_n} w_{kj}^{(n+1)} h_n\left(\sum_{i_{n-1}=0}^{M_{n-1}} w_{ji}^{(n)} h_{n-1}\left(\cdots \sum_{i_1=0}^{M_1} w_{kj}^{(2)} h_1\left(\sum_{i_0=0}^{M_0} w_{ji}^{(1)} x_i\right)\cdots\right)\right)\right)$$

Parameter optimisation

The task is now to find the optimal weight vector $\mathbf{w}$, which minimise the chosen error function $E(\mathbf{w})$. Having a data set, $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{t}_n)\}$, $n = 1, \ldots, N$ of N corresponding values of $\mathbf{x}$ and $y_k(\mathbf{x})$, we wish to learn the optimal parameters from the training data. This is done by minimising an error function that measures the misfit between the function $y_k(\mathbf{x}, \mathbf{w})$ and the training data points. A widely used error function is given by the sum-of-squares error between the predictions $y_k(\mathbf{x}_n, \mathbf{w})$ for each data point $\mathbf{x}_n$ and the corresponding target values $\mathbf{t}_n$, so that we minimise

$$E(\mathbf{w}) \quad = \quad \frac{1}{2} \sum_{k=1}^{K} \sum_{n=1}^{N} \{y_k(\mathbf{x}_n, \mathbf{w}) - t_{n,k}\}^2 \tag{36}$$

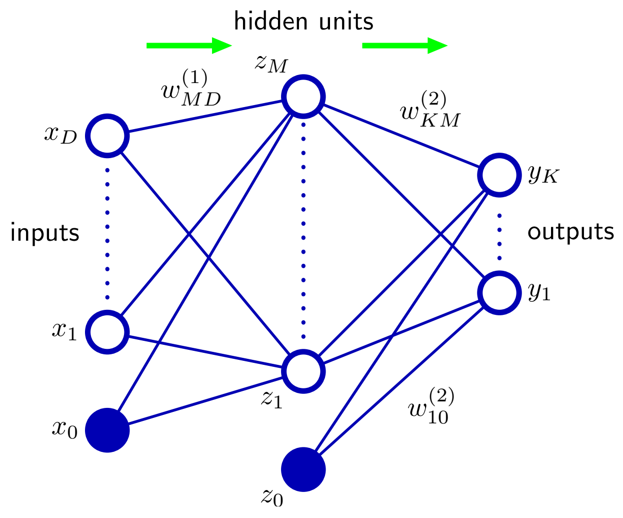


Figure 26: Neural network with one hidden layer.

The task of finding the weight vector $\mathbf{w}$ that minimises the chosen error function $E(\mathbf{w})$ can be done by a various number of heuristics following an iterative scheme taking steps in the weight space

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \Delta\mathbf{w} \tag{37}$$

A simple method for computing $\Delta\mathbf{w}$ is the *gradient descent* method, which moves the weights in the opposite direction of the error function gradient (36)

$$\Delta\mathbf{w} = -\eta\frac{\partial E(\mathbf{w})}{\partial\mathbf{w}} \tag{38}$$

where η is the *step size* parameter that determines how long a step $\Delta\mathbf{w}$ should be. This can be seen in Figure 27.

To compute the gradient of the error function efficiently, local quadratic approximation and error backpropagation is used. For details, see [3].

Classification using neural networks

As we explained in Section 2.1, we can use Bayes' theorem for the classification problem, that is, given an input vector $\mathbf{x}$, we want to compute the probability of $\mathbf{x}$ belonging to a specific class $\mathcal{C}_k$

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}$$

Suppose that our goal is simply to make as few misclassifications as possible. We would like to have a signal detection system (pattern classifier) to provide a rule for assigning a measurement $\mathbf{x}$ to a given signal category, i.e given class. Hence, a classifier divides the measurement space, i.e. feature space, into disjoint regions $R_1, R_2, \ldots, R_c$, such that measurements that fall into region R_k are assigned with class $\mathcal{C}_k$. The boundaries between decision region are called

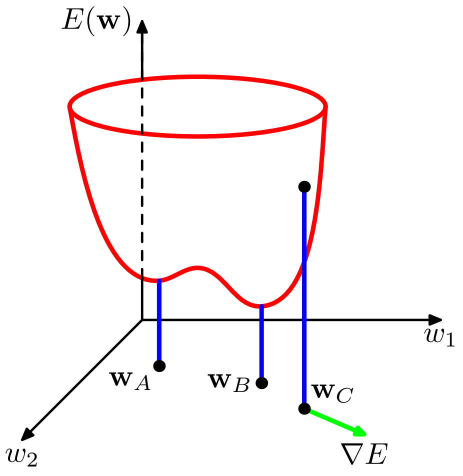


Figure 27: Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. The point $\mathbf{w}_A$ and $\mathbf{w}_B$ are minima, where $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is a global minimum. The negative of the gradient ΔE in the error surface, corresponding to the weight $\mathbf{w}_C$, is giving the direction where the function is decaying the most.

decision boundaries. For a two-class problem depicted in Figure 28, a mistake occurs when an input x belonging to class $\mathcal{C}_1$ is assigned to class $\mathcal{C}_2$ or vice versa, and can be written as

$$\begin{aligned} P(error) &= P(x \in \mathcal{R}_2, \mathcal{C}_1) + P(x \in \mathcal{R}_1, \mathcal{C}_2) & (39)\\ &= P(x \in \mathcal{R}_2|\mathcal{C}_1)P(\mathcal{C}_1) + P(x \in \mathcal{R}_1|\mathcal{C}_2)P(\mathcal{C}_2) & (40)\\ &= \left(\int_{\mathcal{R}_2} P(x|\mathcal{C}_1)dx\right) P(\mathcal{C}_1) + \left(\int_{\mathcal{R}_1} P(x|\mathcal{C}_2)dx\right) P(\mathcal{C}_2) & (41) \end{aligned}$$

We can choose the decision boundary as we want, i.e. we can assign each of the points x to one of the two classes. We should minimise $P(error)$, and therefore we should clearly assign each x to the class that have the smaller of the integrand in (41). Thus, the probability of error is minimised if we assign points to $\mathcal{R}_1$, whenever

$$P(x|\mathcal{C}_1)P(\mathcal{C}_1) > P(x|\mathcal{C}_2)P(\mathcal{C}_2) \Rightarrow P(\mathcal{C}_1|x) > P(\mathcal{C}_2|x)$$

where we have used $P(x|\mathcal{C}_1)P(\mathcal{C}_1) = P(\mathcal{C}_1|x)P(x)$.

Using a neural network for a multiclass classification problem in which each input is assigned to one of K mutually exclusive classes, we will train the network to approximate the posterior probabilities $P(\mathcal{C}_k|\mathbf{x})$ by interpreting each of the outputs y_t as the probability for classifying the input in class t. More formally, the output can be interpreted as $y_k(\mathbf{x}, \mathbf{w}) = P(t_k = 1|\mathbf{x})$, where $t_k \in \{0, 1\}$ is a binary variable indicating the class by using the 1-of-K coding scheme. Since the output of a neural network with no activation function on the output layer can give arbitrarily large values, the softmax function is used for mapping an in theory infinite interval into a finite one. The softmax function is given by

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp \phi_k(\mathbf{x}, \mathbf{w})}{\sum_{j=1}^{K} \exp(\phi_j(\mathbf{x}, \mathbf{w}))} \qquad (42)$$

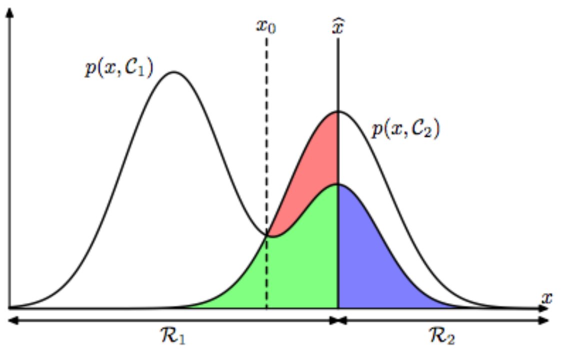


Figure 28: Schematic illustration of the joint probabilities $P(x, \mathcal{C}_k)$ for each of the two classes plotted against $\mathbf{x}$. Due to the decision boundary $x = \hat{x}$, the values $x \geq \hat{x}$ are classified as class $\mathcal{C}_2$, where values $x < \hat{x}$ are classified as class $\mathcal{C}_1$, belonging to region $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. The errors arise from the red, green and blue regions. If points $x < \hat{x}$ are misclassified as belonging to class $\mathcal{C}_1$, but belongs to class $\mathcal{C}_2$, the error is represented by the sum of the red and green area. Conversely, if points $x \geq \hat{x}$ are misclassified as belonging to class $\mathcal{C}_2$, but belongs to class $\mathcal{C}_1$, the error is represented by the blue region. As we vary the location $\hat{x}$, the combined area of the green and blue region remains the same, but the red area varies. We would like the red area to be as small as possible, since this will give the minimum misclassification. This is the case when $x_0 = \hat{x}$, since the red area disappears. This is equivalent to the misclassification rule.

which satisfies $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$. Hence, what we get from the softmax is the probability for each of the K classes, that is $y_k(\mathbf{x}, \mathbf{w})$ is the probability of $\mathbf{x}$ belonging to class $\mathcal{C}_k$.

9.2 Agglomerative hierarchical clustering

In pattern recognition problems where data consists of a set of input vectors $\mathbf{x}$ without any corresponding target values, these problems are called unsupervised learning. If the goal is to discover groups of similar examples within the data, it is called clustering, and is the assignment of a set of observations into subsets (the clusters) so that observations in the same cluster are similar in some sense.

Agglomerative hierarchical clustering[13][14]. is a bottom-up approach, that creates a hierarchy (a tree) of clusters, that may be represented in a dendrogram. The root of the tree consists of a single cluster, and the leaves corresponds to individual observations. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be N patterns to be clustered. The agglomerative hierarchical clustering starts at level N with N clusters $S_1, S_2, \ldots, S_N$ each having exactly one pattern. At level $N-1$, the two patterns with the smallest distance is merged into one new cluster, and are combined with the other clusters, giving $S_1^{(2)}, S_2^{(2)}, \ldots, S_{N-1}^{(2)}$. This process is repeated until only one cluster, $S^{(N)}$ is left. The overall history can be illustrated by a dendrogram, and can be seen in Figure 29 where eight motifs are compared. We see, that we can get the desired number of clusters by cutting the tree at a given level.

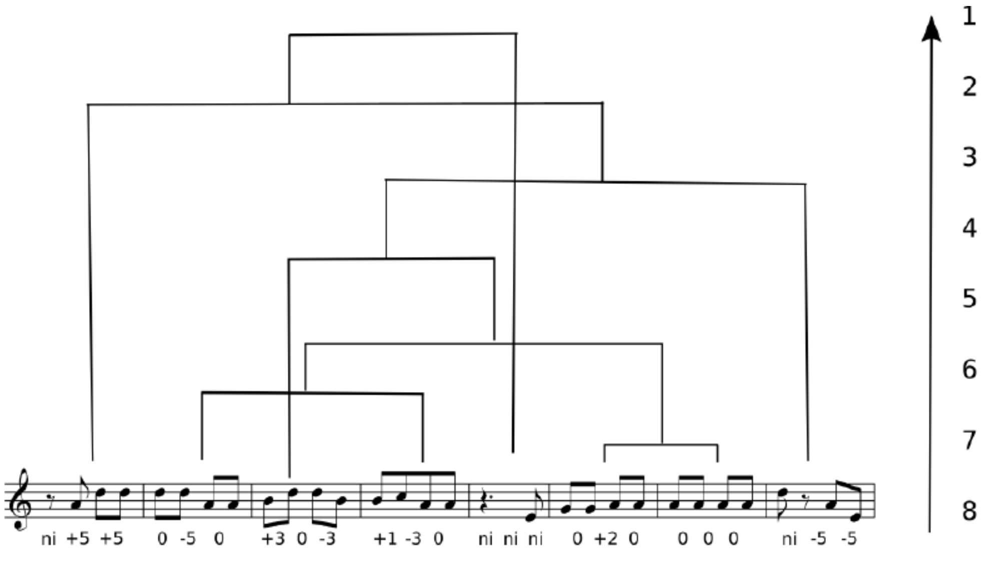


Figure 29: Agglomerative clustering algorithm represented as a dendrogram. In the bottom of the tree, all 8 motifs are assigned exactly one cluster each. At the next level, the two motifs with the smallest distance are merged. The same procedure is repeated until all motifs are assigned to only one cluster. The distance function is Euclidean, with the distance between a "no interval" (ni) and a interval is defined to be 12.

[13]http://gauss.nmsu.edu/~lludeman/video/ch6pr.html (Sun Jan 10 14:59:29 CET 2010)
[14]http://en.wikipedia.org/wiki/Cluster_analysis (Sun Jan 10 14:59:29 CET 2010)

9.3 Learning melodic variation

Having described the theory above, we now go into details with the implemented method for melodic variation.

Structure of the neural networks

We can decompose the variational task of Cremo into five subtasks [14]:

1. A melody variation is considered at a higher time scale consisting of a number of notes, called a *motif*. A motif can consist of a varying number of notes and note values. In the tests we fix the number of notes to 4 and vary the value of the notes of either eighth notes or quarter notes, giving a motif length of 2/4 and 4/4, respectively.

2. Before training the network, the motifs are classified using hierarchical clustering. Motifs that are classified in the same cluster can be seen as having the same characteristics, such as melodic contour and rhythm.

3. One neural network is used to learn the abstract sequences of motifs. Each motif is represented by a 1-of-K coding scheme, and the task for the network is to classify a melody note to either one of the motif classes that fits the note the best, considering the melodic context and the motifs that have occured previously. No concrete notes are fixed, the network is only classifying the given note as 1-of-K motif classes. This neural network is called the *supernet*.

4. Another neural network learns the motif classes classified by the supernet, as concrete notes, depending on the given harmonic context. It produces the actual melodic variations, namely the motifs. These motifs can have the properties explained in 1. Because they work at a time scale below the supernet, this neural network is called the *subnet*.

5. The motifs produced by the subnet are mainly influenced by the motif classification computed by the supernet, but since it has to find a meaningful realisation given the harmonic context, it is possible for the subnet to produce a motif that is not belonging to the motif class determined by the supernet. Therefore this motif will be reclassified by the *motif classification* component before the supernet determines the next motif class.

The motivation of this separation into supernet and subnet arose from the considerations, that it would be easier to predict the motifs given the *contour* of the new motif; that is, the overall shape of the motif given by the interval direction for each motif. An examples can be seen in Figure 30, where the red line depicts the contour of the given melody line.

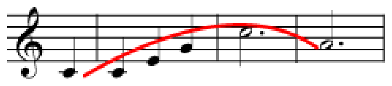


Figure 30: Contour of a melody line depicted by the red line.

The overall system can be seen in Figure 31.

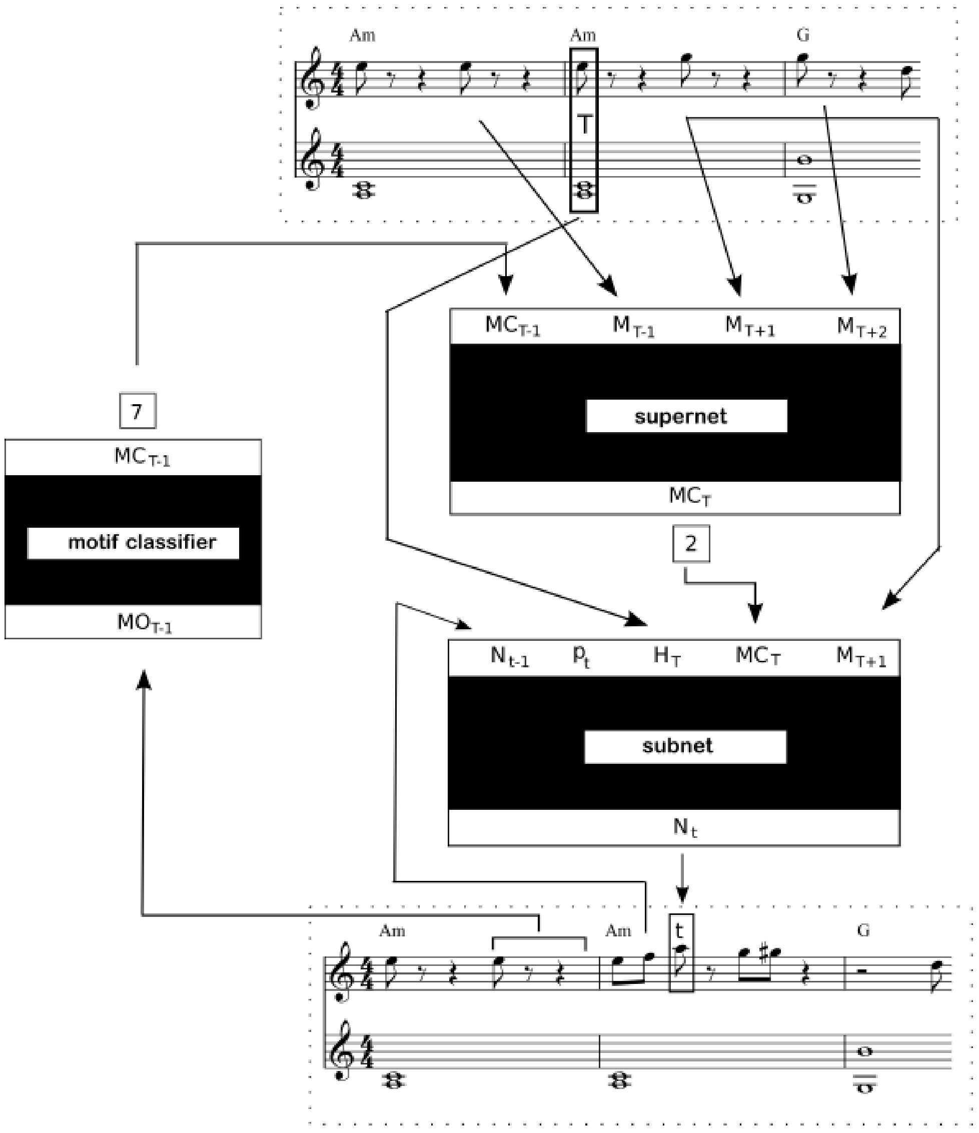


Figure 31: The structure of Cremo and the process of varying an input melody with chords. The harmonised melody is feed into the supernet, which predicts the current motif class MC_T from a local window consisting of the previous motif class MC_{T-1} and the following three notes M_{T-1}, M_{T+1} and M_{T+2}. A similar process is perform in the subnet, where the next motif note N_t is predicted on the basis of the previous note N_{t-1}, the harmony H_T for the given melody note M_T, the motif class MC_T and the preceding note M_{T+1}. The position p_t is also used, which gives the position inside the motif. When all notes for a motif is computed by the subnet, the motif MO_{t-1} is classified by the motif classifier as MC_{T-1} The classification is then feed into the supernet, and the process is repeated.

$$\begin{bmatrix} M_{T-1} & & M_{T+1} & M_{T+2} \\ MC_{T-1} & \mathbf{MC_T} & & \end{bmatrix} \tag{43}$$

$$\begin{bmatrix} 0 & & 3 & 0 \\ 7 & \mathbf{2} & & \end{bmatrix} \tag{44}$$

$$\begin{bmatrix} 100000000000 & & 000100000000 & 100000000000 \\ 000000100000 & \mathbf{010000000000} & & \end{bmatrix} \tag{45}$$

Figure 32: Supernet feature and values/codings for the music example in Figure 31. In (43), column 1,2,3 and 4 corresponds to time $T-1$, T, $T+1$ and T, respectively. The output feature for time T of the supernet is the motif class MC to be classified, and is depicted in boldface. The input features are the melodic context given by the preceding note M_{T-1} and the following two notes, M_{T+1} and M_{T+2} together with the preceding motif class MC_{T-1}. In (44), the actual values are depicted and (45) shows the bit encoding for these values.

$$\begin{bmatrix} & MC_T & \\ & H_T & \\ & & M_{T+1} \\ N_{t-1} & \mathbf{N_t} & \\ & p_t & \end{bmatrix} \tag{46}$$

$$\begin{bmatrix} & 2 & \\ & \text{minor} & \\ & & 3 \\ 1 & \mathbf{4} & \\ & 2 & \end{bmatrix} \tag{47}$$

$$\begin{bmatrix} & 020000000000 & \\ & 001000000000 & \\ & & 000100000000 \\ 010000000000 & \mathbf{000010000000} & \\ & 010 & \end{bmatrix} \tag{48}$$

Figure 33: Subnet feature and values/codings for the music example in Figure 31. In (46), column 1,2 and 3 corresponds to time $T-1$, T and $T+1$, respectively. The output feature for time T of the supernet is the note N to classify, and is depicted in boldface. The input features are the previous motif note N_{t-1}, the next melody note M_{T+1}, the harmony H_T of the present melody note and the position within the motif p_t.

As explained in the caption of the figure, the supernet classifies the motif class from a local window, where this class together with other relevant information is feed into the subnet. The subnet then predicts the next motif note. The motif classifier then classifies the new motif, which is feed into the supernet, and the process is repeated. The position for the actual note N_T in the subnet is captured by three position bits denoted by p_t. The reason for this is, that we hope to capture evolutional aspects for the notes inside the motifs. How we represent the notes are explained in the following. Because of the way we represent notes, N_t is an interval relative to the previous note, where the first note in each motif is given by the original melody note, which is explained in the next subsection.

Interval representation

Inspired by [14], we have chosen an interval representation, where each note is given relative to the previous note. These intervals are all relative to the first motif note, corresponding to the original melody note, called the *reference note*. The coding for the interval between notes is shown in Table 19, where each bit is given to individual input units in the neural networks, each interval corresponding to M and N. The *direction* of the interval is given by the first three bits, *octave invariance* is given by the next two bits, the actual interval size is given by the next 12 bits, and the last two bits are the pause bits, indicating whether the interval is leading to a note or a rest[15]. Complementary interval such as ascending fourth and descending fifths have similar encodings in the interval size, because they lead to the same note and can be regarded as harmonic equivalent. For example, in C major, an ascending fourth is leading to the note F, where an descending fifth is leading to the same note F, though in different octaves.

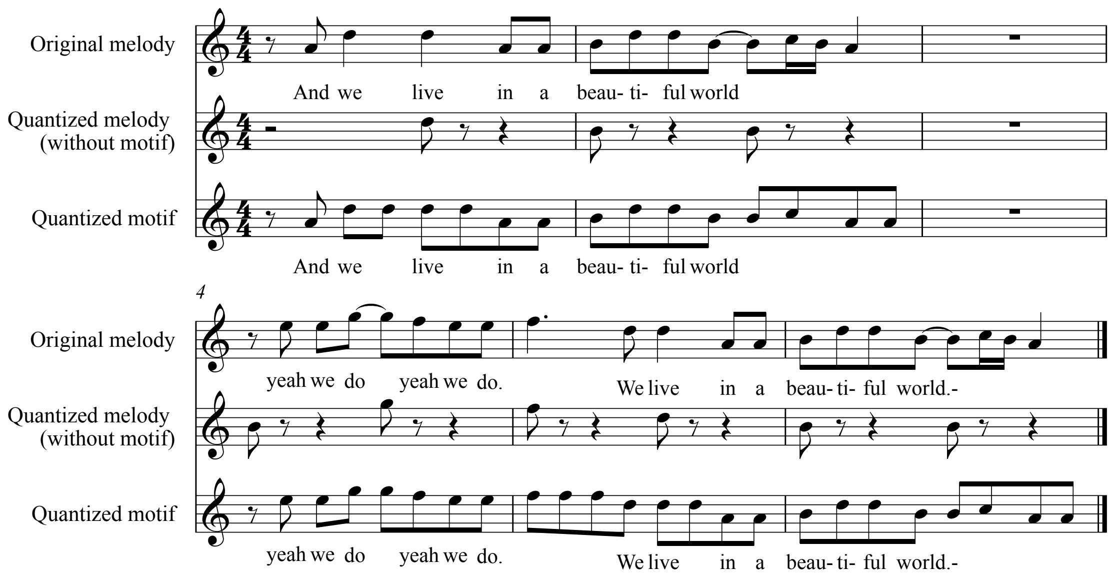
		Direction	Octave	Interval Size	Pause bit
Minor Ninth	↑	0 0 1	0 1	0 1 0 0 0 0 0 0 0 0	1 0
Octave	↑	0 0 1	1 0	1 0 0 0 0 0 0 0 0 0	1 0
Major Seventh	↑	0 0 1	1 0	0 0 0 0 0 0 0 0 0 1	1 0
⋮	⋮	⋮	⋮	⋮	⋮
Major Second	↑	0 0 1	1 0	0 0 1 0 0 0 0 0 0 0	1 0
Minor Second	↑	0 0 1	1 0	0 1 0 0 0 0 0 0 0 0	1 0
Prime	→	0 1 0	1 0	1 0 0 0 0 0 0 0 0 0	0 1 / 1 0
Minor Second	↓	1 0 0	1 0	0 0 0 0 0 0 0 0 0 1	1 0
Major Second	↓	1 0 0	1 0	0 0 0 0 0 0 0 0 1 0	1 0
⋮	⋮	⋮	⋮	⋮	⋮
Major Seventh	↓	1 0 0	1 0	0 1 0 0 0 0 0 0 0 0	1 0
Octave	↓	1 0 0	1 0	1 0 0 0 0 0 0 0 0 0	1 0
Minor Ninth	↓	1 0 0	0 1	0 0 0 0 0 0 0 0 0 1	1 0

Table 19: Complementary interval coding.

We also need an appropriate way of representing the harmonisation. We only need the harmonic context of the chord, e.g. minor 7 or maj7 as explained in Section 6.2 and summarised in Table 10, 11 and 12. We already have an interval representation for chords given in Section 6.3 where all notes are relative to the root. Since the model is only relative to the reference note, i.e. the first note in each motif, we can can omit the absolute pitch and only use the interval representation. This coding help the neural network to directly establish a relationship between intervals and a given harmony [14].

For the example given in Figure 31, Figure 32 and Figure 33 show the features and values/codings for the supernet and subnet, respectively. The first line is giving the features,

[15]With an interval to a rest, we simply mean that the given note should be a rest, hence no interval is present. If the the next note after the rest is a note, the interval to that note is measured from the last non-rest.



Original melody
Quantized melody (without motif)
Quantized motif
And we live in a beau- ti- ful world
4
yeah we do yeah we do. We live in a beau- ti- ful world.-

Figure 34: Staff 1: First 6 bars of the chorus from the song "Don't Panic", composed by Coldplay. Staff 2: the quantized melody without motifs. Staff 3: the quantised motifs. Here, the motif length is 2/4 and the quantisation is performed on the eight-notes. Notice, that the distance between the notes in the quantised melody corresponds to the motif length.

the second line is giving the values and the third line is giving the binary representation explained above. For example, in Figure 32 the previous note is given by M_{T-1} and has the value 0.

Training

The neural networks used in the supernet and subnet are both standard feed-forward network, as explained in Section 9.1. We are training the networks individually on the two genres, melodic rock and standard jazz, and the encoded features are all concatenated to form one learning pattern for each of the genres. That is, all parts for all songs are concatenated. The supernet and the subnet are trained individually on a training set. We have preprocessed the songs, such that the input melody contains no motifs, which is done by quantising the melody line as depicted in Figure 34. The quantised motifs from the original melody is classified with the hierarchical clustering algorithm, where we have chosen the number of classes to be $n = 12$, as recommended in [14]. The length of the motifs here are 2/4, but other lengths can be used as well. Thus, the input features $\mathrm{M}_{T-1}, \mathrm{M}_{T+1}$ and M_{T+2} to the supernet is the quantized melody (without the motifs) and the target MC_T is the classified motif corresponding to the motif in the original melody at position M_T. For the subnet, the input feature H_T (harmony) is given from the quantized melody and MC_T is again the motif class classified by the hierarchical clustering algorithm, and the target N_t is the quantized motifs, depicted in node line 3 in Figure 34.

We are using a single layer neural network with 35 hidden units for both the supernet and the subnet. For $n = 12$ clusters, the number of input units for the supernet is $12+19+19+19 = 61$ and the number of output units is 12. For the subnet, the number of input units is

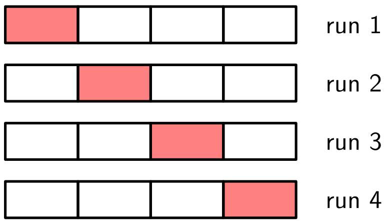


Figure 35: K-fold for $K = 4$. The red square is the test data, whereas the white squares is the training data.

$19 + 3 + 11 + 12 + 19 = 64$ and the number of output units is 19.

The training was formed by using the K-fold cross-validation [3], where we have partitioned our data set into $K = 5$ almost equal subsamples. 4 out of 5 of these samples are used as training set and the last sample as validation set. The cross-validation process is then repeated $K = 5$ times, with each of the 5 subsamples used exactly once as the validation data. The average is then taken for the sum of all the test errors. The process is depicted in Figure 35, for $K = 4$.

Cross-validation is useful when the amount of data for training and testing is limited, and we wish to train on a training set as big as possible. In [14], 35 and 25 hidden units are used for the supernet and subnet, respectively, and the learning algorithm RPROP is used for training. RPROP is a heuristic for finding a minimum for the error function, like the more simple gradient descent algorithm from Section 9.1. We have also used RPROP and based our weights one these values, though we have chosen to use 35 hidden units for the subnet. Because of time constraints we have not managed to train our networks optimally, but have loosely chosen the following parameters for the training of the subnet and supernet:

Parameter	Value
Max epochs (supernet)	150
Desired error (supernet)	0.5
Hidden units (supernet)	35
Max epochs (subnet)	200
Desired error (subnet)	0.7
Hidden units (subnet)	35
Ignore intervals when pause	Yes
Ignore octaves	Yes

The average training and validation errors for the two genres can be seen in Figure 20 and 21.

9.4 Implementation details

The first task for realising the method presented above, is to be able to extract and quantise the motifs and to extract and quantise the melody line without the motifs, as depicted in Figure 34. Just as in the harmonisation, we have to extract motifs relative to the chords. The process is performed by the function ExtractMotifs as seen in Figure 37. Note the close similarity with the function ExtractFeatures2 from Section 6.5. The only difference is

	Supernet **test / val**	**Subnet** **test / val**
First fold	0.58 / 1.04	1.87 / 2.23
Second fold	0.61 / 0.98	1.89 / 2.22
Third fold	0.61 / 1.00	1.91 / 2.09
Fourth fold	0.61 / 0.86	1.94 / 2.05
Fifth fold	0.61 / 0.92	1.90 / 2.19
Average error	0.60 / 0.96	1.90 / 2.16

Table 20: Training and validation error for melodic rock for both the supernet and the subnet.

	Supernet **test / val**	**Subnet** **test / val**
First fold	0.50 / 1.05	1.90 / 2.35
Second fold	0.50 / 1.12	1.87 / 2.36
Third fold	0.50 / 1.19	1.90 / 2.44
Fourth fold	0.50 / 1.03	1.83 / 2.27
Fifth fold	0.50 / 1.25	1.87 / 2.39
Average error	0.50 /1.13	1.88 / 2.37

Table 21: Training and validation error for standard jazz for both the supernet and the subnet.

that if a chord spans more than one motif duration, we keep that chord for the next recursion an extract the next motif under it. Having extracted the motifs, we represent the note values as intervals.

The way we calculate intervals to a given target note N, is to calculate the interval from the last non-rest to the given note. This can in some circumstances remove information about the order of the notes and rests, which can be a problem since we need to compare intervals in the hierarchical clustering. The problem is depicted in Figure 36. We have implemented intervals between a rest and a note or the interval between two rests as `NoInterval`, and the interval between two notes by `Interval i`, where `i` is the actual distance. Here we see, that the motifs in measure 1 and 2 are converted to the same interval representation ni, +4 and -2 corresponding to `[NoInterval,Interval 4,Interval (-2)]`, though the original motifs are different.

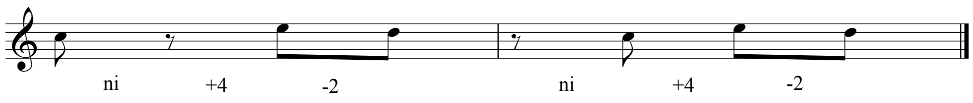


Figure 36: Two different motifs in bar 1 and 2, respectively. 'ni' is no interval and the numbers represents the interval difference. The interval representation is representing the two motifs as being the same, despite the obvious difference in the original note representation.

When calculating the distances between motifs, we use the Euclidian distance. We have to define what we mean by taking the difference between corresponding intervals for each motifs when working with motif intervals of type `NoInterval` and `Interval i`. The distance between `Interval i` and `Interval u` is computed by taking the difference of i and u, where

```
$\textsc{ExtractMotifs}\ q\ m\ \mathbf{m}_{db}\ \mathbf{h}_{db} =$
1  let
2      $\mathbf{m}'_{db} = \textsc{ExtractMelody}\ a\ \mathbf{m}_{db}$
3      $\mathbf{m}''_{db} = \textsc{ExtractBeatNotes}\ q\ \mathbf{m}'_{db}$
4      $\mathbf{c}_1 = \text{head}\ \mathbf{h}_{db}$
5  in
6      if $D(\mathbf{c}_k) \geq 2m$   - - Another motif can be within the duration of the chord
7          let
8              $\mathbf{m}_{db,r} = \textsc{RestMelody}\ m\ \mathbf{m}_{db}$
9              $\mathbf{h}_{db,r} = \mathbf{c}'_1 \cup (\mathbf{h}_{db} \setminus \mathbf{c}_1)$ , where $D(\mathbf{c}'_1) = D(\mathbf{c}_1) - m$
10         in
11             $\mathbf{m}''_{db}\ \cup\ \textsc{ExtractMotifs}\ q\ m\ \mathbf{m}_{db,r}\ (\mathbf{c}'_1 \cup \mathbf{h}_{db,r})$
12     else   - - No other motif can be within the duration of the chord
13         let
14             $\mathbf{m}_{db,r} = \textsc{RestMelody}\ D(\mathbf{c}_1)\ \mathbf{m}_{db}$
15             $\mathbf{h}_{db,r} = \mathbf{h}_{db} \setminus \mathbf{c}_1$
16         in
17             $\mathbf{m}''_{db}\ \cup\ \textsc{ExtractMotifs}\ q\ m\ \mathbf{m}_{db,r}\ \mathbf{h}_{db,r}$
```

Figure 37: Motif extraction

the distance between two `NoInterval`'s is 0. It is difficult to reason about exactly how different a `Interval` is from a `NoInterval`, but we have defined the distance to be 12, which corresponds to the difference of an octave between two intervals.

To our knowledge, no Haskell library for hierarchical clustering is available. Therefore, we turned our attention to the open source C library "C Clustering Library"[16]. This necessitates interaction between Haskell and C which we managed by simple IO interaction. Since the classification is done only initially, this method is feasible. The motifs extracted by Haskell is written to a file, imported into C, where hierarchical clustering is performed. The C program writes the individual classes assigned for each motif to a file, that is imported into Haskell. An example of the motifs from Haskell written to the file is shown is Figure 38

The motif classifier from Figure 31 is computing the distance to all motifs, and the index for the smallest distance is used for finding the corresponding cluster by looking up in an array consisting of the the assigned cluster for all motifs (computed initially by the C program, as explained above).

Discussion

In this section, we will emphasise the differences between Melonet and the implemented version in Cremo. Furthermore, we will discuss the implementation design of the model, and point out some problems we have encountered during the tests. An example of a generated variation is seen in Figure 39.

[16]http://bonsai.ims.u-tokyo.ac.jp/m̃dehoon/software/cluster/software.htm (Mon Jan 11 21:44:29 CET 2010)

```
2633 3
[Interval 0,Interval (-2),Interval 0]
[Interval (-3),Interval 0,Interval 0]
[Interval (-2),Interval 0,Interval 0]
[Interval 0,Interval 0,Interval (-1)]
[Interval 2,Interval 0,Interval 0]
[NoInterval,Interval 0,NoInterval]
.
.
.
```

Figure 38: The motif list written to a file. The first number in line 1 is giving the number of motifs and the second number is telling how many intervals each motif consists of.

Melonet is a system for baroquestyle chorale variation of a melody line in the style of J. Pachelbel. Only melodic variations consisting of four sixteenth notes are considered. This is a big difference compared to the database of Cremo, which consists of a wide range of songs within the genre of melodic rock and standard jazz. The notes in these songs can last several bars, or be sixteenth notes, and everything in between. Therefore, we have implemented a way to quantise melody and motifs, as explained in Section 9.3. The problem here can be, that there are many notes with a long durations, resulting in a lot of intervals of size 0. Since such an interval is not very interesting, when considering individual motifs of e.g. 2/4, we have implemented a way of removing such intervals from the corpus. Similarly, we can also have many bars with no melody, only consisting of rests, therefore a method for removing these are also implemented. A threshold can be set, so that 0, 1, 2 or 3 `NoIntervals` and/or 0, 1, 2 or 3 `Intervals 0` can be allowed in each motif, forcing the network to produce more interesting motifs. These issues can be seen in Figure 40.

The classification algorithm for most of the artists in the database is classifying most of the motifs to one class and the rest of the motifs are distributed among the remaining 11 classes (for $n = 12$), which is related to the problems discussed above. Because of time constraints we have not made an in-depth investigation of the problem, but we are convinced that the problem is caused by having many motifs that are relatively close to each other, whereas a few other motifs (> 11) are very different from these. The result of this scenario is, that the group of many similar motifs are assigned to the same cluster, and the ones very different from these are assigned to the remaining clusters.

We have implemented our own pause bits in the supernet by adding 2 extra bits. This sometimes cause problems, since it is possible for the network to misclassify the input melody, so that both the pause bit and an interval different from the prime occures simultaneously. We deal with that problem by selecting the pause bit, if this occurs. The implementation of the pause bit could have been implemented by adding an extra bit to the complementary interval coding depicted in Table 19, giving unambiguous results. Though, the implementation in [14] deals with the same kind of dependencies for the octave bit and the coding for the intervals, but they argue that this coding gave

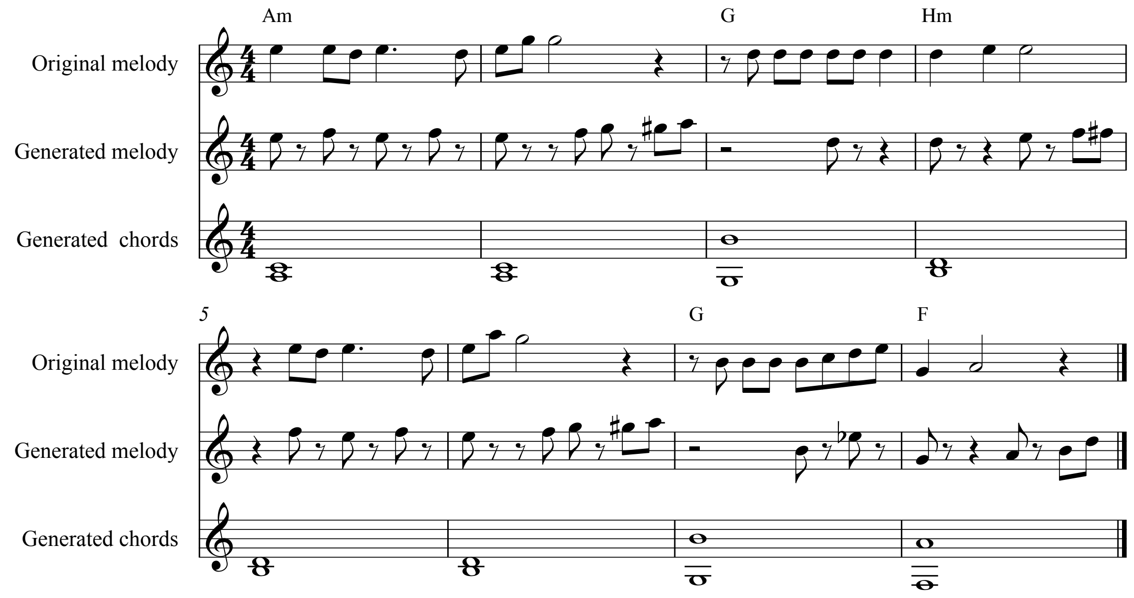


Figure 39: An example of a melody variation over Burma by Börsenfieber. The first staff shows the original input melody and the third staff shows chords that have been generated by the harmonisation part. The second staff shows the variation that has been generated from the original melody and the chords. Note that there is very little similarity between the variation and the original melody and that the variation is out of key in several places.

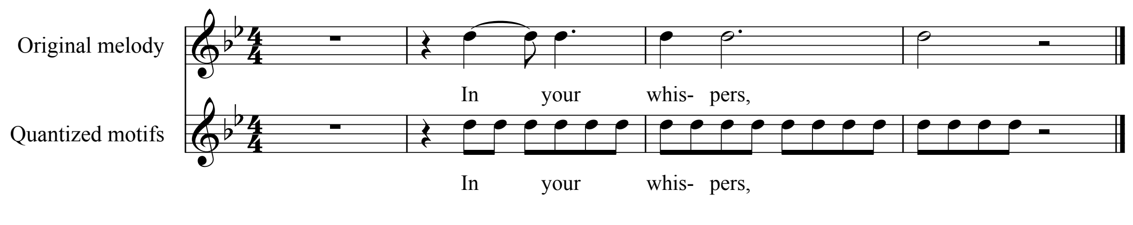


Figure 40: Staff 1: First 4 bars of the a part from the song "Thoughts Of A Dying Atheist", composed by Muse. Staff 2: the quantised motifs. Since the melody consists of the same note and the duration of the half notes in bar 3 and four have a relative long duration, we get a lot of 0 intervals. Furthermore, bar 1 consists of only rests, resulting in many no intervals.

the best results in their tests.

We observed that the octave bits often was set to 0 1, meaning that the intervals were greater that one octave. Because these intervals sounded rather unnatural, we decided to ignore these bits.

In [14], additional information is given about the position for the melody note N_T in the supernet, similar to the position p in the subnet. Due to the missing information about bars and time signature, we have no information about the position within a bar. Implementing such features in Euterpea would require more work than time admitted for this project.

The chromatic interval representation was used, in contrast to the diatonic interval representation used by [14], because the music in the database is not strictly diatonic. Tunes in both genres can have chromatic motifs and modulations. But the chromatic interval representation proved to generate to many chromatic movements and therefore "wrong" notes with respect to the underlying harmony.

The training and validation errors from Table 21 are used primarily for ensuring that data is not being overfitted, and we see that the differences between the average traning- and validation error is quite high, especially for the supernet. Because of time constraints, we have not managed to train our networks optimally, and therefore we can not give a single answer for the rather poor result. Besides overfitting of the training data, the reason could be caused by a sparse data set, resulting in a network having problems with new data not encounted before, or because the model is not capturing the melodic content very well. We have a presumption that the model is having trouble capturing the melody variations for the genres used. In contrary to baroque chorales, we believe that the musical information for melodic rock and standard jazz is more scattered, because 1) durations of chords and notes can last for several bars, 2) the interval between notes can have have little variations over a short period of time. Therefore, the information given to the supernet and subnet is probably not containing enough information to give unequivocal variations for the input melody.

10 Final user test

For the test of the final version of Cremo, we needed to test the system in its final state. We have therefore changed the structure of the test compared to the prototype test in Section 7. The questions and all the answers from this test is shown in Appendix C.

10.1 Goals of the test

We have chosen to test Cremo with the test persons from the prototype test and with some new test persons. Out of the four test persons from last test, three of them had the possibility to conduct the test: Two in the melodic rock genre and one in the standard jazz genre. They are named Respondent 1, 2 and 3 and are the same as Respondent 1, 2 and 3 from the prototype user test. We have contacted 3 new test persons, where only one had the possibility to do the testing. He will be labeled Respondent 4 and his genre is melodic rock. The goal of the user test is to test the harmonisation and melody variation is the same as for the prototype test in Section 7.1, except that we do not test whether the generated melodies are representative for the genre.

10.2 The test

Contrary to the prototype test, the final test is carried out as an email correspondence. For the actual evaluation of Cremo, we chose all the artists in the database for the respective genres, and send the generated MIDI harmonisations and variations along with a questionnaire for each MIDI file by email to the test persons. The email explained where to find MIDI files and in which order to listen to them. The supplied questionnaire was a Word-document, in which the answers could be written directly below the questions.

The evaluation protocol can be summarised as follows:

1. Tell the test person that this is the final version of Cremo.
2. Explain that four harmonisations and four variations of the supplied melody have been generated.
3. List the order in which the MIDI's should be played:
    - Harmonisation 1 through 4
    - Variation 1 through 4
4. Explain that the questionnaires are placed in the same folder as the corresponding MIDI files and that answers can be written directly in these Word documents.
5. Supply the deadline for the answers and explain that if any questions or problems can be mailed to us at any time.

10.3 Results

In this section, we summarise the answers from the evaluation questionnaire. The results should be met with the same reservation as for the results in Section 7.3. All answers from multiple choices considering harmonisation and melody variation for the both genres can be seen in Table 22 and Table 23, respectively. These tables are read as follows: Each row is a question and the columns labeled columns are the multiple choice answers. These answers are different for the various questions, so each question row is divided into two: The top row stating the question along with the answer alternatives and the bottom row stating the results for each answer alternative. The numbers are the number of harmonisations (or melodies in the case of Table 23) that have been given the particular answer to the particular question. Since we have been testing on three rock musicians and one jazz musician, giving them four harmonisations and four melody variations each, we have 16 harmonisations and 16 melody variations in total. In this summary we will consider all harmonisations and variations together, since we only have one test person in the jazz genre and one new test person.

Notice in the answers to Question 1.1 in Table 22 that there is a clear tendency towards the negative side; the chords do not fit the melody. This is also supported by the tendencies in Q1.3 and Q1.7, in which the respondents in almost clear unity state that the chords do not exhibit proper tension/release properties and therefore cannot be used as a direct harmonisation of the melody. Note, however, that the answers to Q1.6, whether there were inspirational aspects in the harmonisation, indicate that some of the harmonisations actually contained interesting and inspiring aspects. Here are some statements, that indicate the general opinion about Cremo's harmonisations:

> "There is no coherence between melody and chords."
> *Respondent 4*

> "The last 4-8 measures sound all wrong. The rest is harmonically correct, I guess. The musical product is not inspiring, though."
> *Respondent 2*

> "I am very impressed with Cremo in the first 6 measures. It almost sounds as something I could have composed myself."
> *Respondent 1*

> "A lot has happened since last time – very good! The harmonisation has become more coherent and makes more sense with respect to the melody."
> *Respondent 3*

Respondent 3 even provides some helpful thoughts:

> "There are extremely many parameters in making a good harmonisation – you have probably considered them all! – I don't know how they are put into a program but they have to be there to make the program usable."
> *Respondent 3*

Notice the answers to Q1.1 and compare these with the answers in Q2.8 in Table 15 and 17. We see that 7 of the harmonisations have been rated as having no coherence, 7 as having some coherence and 2 as having good coherence in both tests. This does not indicate an improvement.

For the melodic variation, Table 23 shows the answers to the questionnaire about the 16 variations. We see here that the quality of the variations is very different (Q2.1), although

Question	A	B	C	D	E
Q1.1: How does the harmonisation of Cremo sound together with the melody?	Chords don't fit	Fit somewhat	Fit well	Fit very well	
Answers	7	7	2	0	-
Q1.3: Does Cremo's harmonisation support the melody? Is there good tension/release in between chords?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	7	4	5	0	0
Q1.4: How would you describe the variation of the relation of the chords?	No variation	Some	Medium	Much	Very much
Answers	0	4	10	2	0
Q1.5: Is the variation of the chords appropriate for the genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	3	3	10	0	0
Q1.6: Could you use the generated chords as a source of inspiration for new chords for the melody or as inspiration for a new melody?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Answers	4	5	5	2	0
Q1.7: Could you use the generated chords directly as a basis for your melody?	Yes	No	Maybe		
Answers	0	14	2	-	-

Table 22: Multiple choice summary for harmonisation. Notice in Q1.1 how there is a clear tendency to discard the harmonisation. Most chords simply do not fit the melody. This also shows clearly in Q1.7. But note that 7 out of 16 could actually use some aspects of the harmonisation to some or greater extend (Q1.6).

there is a tendency towards the negative side. Whether the intervals are musical (Q2.4), most answers that the they do not fit the genre very well. A far more serious problem is that in 12 out of 16 variations the original melody is not recognisable any more (Q2.7). This is a problem, because we are trying to do melodic *variation*, but if the original melody is not recognisable in the variation, we might as well have created a whole new melody. Whether the variation can be used as an inspiration is not unequivocal either; 6 variations could be used to some extend and 4 to a lesser extend and 6 not at all. The general feeling can be summarised with the following comments:

> "What I am hearing is actually fine – melody and chords fits together. BUT – it has nothing to do with the original starting point!!!!!"
> *Respondent 3*

> "Funny input. I like that the melody is consistently working with ascending

lines and large intervals. I would check this universe out.”
Respondent 3

“Especially in the beginning I can follow Cremo’s choices but after that it blacks out again. It is as if it yells ”Too much Data, Too much Data!!” and tries to fit too many notes into the song.”
Respondent 1

“The melody is quite psychedelic, which doesn’t fit the genre.”
Respondent 4

Question	**A**	**B**	**C**	**D**	**E**
Q2.1: How does Cremo’s variation sound together with the chords?	Melody doesn’t fit	Fits somewhat	Fits well	Fits very well	
Answers	6	5	2	3	-
Q2.3: Does Cremo’s melody support the chords? Is there good tension/release?	Not at all	To a lesser extend	To some extend	To a great extend	Don’t know
Answers	5	5	6	0	0
Q2.4: Is the composition of intervals between notes musical with respect to the chords and genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don’t know
Answers	3	8	1	0	4
Q2.5: How would you describe the variation of the melody?	No variation	Some	Medium	Much	Very much
Answers	0	2	5	9	0
Q2.6: Is the variation of the melody appropriate for the genre?	No variation	Some	Medium	Much	Very much
Answers	4	3	5	0	4
Q2.7: Could you recognise your own melody?	Not at all	To a lesser extend	To some extend	To a great extend	Don’t know
Answers	12	3	1	0	0
Q2.8: Could you use aspects of the generated variation as a source of inspiration for the original melody or a new compositions?	Not at all	To a lesser extend	To some extend	To a great extend	Don’t know
Answers	6	4	6	0	0

Table 23: Multiple choice summary for the melodic variation. Note how there is no clear indication of whether the melodies fit the chords or not (Q2.1), although a slight majority tends to the negative side. The same pattern is seen in Q2.3. Also note that in 12 out of 16 variations the original melody was not recognisable.

10.4 Conclusion

Compared to the prototype test, the cohence between the melody and the chords is exactly the same in the final harmoniser (see Q2.8 in Figure 15, 17 and 22). This came as a surprise for us, since we believed that the final harmonisation was greatly improved.

For the question concerning whether the harmonisations could be used as a source of inspiration, for a total of 16 harmonisations, 5 could to a less extend be used, 5 could to some extend be used and finally 2 could to a great extend be used. These results show us, that the generated harmonisations actually contain inspirational aspects in most cases. Therefore, we conclude that Cremo is able to generate harmonisations, that can be used as an inspiration for the musician.

Whether the original melody line could be recognised after the variation, 12 out of 16 variations could not be be recognised at all. A reason for this could be that the generated variation was done on the basis of chords generated by the harmoniser and not on the original chords, another reason could be that the quantisation ruins the original rhythm of the melody.

For the question concerning whether the melodic variation of a melody line could be used as a source of inspiration, 6 could not be used at all, 4 to a lesser extend and, finally, 6 could be used to some extend. This means that the majority of the variations contain aspects that could inspire the musician, and this must be interpreted as a modest success.

11 Conclusion

We have created a system in Haskell for harmonising and varying a melodic input and for generating new melodic content on the basis of a statistical analysis of a large database of music in two different genres: Melodic rock and standard jazz. The first task of this project was to build the database. This was done by scanning written notes into MIDI files using a note-OCR program and a music notation program. These MIDI files was then split up into the main melody line and a chord line and placed in different MIDI channels. Each tune was additionally split up into its different parts (e.g., a, b, bridge, chorus) in order to be able to analyse the individual parts of the tunes.

Choosing to base our system on statistical machine learning techniques provided us with the possibility of being able to expand the database with any music in any genre, constrained by having a clear melody line with matching chords. This means that we do not have to alter any part of the program in order to generate material in other genres. Simply add the MIDI files to the database and adjust the parameters to take the new genre into account, and Cremo can generate content in this new genre.

A simple Markov chain model was created for developing new melodic lines from an analysis of the database content. Using a k'th order Markov chain we can model probabilities of the $(k+1)$'th note given the k preceding notes and thereby generate new melodies with similar statistics as the ones we have in the database.

A harmonisation model using Hidden Markov Model was developed for placing chords at given intervals under the melody. The model considers a sequence of chords as a Markov chain emitting a melody. This way we can set the parameters of the HMM from the contents of the database and use the Viterbi algorithm to generate new harmonies for the input melody.

Melodic variation was implemented using two neural networks looking at motifs. The first network (called the supernet) was used to classify motifs in the input melody and the second network (called the subnet) was used to generate new motifs. Both networks were trained individually on the contents of each genre in the database.

Finally, two user tests were conducted to evaluate the output of the system. The objective of the first user test was to evaluate a prototype of the system. The prototype system consisted of the Markov chain model for melodic development and harmonisation of melody lines by a first version of the HMM harmoniser. These were evaluated by four musicians; two in each genre. Based on the results of this test, we further developed the harmonisation model and created the neural network model. This expanded system was then evaluated by four musicians, three of which also participated in the previous test.

The prototype test showed that the Markov chain model for generating new melodic lines was too simple to produce coherent and inspiring melodies, and we saw no prospects for further development of this particular model. Instead, we focused on improving the harmoniser and developing the melodic variation system.

In the final test, for the questions concerning whether the harmonisations could be used as a source of inspiration, out of 16 harmonisations, only 4 could not use it at all, this means that the remaining 12 harmonisations contain inspirational aspects. From these answers, we conclude that the goal of producing harmonisations that can inspire the musician, is fulfilled.

Concerning whether the melodic variation of a melody line could be used as a source of inspirations, out of 16 variations, 6 could not be used at all, 4 could to a lesser extend be used and finally 6 could to some extend be used. Since the majority of the variations contain aspects that could inspire the musician, we will consider the variation of melodies as a modest success.

11.1 Further work

Using HMM for the harmonisation we consider chords as a first order Markov chain. This means that each chord is only dependant on the previous chord. Since cadences often consist of three or four chords [21] it would be appropriate if we could extend this model to consider more than one previous chord. Christian Walder [25] has created a model for harmonising Bach chorales similar to [1] but instead of using HMM, he uses a support vector machine, and implements a dynamic programming algorithm similar to the Viterbi algorithm that considers more than one previous chord.

As discussed in Section 9.4 the difference between the average training and validation errors from Table 21 are quite high, especially for the supernet. We believe, that considering information in such a small window is not sufficient for the given genres, therefore some modifications should be done. By using a bigger time window, another neural network considering intervals and slurs between nodes – together with the rhythmic structures of the motifs based on the motifs in the database and the original structure of the input melody – longer melody lines could probably be learned. We could then rebuild the melodic lines, so that for example three successive eighth notes could be combined to a dotted quarter note. Combined with note predictions for several bars at the time, we could hope for more coherent melody lines resembling the genre we would like to mimic.

A more thorough user test would have to be conducted to fully establish whether Cremo is actually able to be an inspiration to musicians, and what the optimal parameters for harmonisation and melodic variation needs to be. Such a test would require a user interface to be developed, so that the respondents could adjust the parameters themselves, thereby finding the parameters they would deem optimal. Since music is such a subjective endeavour we cannot expect to find parameters that would be optimal in for all musicians in all genres, but we would possibly be able to cut down the the parameter space to a selection of a few preset parameters for each genre.

A Contents of the database

A.1 Melodic rock

Coldplay

1. A Message
2. Don't Panic
3. Everything's Not Lost
4. Fix You
5. High Speed
6. Low
7. Life Is For Living
8. Parachutes
9. Shiver
10. Sparks
11. Speed Of Sound
12. Spies
13. Square One
14. Swallowed In The Sea
15. Talk
16. The Hardest Part
17. Til Kingdom Come
18. Trouble
19. Yellow
20. We Never Change
21. What If
22. White Shadows
23. X & Y

Jeff Buckley

1. Dream Brother
2. Eternal Life
3. Everybody Here Wants You
4. Grace
5. Hallelujah
6. Last Goodbye
7. Lilac Wine
8. Lover, You Should've Come Over
9. Mojo Pin
10. Nightmares By Sea
11. So Real
12. The Sky Is A Landfill

Mercury Rev

1. A Drop In Time
2. Chains
3. Hercules
4. Lincoln's Eyes
5. Little Rhymes
6. Nite and Fog
7. Spiders and Flies
8. The Dark Is Rising
9. Tides Of The Moon
10. You're My Queen

Muse

1. Butterflies and Hurricanes
2. Bliss
3. Dead Star
4. Hyper Music
5. Hysteria
6. In Your World
7. Micro Cuts
8. Muscle Museum
9. New Born
10. Plug In Baby
11. Sing For Absolution
12. Soldier's Poem
13. Space Dementia
14. Starlight
15. Sun Burn
16. Thoughts Of A Dying Atheist
17. Time Is Running Out
18. Unintended

Radiohead

1. 2+2=5
2. All I Need
3. Anyone Can Play Guitar
4. A Punchup At A Weeding
5. A Wolf At The Door
6. Creep
7. Everything In Its Right Place
8. Fake Plastic Tree
9. Faust Arp
10. High & Dry
11. House Of Cards
12. Idioteque
13. I Will
14. Knives Out
15. Life In A Glass House
16. Morning Bell
17. Motion Picture Soundtrack
18. Nude
19. Optimistic
20. Reckoner
21. Scatterbrain
22. We Suck Young Blood
23. You And Whose Army

A.2 Standard jazz

Bill Evans

1. 34 Skidoo
2. Detour Ahead
3. Emily
4. Funkallero
5. Gloria's Step
6. Peri's Scope
7. Re: Person I Knew
8. Since We Met
9. Spring Is Here
10. Story Line
11. Time Remembered
12. Turn Out The Stars
13. Up With The Lark
14. Very Early

Charlie Parker

1. Anthropology
2. Blues For Alice
3. Shaw 'Nuff
4. Wee (Allen's Alley)

John Coltrane

1. 26-2
2. Autumn Serenade
3. Bessie's Blues
4. Body And Soul
5. Central Park West

Miles Davis

1. Bye Bye Blackbird
2. Dig
3. E.S.P
4. If I Were A Bell
5. Lady Bird
6. Nefertiti
7. Solar
8. Tune Up

Thelonious Monk

1. In Walked Bud
2. Monk's Mood
3. Off Minor
4. Well, You Needn't

Wayne Shorter

1. Ana Maria
2. Beauty And The Beast
3. Black Nile
4. El Gaucho
5. Night Dreamer
6. Speak No Evil
7. Toy Tune
8. Wildflower

B Prototype user test

B.1 Protocol

The goal of our first user test is to verify that our prototype system works according to our synopsis description. Since we cannot use automated tests to verify this, we need the ears of a working musician. First we will let 4 musicians test Cremo and answer two questionnaires. The questionnaires are formed so that we can extract information about three different aspects: First of all, we need to know how much into music theory they are and how well they know the artists in our database, so that we can see how educated their answers will be. This will be the contents of the first questionnaire (background questionnaire). Secondly, we need to know if Cremo complies with the basic rules of music (e.g. dissonance and tension/release). Thirdly, we want to figure out if composers can actually use Cremo for their composing duties. This will be the contents of the second questionnaire (evaluation questionnaire). This first iteration will focus on harmonization of an input melody as well as the generation of a completely new melody composed by Cremo from the contents of the database.

Preprocessing

We will send the background questions by email to the musician. These questions will answer what kind of musical education he has and how well he is with the music theory relevant for Cremo. Also, it will answer how well the music in our database is known and if the musician is a composer. On the basis on these questions, we will judge who is capable of evaluating Cremo by the following criteria:

1. Two out of five of the theoretical questions should be answered correctly (from the section "Teoretisk viden"). Though, for the test persons in the standard jazz genre, one of the questions 3-5 should be among the correct answers, since these questions are important for evaluating this genre.
2. Should at least have composed 10 songs
3. For test persons who should test the rock genre, he or she should be able to describe at least 3 of the artists in this genre. For the standard jazz genre, we will require that the test person is able to describe at least 4 of the artists in this genre, since the corpus of the individual artists in this genre is smaller.

If the test person complies with above criteria, we will use his or her melody for the test. The melody can be either a fully composed piece or a subset consisting of minimum 16 bars of melody. Preferably, the format of this piece is either MIDI or a Sibelius file but a PDF notation or a paper form will also work. We will then preprocess the piece and make sure it complies with the format for Cremo: Convert to proper MIDI format and place melody in Channel 1.

Thus, we have the following protocol for the preprocessing of our user test:

1. Send questionnaires to musicians via email and a request for minimum 16 bars of melody.
2. The musicians return the questionnaire by email and attach a melody line (minimum 16 bars) in proper format (MIDI, Sibelius or PDF) or hand us the answers and melody line in paper form.
3. We go through the answers and pick those musicians that match the criteria as defined above.

Given the parameters estimated from his answers to the preprocessing questionnaire, we generate harmonizations in his preferable genre in the style of an artist he knows. We will choose parameters for Cremo for individual test persons, depending on the knowledge of artists in our database, so that we can make sure that the output should be reminiscent of an artist that is known to the test person. We will create four different harmonisations of the input melody and generate one new melody:

1. In the melodic rock genre, we pick the three best-known artists by the test person, and create three harmonisations on the basis of all permutations of these artists, where two artists are use

for every harmonisation, and a harmonisation with all artists from the melodic rock genre. For the standard jazz genre, we pick the four best-known artists by the test person, and create four harmonisations on the basis of all permutations of these artists, where three artists are use for every harmonisation. For example, if the artists Radiohead, Coldplay and Mercury Rev are the best-known artists by the test person for the melodic rock genre, we then have the following permutations:

- Radiohead - Coldplay
- Radiohead - Mercury Rev
- Coldplay - Mercury Rev

We will then generate three harmonisations with these artists and a fourth harmonisation with all the artists in the genre.

If the four best-known artists for the standard jazz genre are for example Miles Davis, Bill Evans, Thelonious Monk and Wayne Shorter, we get the following permutations:

- Miles Davis - Bill Evans - Thelonious Monk
- Miles Davis - Bill Evans - Wayne Shorter
- Miles Davis - Bill Evans - Wayne Shorter
- Wayne Shorter - Bill Evans - Thelonious Monk

We will then generate four harmonisations with these parameters.

2. A melody is created using the two (melodic rock) or three (standard jazz) best known artists.

To summarise: For both genres we will test 4 harmonisations of the user melody and 1 melody line from selected artists.

Instructions for the test

We meet the test persons either at their home or at DIKU[17], where we start the interview by explaining the goal of the project as defined in the synopsis; that the objective of the project is to create a program that supports the following:

1. Take an input melody composed by the user and alternate this melody on the basis of a given genre from the database.
2. The program should generate harmonisations on the basis of either the alternated melody and/or the original melody.

Since this is the first prototype, we explain what we will test:

1. Cremo will generate harmonisation on the basis of the input melody that the test person has given us, and we will tests the quality of these harmonisations, based on a user test.
2. The melody generator will generate a new melody on the basis of a few artists from the database, and not on the basis on the test persons melody, and we will tests the quality of the melody generation, based on a user test.

We bring along a laptop with the generated melodies and harmonies and the test will be conducted on this laptop. The protocol for the verbal explanation given to the test persons is as follows:

1. Explain that we have generated 4 different harmonisations with and without melody on the basis of different artists from the database and emphasise that they should not be afraid of answering critically

[17]Datalogisk Institut, Københavns Universitet

2. Explain that all generated music will be in the MIDI format, and we would like the test person to neglect the rather unnatural sound.
3. Play the test person's melody without chords in MIDI format and play a tune of the test person's own choice from the database. This is to get the test person used to the MIDI sound.
4. Show how to play the MIDI files.
5. Explain how the different files are created without revealing the actual artists used.
6. Leave the room and let the test person do the test alone. This will take about 30-60 minutes.
7. Reenter the room when the test person is done answering.
8. Explain that Cremo has generated a melody, not on the basis of the user melody, but on the basis of a given number of artists in the database. Again, do not tell the test person which artists are used.
9. Leave test person alone again for 5-15 minutes.
10. Reenter the room and skim the answers and ask a few elaboratory questions. These are of a more general character: "What do you think of Cremo?" and "Do you have any further comments?".
11. The test is finished. Make a short debriefing, telling what difficulties we have had so far and which aspects we have to improve in our own opinion, and ask them if they would like to test the program when we have ameliorated the program on the basis on their answers.

Static parameters

The only parameters we are allowed to change for the individual test persons are the ones for selecting the artists for which the harmonisations and melodic generation are done. But there are some static parameters that we have preset for all test persons. For melodic rock harmonisation we place one chord at each bar on the first beat and for jazz we place two chords at each bar on the first and the second beat. For the harmonisation this is the only static parameter.

	Pitch	Duration
Markov chain order	4	4
Offset	69	1
Seed	154	213

Table 24: Parameters for melody generation

For the melody generation we use a pitch-based Markov chain and a duration-based Markov chain with parameters as seen in Table 24. The order of the chain is chosen to be four for each chain because we found that it gave the best balance between copying directly from the database and creating random notes. The offset and seed parameters are chosen more or less randomly.

Using these static parameters and having decided on a combination of artists, Cremo generates the same harmonisations and melodies each time the program is run. So, using the parameters just discussed we are able to completely recreate the test setup.

B.2 Results

Here we will list the answers of all the respondents. The interpretation and conclusion of the overall results is presented in Section 7. The parameters used for the individual test persons are seen in Table 25

Respondent 1	**Coldplay**	**Muse**	**Radiohead**	**Buckley**	**Mercury Rev**	
Harmonisation 1			×	×		
Harmonisation 2			×		×	
Harmonisation 3				×	×	
Harmonisation 4	×	×	×	×	×	
Melody			×	×		

Respondent 2						
Harmonisation 1		×	×			
Harmonisation 2			×	×		
Harmonisation 3		×		×		
Harmonisation 4	×	×	×	×	×	
Melody		×	×			

Respondent 3	**Miles**	**Parker**	**Coltrane**	**Evans**	**Shorter**	**Monk**
Harmonisation 1	×				×	×
Harmonisation 2	×		×			×
Harmonisation 3	×		×		×	
Harmonisation 4			×		×	×
Melody	×				×	×

Respondent 4						
Harmonisation 1	×		×	×		
Harmonisation 2			×	×		×
Harmonisation 3	×			×		×
Harmonisation 4	×		×		×	
Melody	×		×	×		

Table 25: Parameters for the different harmonisations and melody generations.

Question	**A**	**B**	**C**	**D**	**E**
Q2.1: How does the harmonisation of Cremo sounds without the melody?	No coherence	Some	Good	Very good	
Q2.3: How would you describe the variation of the relation of the chords?	No variation	Some	Medium	Good	Very good
Q2.4: Are the variation of the chords appropriate for the genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q2.5: Are the types of the chord progressions representative for the given genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q2.7: Could you use the generated chords as a source of inspiration for a new melody?	Yes	No	Maybe		
Q2.8: How does the harmonisation of Cremo sound together with the melody?	Chords don't fit	Fit somewhat	Fit well	Fit very well	
Q2.9: Could you use the generated chords as a source of inspiration for new chords for your melody?	Yes	No	Maybe		
Q2.10: Could you use the generated chords directly as a basis for your melody?	Yes	No	Maybe		

Table 26: Questions and answer alternatives for the harmonisation questionnaire

Question	**A**	**B**	**C**	**D**	**E**
Q3.1: Are there aspects of the melody that sounds musical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q3.2: Are there aspects of the melody that sounds unmusical?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q3.4: Can you recognise one or more artists in the generated melody?	Yes	No	Maybe		
Q3.6: Could you use the melody as a source of inspiration?	Yes	No	Maybe		

Table 27: Questions and answer alternatives for the melody questionnaire

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.7**	**Q2.8**	**Q2.9**	**Q2.10**
H1	C	B	D	C	A	B	C	B
H2	B	B	B	B	B	A	A	B
H3	D	C	D	C	A	C	A	C
H4	B	A	C	C	A	B	C	B

	Q3.1	**Q3.2**	**Q3.4**	**Q3.6**
M	C	C	C	B

Table 28: Respondent 1 (melodic rock)

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.7**	**Q2.8**	**Q2.9**	**Q2.10**
H1	D	D	D	D	C	B	B	B
H2	B	E	B	C	B	A	B	B
H3	D	E	B	B	B	B	B	B
H4	A	D	B	B	B	B	B	B

	Q3.1	**Q3.2**	**Q3.4**	**Q3.6**
M	C	C	C	B

Table 29: Respondent 2 (melodic rock)

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.7**	**Q2.8**	**Q2.9**	**Q2.10**
H1	C/D	C	B	C	A	A	B	B
H2	B	D	A	C	A	A	B	B
H3	C/D	D	A	C	A	B	B	B
H4	C/D	D	A	C	A	A	B	B

	Q3.1	**Q3.2**	**Q3.4**	**Q3.6**
M	D	D	A	A

Table 30: Respondent 3 (standard jazz)

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.7**	**Q2.8**	**Q2.9**	**Q2.10**
H1	B	C	B	C	A	A	A	B
H2	C	B	C	C	A	B	A	B
H3	B/C	B	B	B	B	C	C	B
H4	D	D	D	D	A	A	C	B

	Q3.1	**Q3.2**	**Q3.4**	**Q3.6**
M	C	D	A	B

Table 31: Respondent 4 (standard jazz)

C Final user test

C.1 Protocol

The goal of this final user test is to verify that our final system works as described in the synopsis. Just as in the prototype user test, we need to test whether the system is usable to a working musician. That is, we need to answer whether the output of the program can be used as a source of inspiration for developing new compositions or used directly as a new composition. The protocol test proved that genre was recreated somewhat successfully in the harmonisations but that cadences weren't always reasonable. Also noted was that certain artists were clearly recognisable in the generated melodies. Besides this, a couple of programming errors were found based on the test.

We now turn our attention to the program as a whole. Since we now harmonise and alter the melody with the improved harmonisation and the new melody variation, we will focus on these two aspects. We will therefore evaluate output of the harmonisation on the original input melody and on the varied melody.

Preprocessing

Just as the prototype test we need to gather information about the musicians in order to verify their reactions to our system. Test musicians therefore have to answer the preprocessing questionnaire in order to qualify for the evaluation part of this user test. If the musician have already answered the questionnaire during the prototype user test, he doesn't have to do this again. New musicians have to take the preprocessing questionnaire before the evaluation part. Here we restate the preprocessing protocol:

We will send the background questions by email to the musician. These questions will answer what kind of musical education he has and how well he is with the music theory relevant for Cremo. Also, it will answer how well the music in our database is known and if the musician is a composer. On the basis on these questions, we will judge who is capable of evaluating Cremo by the following criteria:

1. Two out of five of the theoretical questions should be answered correctly (from the section "Teoretisk viden"). Though, for the test persons in the standard jazz genre, one of the questions 3-5 should be among the correct answers, since these questions are important for evaluating this genre.
2. Should at least have composed 10 songs
3. For test persons who should test the rock genre, he or she should be able to describe at least 3 of the artists in this genre. For the standard jazz genre, we will require that the test person is able to describe at least 4 of the artists in this genre, since the corpus of the individual artists in this genre is smaller.

If the test person complies with above criteria, we will use his or her melody for the test. The melody can be either a fully composed piece or a subset consisting of minimum 16 bars of melody. Preferably, the format of this piece is either MIDI or a Sibelius file but a PDF notation or a paper form will also work. We will then preprocess the piece and make sure it complies with the format for Cremo: Convert to proper MIDI format and place melody in Channel 1.

Thus, we have the following protocol for the preprocessing of our user test:

1. Send questionnaires to musicians via email and a request for minimum 16 bars of melody.
2. The musicians return the questionnaire by email and attach a melody line (minimum 16 bars) in proper format (MIDI, Sibelius or PDF) or hand us the answers and melody line in paper form.
3. We go through the answers and pick those musicians that match the criteria as defined above.

If the test person complies with the above criteria, we will use his or her melody for the test. The melody can be either a fully composed piece or a subset consisting of minimum 16 bars of melody. Preferably, the format of this piece is either MIDI or a Sibelius file but a PDF notation or a paper form will also work. We will then preprocess the piece and make sure it complies with the format for Cremo: Convert to proper MIDI format and place melody in channel 1.
For this final user test we generate four different harmonisations and four different melody variations using all artists in the given genre. For the harmonisation we have to variable parameters: The quantisation q_h and the analysis duration a:

Harmonisation	q_h	a
H1	1/8	2/4
H2	1/4	2/4
H3	1/8	4/4
H4	1/4	4/4

For the melody variations we fix the harmonisation analysis, such that $q_h = 1/8$ and $a = 2/4$ and introduce the following three parameters: The motif quantisation q_m, motif length l_m and the maximum allowed No Interval threshold t. The parameter set is then given as

Variation	q_m	l_m	t
V1	1/8	2/4	0
V2	1/8	2/4	1
V3	1/4	4/4	0
V4	1/4	4/4	1

The test

The test is carried out as an email correspondence. The email is send to the test person with the generated MIDI harmonisations and variations along with a questionnaire for each MIDI file. The email will explain where to find MIDI files and in which order to listen to them. The supplied questionnaire will be a Word-document, in which the answers can be written directly below the questions. Also attached is a document for writing further comments. The structure of the email can be summarised as follows:

1. Tell the test person that this is the final version of Cremo.
2. Explain that four harmonisations and four variations of the supplied melody have been generated.
3. List the order in which the MIDI's should be played:
    - Harmonisation 1 through 4
    - Variation 1 through 4
4. Explain that the questionnaires are placed in the same folder as the corresponding MIDI files and that answers can be written directly in these Word documents.
5. Supply the deadline for the answers and explain that if any questions or problems can be mailed to us at any time.

The answers to these questions form the final evaluation of Cremo.

Static parameters

The variable parameters listed above are dependent on other parameters in Cremo. These other parameters have to be static for each generated MIDI in order to evaluate the effect of the variable parameters. They are chosen only on the basis of the test generations we made during development, and are therefore not necessarily the most optimal parameters. The following static parameters were set:

Parameter	Value
Max epochs (supernet)	150
Desired error (supernet)	0.5
Hidden units (supernet)	35
Max epochs (subnet)	200
Desired error (subnet)	0.7
Hidden units (subnet)	35
Ignore intervals when pause	Yes
Ignore octaves	Yes

C.2 Results

Question	**A**	**B**	**C**	**D**	**E**
Q1.1: How does the harmonisation of Cremo sound together with the melody?	Chords don't fit	Fit somewhat	Fit well	Fit very well	
Q1.3: Does Cremo's harmonisation support the melody? Is there good tension/release in between chords?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q1.4: How would you describe the variation of the relation of the chords?	No variation	Some	Medium	Good	Very good
Q1.5: Is the variation of the chords appropriate for the genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q1.6: Could you use the generated chords as a source of inspiration for new chords for the melody or as inspiration for a new melody?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q1.7: Could you use the generated chords directly as a basis for your melody?	Yes	No	Maybe		

Table 32: Multiple choice questions and answer alternatives for the harmonisations

Question	**A**	**B**	**C**	**D**	**E**
Q2.1: How does Cremo's variation sound together with the chords?	Melody doesn't fit	Fits somewhat	Fits well	Fits very well	
Q2.3: Does Cremo's melody support the chords? Is there good tension/release?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q2.4: Is the composition of intervals between notes musical with respect to the chords and genre?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q2.5: How would you describe the variation of the melody?	No variation	Some	Medium	Much	Very much
Q2.6: Is the variation of the melody appropriate for the genre?	No variation	Some	Medium	Much	Very much
Q2.7: Could you recognise your own melody?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know
Q2.8: Could you use aspects of the generated variation as a source of inspiration for the original melody or a new compositions?	Not at all	To a lesser extend	To some extend	To a great extend	Don't know

Table 33: Multiple choice questions and answer alternatives for the melodic variations

	Q1.1	**Q1.3**	**Q1.4**	**Q1.5**	**Q1.6**	**Q1.7**
H1	B	C	C	B	B	B
H2	B	C	B	C	C	C
H3	C	C	C	C	D	C
H4	A	A	D	A	A	B

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.6**	**Q2.7**	**Q2.8**
V1	B	B	B	D	B	B	B
V2	A	A	A	D	A	A	A
V3	A	B	B	C	B	B	A
V4	B	C	B	C	C	C	C

Table 34: Respondent 1 (melodic rock)

	Q1.1	**Q1.3**	**Q1.4**	**Q1.5**	**Q1.6**	**Q1.7**
H1	A	A	C	C	A	B
H2	A	A	C	C	A	B
H3	A	A	C	C	A	B
H4	A	A	D	B	B	B

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.6**	**Q2.7**	**Q2.8**
V1	B	B	B	D	B	A	B
V2	C	C	B	C	C	B	C
V3	B	C	C	D	C	A	C
V4	A	A	B	C	C	A	A

Table 35: Respondent 2 (melodic rock)

	Q1.1	**Q1.3**	**Q1.4**	**Q1.5**	**Q1.6**	**Q1.7**
H1	B	B	C	A	B	B
H2	B	B	C	A	C	B
H3	B	B	B	B	C	B
H4	B	C	C	C	C	B

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.6**	**Q2.7**	**Q2.8**
V1	D	C	E	C	E	A	C
V2	D	C	E	D	E	A	C
V3	C	B	E	B	E	A	B
V4	D	C	E	D	E	A	C

Table 36: Respondent 3 (standard jazz)

	Q1.1	**Q1.3**	**Q1.4**	**Q1.5**	**Q1.6**	**Q1.7**
H1	C	C	B	C	C	B
H2	B	B	B	C	D	B
H3	A	A	C	C	B	B
H4	A	A	C	C	B	B

	Q2.1	**Q2.3**	**Q2.4**	**Q2.5**	**Q2.6**	**Q2.7**	**Q2.8**
V1	A	A	A	D	A	A	A
V2	A	A	A	D	A	A	A
V3	A	A	B	D	A	A	A
V4	B	B	B	B	C	A	B

Table 37: Respondent 4 (melodic rock)

References

[1] Allan, M., and Williams, C. K. I. Harmonising chorales by probabilistic inference.

[2] Biles, J. A. Genjam: A genetic algorithm for generating jazz solos. *International Computer Music Conference* (1994), 131–137.

[3] Bishop, C. M. *Pattern Recognition and Machine Learning.* Springer Science, 2006.

[4] Boling, M. E. *Jazz Theory Workbook.* Advanced Music, 1993.

[5] Cope, D. *Computers and Musical Style.* A-R Editions, Inc., 1991.

[6] Cope, D. *Experiments in Musical Intelligence.* A-R Editions, Inc., 1996.

[7] Cope, D. *The Algorithmic Composer.* A-R Editions, Inc., 2000.

[8] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[9] Eck, D., and Schmidhuber, J. A first look at music composition using lstm recurrent neural networks. Tech. Rep. IDSIA-07-02, Instituto Dalle Molle di studi sull' intelligenza artificiale, 2002.

[10] Grimmett, G., and Stirzaker, D. *Probability and Random Processes*, 3rd ed. Oxford University Press, 2001.

[11] Hiller, L. A., and Isaacson, L. M. *Experimental Music.* McGraw-Hill Book Company, 1959.

[12] Hudak, P. *The Haskell School of Expresssion.* Cambridge University Press, 2007.

[13] Hudak, P. The haskell school of music. http://plucky.cs.yale.edu/cs432/HSoM-V0.14.pdf, September 9, 2009.

[14] Hörnel, D., and Menzel, W. Learning musical structure and style with neural networks. *Computer Music Journal 22*, 4 (Winter 1998), 44–62.

[15] MMA. Midi 1.0 detailed specification: Document version 4.1.1. "`http://www.midi.org/about-midi/specinfo.shtml`", February 1996.

[16] Mozer, M. C. Neural network music composition by prediction: Exploring the benefits of psychoacoustic contraints and multiscale processing. *Connection Science* (1994).

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

[18] Papadopoulos, G., and Wiggins, G. Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *In AISB Symposium on Musical Creativity* (1999), pp. 110–117.

[19] Papadopoulos and Geraint Wiggins, G. A genetic algorithm for the generation of jazz melodies. *SteP'98* (1998).

[20] Papadopoulos and Geraint Wiggins, G., Phon-Amnuaisuk, S., and Tuson, A. Evolutionary methods for musical composition. *International Journal of Computing Anticipatory Systems* (1991).

[21] Rawlins, R., and Bahha, N. E. *Jazzology - The Encyclopedia of Jazz Theory for All Musicians.* Hal Leonard, 2005.

[22] Skovenborg, E., and Arnspang, J. Extraction of structural patterns in popular melodies. *Proceedings of Computer Music Modeling and Retrieval 2003, LNCS 2771* (2004), 98–113.

[23] Thielemann, H. Audio processing using haskell. *7th International Conference on Digital Audio Effects* (October 2004).

[24] Vercoe, B. The canonical csound reference manual: Document version 5.10. "`http://www.csounds.com/manual/`", February 2008.

[25] Walder, C. Automatic chorale harmonisation via local learning and dynamic programming. 2009.